A Data Serialization-based Framework for Efficient IoT Management

Khayal Huseynov*, Lal Verda Cakir‡, Sarah Al-Shareeda†*, Mehmet Özdem§ and Berk Canberk‡†
*BTS Group, Turkey
†Department of AI and Data Engineering, Istanbul Technical University, Turkey
‡School of Computing, Engineering and The Built Environment, Edinburgh Napier University, UK
§Türk Telekom, Turkey
khayal.huseynov@btsgrp.com, {alshareeda, canberk}@itu.edu.tr, {lal.cakir, b.canberk}@napier.ac.uk
mehmet.ozdem@turktelekom.com.tr

***Abstract*—The Internet of Things (IoT) management relies on the efficient and timely transfer of data from sensors to applications. Processing required data transformations at the edge gateway introduces spatial complexity issues, particularly concerning resource constraints and latency requirements. By adopting a zero-copy binary format, Flatbuffers, we reduce the spatial complexity of processing at IoT edge gateway. However, at the service applications, interoperability challenges may arise when dealing with binary data formats compared to text-based formats. To accommodate the constraints of IoT edge gateways and service applications, we introduce a framework that aims to improve the data exchange rate between the IoT layer and the management layer while maintaining interoperability. Our comparative analysis across three scenarios, involving single and multiple sensors, shows that the proposed FlatBuffers-based framework outperforms the conventional JSON and Protocol Buffers formats in terms of frequency. These findings highlight the significant potential of FlatBuffers in boosting the real-time interaction capabilities of IoT systems.**

***Index Terms*—Internet of Things (IoT), Data Serialization, Serialization Format, Edge Computing**

I. Introduction

Internet of Things technologies have revolutionised how the physical and virtual worlds work together. With this, IoT sensors started producing massive amounts of data to serve the demands of various applications. In recent years, Digital Twin (DT) technology has emerged, accelerating this demand even more and requiring real-time characteristics [1]. However, efficient and instant communication in resource-constrained environments to support such applications remains a significant challenge.

To counteract that, employing processing techniques such as compression before transmission has been explored [2]. However, the IoT sensors' limited computational capabilities may prevent such local processing, leading to processing delays. Furthermore, these IoT sensors are often deployed in remote areas without reliable network connectivity, resulting in intermittent data transmission, loss, and increased latency. As a result, this hinders the efficiency of the IoT management [3].

IoT edge gateways serve as a bridge connecting numerous IoT sensors and performing required data transformations. Here, syntactic interoperability issues arise when integrating diverse sensor types, necessitating a unified data format to simplify data sharing [4]. On top of this, supplying continuous or periodic data brings the challenge of depleting the gateway's resources and extending processing latencies. Coupled with the interoperability challenges, this load may lead to gateway memory potentially reaching full capacity, preventing its ability to process new data. Considering these, it is evident that the challenges of space and time efficiency and syntactic interoperability have to be addressed in IoT management.

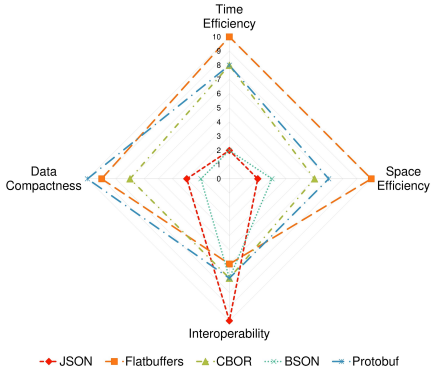


Fig. 1: Comparison of Serialization Formats based on [5]–[8]

To answer these challenges, data serialization techniques offer a promising approach. By reducing the size of the data, they improve the latency encountered in the network. Moreover, they provide compatibility across different devices and platforms which can enable interoperability. JavaScript Object Notation (JSON) has been widely used due to its flexibility, which is owed to formatting the data in human-readable name-value pairs. However, this requires the data to be parsed in character bases, resulting in higher memory and time requirements. To overcome this, binary formats have been developed: Protocol Buffers (Protobuf), Concise Binary Object Representation (CBOR), Binary JSON (BSON), and Flatbuffers. However, each formatting comes with different space and time complexities based on how the serialisation, deserialization and parsing are performed. As shown in Figure 1, Flatbuffers is a strong candidate to answer the needs

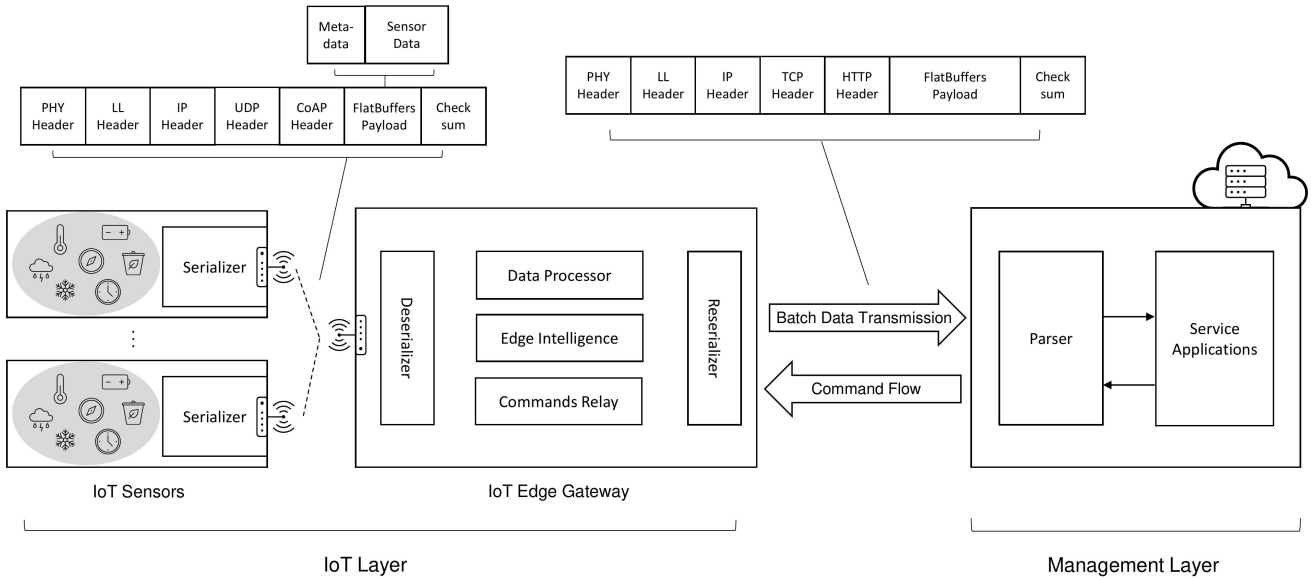


Fig. 2: The Proposed IoT Management Architecture

of IoT management and in larger systems like Digital Twins, which rely heavily on accurate and timely data. Considering this, we explore data serialisation in IoT management and introduce a new framework shown in Figure 2. Consequently, we contribute to the literature as below.

- To improve the efficiency in IoT management, we employ a zero-copy data format using Flatbuffers, enabling the payload size to be reduced by 50% on average compared to human-readable formats. Then on the IoT edge gateway, we create an improved data pipeline by deserialization, processing and reserilization at the IoT edge gateway. This allows the latencies to be reduced while allowing data transformations and intelligence at the edge.
- We create a Parser interface in the Management Layer to ensure compatibility with a wide range of service applications. Here, we transform FlatBuffers into JSON without incurring additional latency at the IoT layer while answering the syntactic interoperability needs.
- We perform empirical experiments on the proposed framework under three scenarios. These reveal that the proposed framework can provide 8.7% improvement on the cycles per second. Lastly, we provide insights into the ongoing discussion on incorporating edge computing into IoT applications.

The rest of the paper is organized as follows: Section II surveys the available literature regarding serialization formats. The preliminary description of our proposed architecture is presented in Section III. In Section IV, the performance of the proposed architecture is evaluated, and the efficiency of FlatBuffers compared to JSON and Protocol Buffers is discussed. Finally, the study is concluded with a future work outline in Section V.

II. Literature Review

The data serialization approaches work to minimize processing time and ensure that the data transfer is efficient. This process converts complex data structures into a streamlined, transportable format. This enables reducing the payload size considerably and thus the processing time [9].

Data serialisation techniques have played a key role in developing digitalized technologies. The Extensible Markup Language (XML), a schema-driven text-based format, established the foundation for web data exchange since its development in the late 1990s. Following the XML, the schema-less JavaScript Object Notation (JSON) format used name and value pairs, simplifying the text-based data format. Although JSON provides a readable and convenient data serialization format, its dynamic-typing serialization structure results in runtime inefficiency. This is because the JSON requires the data to be parsed character by character, leading to increased serial and deserialization delays. To overcome this, Google introduced Protocol Buffers (Protobuf) in the late 2000s, utilizing binary encoding in serialization. In this approach, the name part of the data is separately held at both ends and values are sent in a binary format and accessed via positional

binding. Reducing the size of the communicated payload and the parsing mechanism allowed faster operations, reducing the latency. Following this, the Binary JSON (BSON) and schema-less Concise Binary Object Representation (CBOR) have been included in the name part of the sent data. This provided more flexibility while compromising on efficiency. Lastly, FlatBuffers, a schema-driven format developed in 2014 by Google, has advanced the deserialisation by allowing direct access to serialized data without parsing or unpacking, significantly reducing latency [5], [7].

There are many serialization formats, each offering different levels of efficiency and flexibility. Therefore, the data serialization format should be selected based on the specific requirements and constraints of the IoT environment [10]. As highlighted in [5], [7], JSON, CBOR, and BSON offer a moderate performance level, providing efficient resource management. However, FlatBuffers and Protobuf are identified as more advantageous formats due to their specific traits. Protobuf is distinguished by its time efficiency thanks to reduced message size and enhanced serialization speed. Conversely, FlatBuffers is acclaimed for its space efficiency and expedited deserialization process. This is achieved by zero-copy deserialization through a compact binary format that enables direct memory mapping without additional data structure allocation, or parsing [9]. Data serialisation techniques employed in IoT Management require space and time efficiency to enable on-the-fly processing and work in resource-constrained environments. The findings in [6] highlight the potential of Flatbuffers to enhance communication efficiency and performance for data exchange processes in real-time applications. Additionally, Flatbuffers has been utilised in other areas such as data center communication [11] and control plane messages [12] to great success. Considering these, FlatBuffers stands out as a worthy candidate for enabling efficient IoT management.

Given these attributes, we utilize FlatBuffers in our proposed framework to ensure minimal overhead and to optimize communication efficiency in terms of frequency.

III. Proposed IoT Management Architecture

The proposed IoT management architecture as illustrated in Fig. 2 comprises of the *IoT Layer* and the *Management Layer*.

A. IoT Layer

1) Serializer: The IoT sensors are equipped with microcontrollers that poll and collect environmental data from onboard sensors. A pre-compiled code is deployed to these microcontrollers to handle the formatting and data structure. Using this, the sensor data in the memory is serialized into FlatBuffers and transmitted to the *IoT Edge Gateway*. An example of temperature sensor payload data structure is given in Listing 1 along with a matching JSON counter-example in Listing 2. While these examples of the schema-defined data structure present have similar sizes, FlatBuffers encodes this into a binary format. Conversely, when JSON-encoded data is used, the exemplified textual delimiters and data keys are included within the payload. Because of this, the Flatbuffers offer a more efficient memory layout and access as well as reduced payload size.

```
namespace Sensors;
table Metadata {
  id: string;
  location: string;
  type: string;
}
table Reading {
  value: float32;
  timestamp: int64;
}
table TemperatureSensor {
  metadata: Metadata;
  readings: [Reading];
}
root_type TemperatureSensor;
```

Listing 1: Example of FlatBuffers Data Schema for Temperature Sensor Data.

```
{
  "TemperatureSensor": {
    "metadata": {
      "id": "string",
      "location": "string",
      "type": "string"
    },
    "readings": [
      {
        "value": "float32",
        "timestamp": "int64"
      }
    ]
  }
}
```

Listing 2: Example of JSON Data Schema for Temperature Sensor Data.

2) Deserializer: The IoT Edge Gateway receives serialized data over the wireless medium and deserializes it for further processing at the *Data Processor*. Thanks to the zero-copy characteristic of FlatBuffers, this process is without additional memory allocations, enabling faster operations and lower memory usage.

3) Data Processor and Edge Intelligence: Following the deserialization, the *IoT Edge Gateway* initiates data processing tasks. These may include filtering, compression, aggregation, and edge intelligence [13]. By performing these operations at the edge, the network efficiency can be enhanced by reducing packet sizes and thus conserving bandwidth, ultimately lowering transmission costs.

4) Reserailizer: After the processing tasks are performed, the refined data is reserialized to be transmitted to the *Management Layer*.

5) Commands Relay: This component directs commands from higher-level *Service Applications* to the IoT sensors. With this, the IoT sensors are instructed to perform actions or adjustments. Upon receiving commands, the *Commands Relay* translates and formats them into instructions compatible with the communication protocols and formatting supported by the sensor devices. While these may be hard-coded into the IoT Edge Gateway, they also can be translated by using Yet Another Next Generation (YANG) data models as in [14].

B. Management Layer

1) Parser: At the *Parser*, sensor data coming from multiple *IoT Edge Gateways* are concatenated. This enables effective interpretation and utilization by *Service Applications*. For this mapping to occur, sensor data must once again be deserialized. After mapping, this data is serialized back into JSON. This conversion to JSON enables compatibility with a broader range of *Service Applications*, ensuring seamless integration and syntactic interoperability within the IoT ecosystem.

2) Service Applications: Service Applications derive insights from sensor data and facilitate predictive maintenance, optimization, and decision-making processes. These processes generate actionable commands which are then communicated back to the *IoT Sensors* via the *IoT Edge Gateway*, enabling closed-loop control.

IV. Performance Evaluation and Results

A. Testbed Implementation and Performance Metrics Definition

The proposed IoT communication framework is tested using an environment consisting of Virtual Machines (VMs) hosted within the same private cloud infrastructure. This setup simulates the real-world configuration with three types of workloads, delineated in Section III. Here, first VM type generates synthetic *IoT Sensor* data. Then, the second VM serves as *IoT Edge Gateway*. This VM is designed to capture sensor data periodically. Finally, third VM plays the role of *Parser* where Protobuf and FlatBuffers scenarios' payloads are parsed into JSON.

For serialization of the data into JSON, Protocol Buffers, and FlatBuffers formats, Go programming language has three widely recognized and native open-source libraries [15]–[17]. As such, the client-server architecture is developed in Go latest stable version 1.22. In this testbed setup, computational resources include an Intel(R) Xeon(R) E7-4860 processor at 2.27GHz, with each VM provisioned with one vCPU and 1GiB RAM to accurately reflect the computational capabilities anticipated in real-world IoT environments. The Rocky Linux 9 operating system is used on VMs. The benchmarks are done by evaluating the code across 50,000 iterations under three scenarios shown in Table I:

1) In Scenario 1, a single IoT sensor is interfaced with the *IoT Edge Gateway*, utilizing 4901 data objects comprising $1\ string\ +\ 1\ int32$.
2) Scenario 2 is setup to include multiple IoT sensors connected to the *IoT Edge Gateway*, testing 4901 data objects: $1\ string\ +\ 6\ int32$.
3) Scenario 3 is the same as Scenario 2 but it uses floating point values, testing 4901 data objects: $1\ string\ +\ 6\ float32$.

TABLE I: Scenario Parameters

Number	Name	Payload Object Count	Payload Object Type
1	Basic	4901 objects	1 string, 1 int32
2	Integer	4901 objects	1 string, 6 int32
3	Floating	4901 objects	1 string, 6 float32

To evaluate the efficacy of the framework in managing diverse sensor configurations and complexities of data, this study employs five critical performance metrics:

- Total Payload Size (KB): the total payload size generated by the sensors and submitted by the edge gateway to the parser.
- Total Serialization Time T_S (ms): the cumulative duration required for data serialization at the sensor and the subsequent reserialization at the edge gateway.
- Total Round-Trip Time T_{RTT} (ms): the duration of data transmission from the sensors to the gateway, then to the management layer, and the returns of acknowledgements in the reverse path.
- Total Deserialization Time T_D (ms): the time taken at the edge for deserializing the received data from the sensors and the time taken at the management layer to parse and deserialize the communicated aggregated data.
- Frequency F (cycle/s): the reciprocal of the end-to-end time which is the sum of serialization time T_S, round-trip time T_{RTT}, and deserialization time T_D:

$$F = \frac{1}{T_S + T_{RTT} + T_D} \tag{1}$$

Collectively, these indicators provide a comprehensive assessment of the system's performance, facilitating an understanding of the framework's efficiency within the context of cyberphysical communications.

B. Results and Discussion

Regarding the performance indicators payload size, T_S, T_{RTT}, T_D, and F, our results indicate that the proposed architecture significantly enhances the efficiency of the communication for both single and multiple sensor scenarios across all object types.

1) Scenario 1's single sensor benchmark involving single-string and integer data objects: FlatBuffers showed a payload size approximately 10.29% larger than JSON and 20.5% larger than Protocol Buffers, measuring 19,684 bytes compared to 17,846 bytes and 15,648 bytes respectively. This difference may be attributed to the data being sparse, meaning

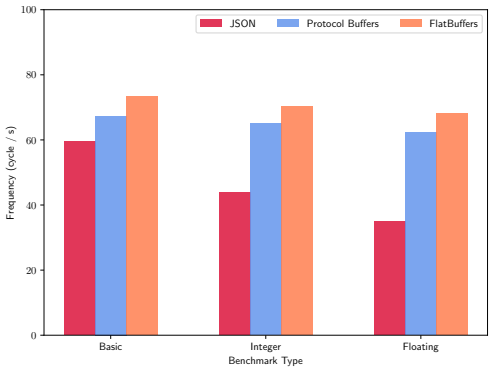
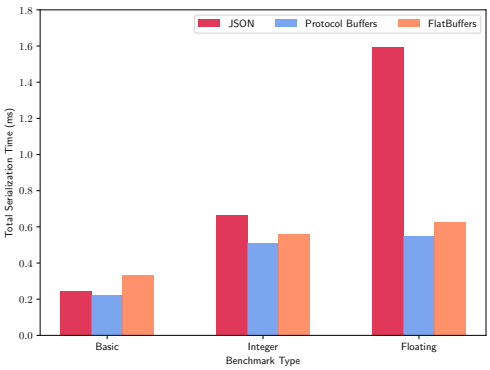


Fig. 3: Frequency performance of the scenarios

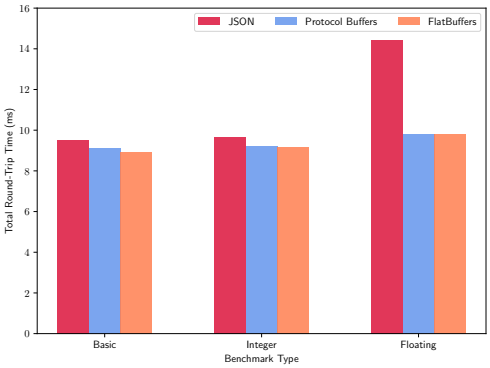
there are not a lot of fields in each object. Despite the expected slower T_S, FlatBuffers demonstrated a substantial increase in deserialization efficiency, reducing both T_{RTT} and T_D. This improvement in deserialization time contributed to a frequency of 73.39 cycle/s, surpassing JSON's 59.49 cycle/s and Protobuf's 67.29 cycle/s. These findings underscore the benefits of FlatBuffers in enhancing deserialization efficiency and optimizing the overall frequency.

2) Scenario 2's multiple sensors and integer benchmark: Here, binary formats have a significantly reduced payload size compared to JSON's 70,315 bytes. For serialization time T_S, FlatBuffers outperform JSON by reducing the serialization duration by 15.19%, recording a time of 562,274 ns in comparison to JSON's 663,194 ns. Moreover, FlatBuffers surpass JSON in T_{RTT}, showing a 5.27% improvement. Notably, the deserialization time T_D with FlatBuffers is significantly reduced, 63.7% lower than JSON's and 25.1% lower than Protobuf's. These enhancements collectively result in an improved frequency F for FlatBuffers, which achieve 70.23 cycles/s. This performance analysis demonstrates FlatBuffers' considerable advantages in handling multiple sensor data, particularly highlighting its superiority in reducing deserialization time which becomes critical for IoT ecosystems with high sensor counts and data throughput.

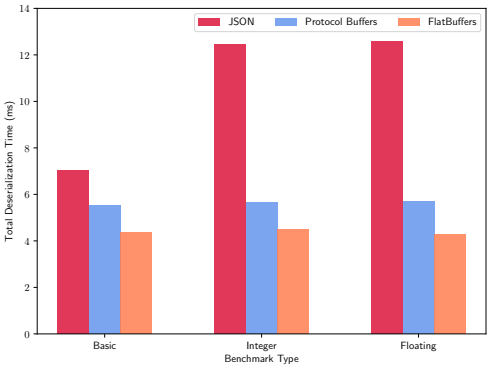
3) Scenario 3's multiple sensors and floating point benchmark: The improvements exhibited in Scenario 2's integer benchmark encourage testing a different benchmark under the same setting of multiple sensors. Here, FlatBuffers diminish the payload size to 33,732 bytes from JSON's 85,939. This reduction is complemented by a notable acceleration in serialization time T_S, where both binary formats are more than twice as fast as JSON. Although Protobuf has an overall better payload size and serialization time, the T_{RTT} and T_D highlights FlatBuffers' superiority just as in Scenario 2. This faster transmission is crucial for systems requiring prompt data exchanges. For deserialization, T_D of FlatBuffers marks 32.9% improvement over Protobuf, a critical factor for efficiently handling complex datasets. Consequently, the Frequency F of FlatBuffers is the highest among three data types. These results highlight FlatBuffers' significant contribution to



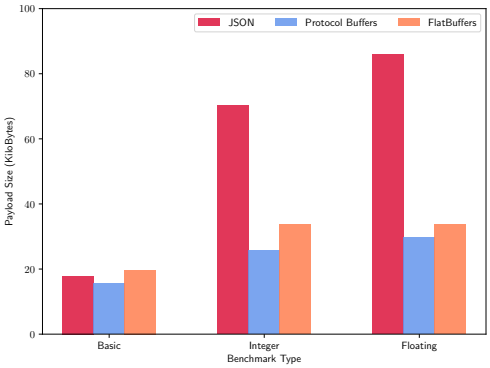
(a) T_S (ms)



(b) T_{RTT} (ms)



(c) T_D (ms)



(d) Payload Size (KB)

Fig. 4: Performance comparison of the scenarios

enhancing the framework in data-intensive IoT environments where precision of the data points are also important.

To summarize the analysis, Fig. 4 visually showcases binary formats' superiority over text-based formats such as JSON in handling IoT communications across single and multiple sensor scenarios. While Protocol Buffers has an overall lower payload size, the zero-copy nature of FlatBuffers concisely captures the frequency gains it offers over JSON and even Protocol Buffers in the IoT communication framework as shown in Figure 3. Protocol Buffers offers a slightly reduced payload size and faster serialization time out of three formats. However, FlatBuffers' substantial improvements in deserialization times (28.29%) enhances data processing speeds. As such, FlatBuffers comes ahead with a 8.7% increase in frequency F compared to second-best Protobufs. Furthermore, these evaluations provide evidence that the trade-offs of the FlatBuffers to JSON or Protocol Buffers to JSON parsing at the management layer are negligible when overall performance improvement is concerned.

V. Conclusion and Extensions

This study introduced a framework to enhance communication between IoT and management layers, leveraging FlatBuffers for serialization to optimize the IoT edge gateway. Our findings show that the proposed architecture outperforms traditional JSON and Protocol Buffers in handling simple and complex data, improving deserialization times and overall frequency by 28.29% and 8.7%, respectively.

Future directions include exploring other serialization formats and integrating the framework into machine learning applications for edge intelligence. This integration aims to address computational challenges of machine learning in microcontroller environments, thereby expanding the framework's utility and efficiency in IoT ecosystems.

Acknowledgments

This work was supported by the ITU Rektorlugu Bilimsel Arastirma Projeleri Birimi (BAP) Fund Grant Number 43981, Abu Dhabi National Oil Company (ADNOC), Emirates NBD, Sharjah Electricity Water & Gas Authority (SEWA), Technology Innovation Institute (TII) and GSK as the sponsors of the 4^{th} Forum for Women in Research (QUWA): Sustaining Women's Empowerment in Research & Innovation at University of Sharjah and The Scientific and Technological Research Council of Turkey (TUBITAK) 1515 Frontier R&D Laboratories Support Program for BTS Advanced AI Hub: BTS Autonomous Networks and Data Innovation Lab Project 5239903

References

[1] E. Ak, K. Duran, O. A. Dobre, T. Q. Duong, and B. Canberk, "T6conf: Digital twin networking framework for ipv6-enabled net-zero smart cities," *IEEE Communications Magazine*, vol. 61, no. 3, pp. 36–42, 2023.

[2] A. Karnaukhov, A. Idelevich, A. Rolich, and L. Voskov, "Data compression strategies for enhancing iort communications over heterogeneous terrestrial-satellite networks," in *2023 XVIII International Symposium Problems of Redundancy in Information and Control Systems (REDUNDANCY)*, 2023, pp. 189–193.

[3] Y. B. Zikria, R. Ali, M. K. Afzal, and S. W. Kim, "Next-generation internet of things (iot): Opportunities, challenges, and solutions," *Sensors*, vol. 21, no. 4, p. 1174, 2021.

[4] S. Al-Shareeda, K. Huseynov, L. V. Cakir, C. Thomson, M. Ozdem, and B. Canberk, *Digital Twins for 6G: Fundamental theory, technology and applications*. IET, 2024, ch. AI-based Traffic Analysis in Digital Twin Networks.

[5] A. Sumaray and 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*, 2012, pp. 1–6.

[6] M. A. Pradana, A. Rakhmatsyah, and A. A. Wardana, "Flatbuffers implementation on mqtt publish/subscribe communication as data delivery format," in *2019 6th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, 2019, pp. 142–146.

[7] D. Friesel and O. Spinczyk, "Data serialization formats for the internet of things," *Electronic Communications of the EASST*, vol. 80, 2021.

[8] M. Noura, M. Atiquzzaman, and M. Gaedke, "Interoperability in internet of things: Taxonomies and open challenges," *Mobile Networks and Applications*, vol. 24, no. 3, pp. 796–809, Jun 2019. [Online]. Available: https://doi.org/10.1007/s11036-018-1089-9

[9] 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)*. IEEE, 2020, pp. 10–18.

[10] E. Al-Masri, K. R. Kalyanam, J. Batts, J. Kim, S. Singh, T. Vo, and C. Yan, "Investigating messaging protocols for the internet of things (iot)," *IEEE Access*, vol. 8, pp. 94 880–94 911, 2020.

[11] T. Zhang, H. Zhou, C. Huang, C. Tian, W. Zhang, X. Wang, Y. Wang, A. M. Abdelmoniem, M. Tan, W. Dou *et al.*, "Achieving zero-copy serialization for datacenter rpc," in *2023 IEEE International Performance, Computing, and Communications Conference (IPCCC)*. IEEE, 2023, pp. 304–312.

[12] M. Ahmad, S. U. Jafri, A. Ikram, W. N. A. Qasmi, M. A. Nawazish, Z. A. Uzmi, and Z. A. Qazi, "A low latency and consistent cellular control plane," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 648–661.

[13] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[14] L. V. Cakir, T. Bilen, M. Özdem, and B. Canberk, "Digital twin middleware for smart farm iot networks," in *2023 International Balkan Conference on Communications and Networking (BalkanCom)*, 2023, pp. 1–5.

[15] "json package," Google. [Online]. Available: https://pkg.go.dev/encoding/json, accessed on Feb 19th, 2024

[16] "Protocol buffers - google's data interchange format," Google. [Online]. Available: https://github.com/protocolbuffers/protobuf, accessed on Feb 20th, 2024

[17] "Flatbuffers: Memory efficient serialization library," Google. [Online]. Available: https://github.com/google/flatbuffers, accessed on Feb 19th, 2024