ADVANCES IN CYBER-PHYSICAL SYSTEMS

Vol. 9, No. 2, 2024

IMPACT OF SERIALIZATION FORMAT ON INTER-SERVICE LATENCY

Eduard Maltsev[1]***, Riaz Ul Amin***[2]

[1]*Lviv Polytechnic National University, 12, Bandera Str, Lviv, 79013, Ukraine,*
[2]*Edinburgh Napier University, Merchiston Campus,*
10 Colinton Rd, Edinburgh, EH10 5DT, United Kingdom.
Authors' e-mails: *eduard.y.maltsev@lpnu.ua, riazulamin@mail.uk*

https://doi.org/10.23939/acps2024.02.

Submitted on 25.08.2024

Abstract**: This study provides an evaluation of the impact of various serialization formats on inter-service communication performance, with a focus on serialization speed, space efficiency, and latency in environments integrating middleware, which are characteristics of microservice architectures. Through an empirical analysis of a wide range of serialization formats and comparison to the traditional standards, it highlights that the compactness of serialized payloads is more critical in reducing end-to-end latency than the sheer speed of serialization itself. Despite their high serialization speeds, FlatBuffers and Cap'n Proto underperform in distributed settings, in contrast to the more balanced performance seen with Avro, Thrift, and Protobuf. This study underscores the importance of message size optimization in boosting network efficiency and throughput.**[1]

Index Terms**: Data communication, Encoding, Information exchange, Protocols, Performance evaluation.**

I. INTRODUCTION

The ever-growing complexity of software systems has necessitated a paradigm shift towards microservices architectures [2,4]. These architectures decompose functionalities into independent, loosely coupled services that communicate to achieve a unified goal. Inter-service communication (ISC) acts as the lifeblood of distributed systems, facilitating the seamless exchange of data between services. However, the efficiency of this exchange directly impacts the overall performance, scalability, and resource utilization of the entire system [3].

One critical element influencing ISC efficiency is the choice of serialization format. Serialization refers to the process of transforming data structures into a transmittable format, allowing data to traverse network boundaries. The receiving service then deserializes the data back into its original form [5]. Compact serialization formats minimize the size of the transmitted data, leading to reduced bandwidth usage and improved network performance. However, this often comes at the cost of increased processing overhead for serialization and deserialization [23].

[1] This article uses the materials and results obtained by the authors during the research work "Intelligent design methods and tools for the modular autonomous cyber-physical systems," state registration number 0124U002340 dated 09.03.2024, which is carried out at the Department of Electronic Computing Machines of the Institute of Computer Technologies, Automation and Metrology of Lviv Polytechnic National University in 2024-2028.

II. LITERATURE REVIEW AND PROBLEM STATEMENT

Let us examine recent studies to understand the current landscape of serialization formats and their implications for inter-service communication in distributed systems. Research in [1] contrasts JSON/XML with Protobuf for data serialization in web services, emphasizing efficiency, readability, and schema enforcement. JSON/XML is preferred in REST for its text-based, human-readable formats, enabling dynamic, schema-less data interchange. Another interesting study in [2] focuses on optimizing inter-service communication in a cloud-native microservice architecture. The study presents Protocol buffers as a language-neutral, platform-neutral, extensible mechanism for serializing structured data.

The study [3] compares various serialization formats, focusing on vehicle-to-cloud communication. The paper evaluates Protobuf and Flatbuffers, two binary serialization formats. It mentions Cap'n Proto as an attractive zero-copy format, which performs similarly to Flatbuffers but with a slight speed advantage. Another alternative mentioned is MessagePack.

Source [4] explores binary versus textual serialization formats for inter-service communication within Java microservices under a K-Native, Kubernetes-managed environment. The study suggests that Protocol Buffers significantly improve response time and payload size performance.

Source [5] evaluates different serialization protocols for improving inter-service communication efficiency within dCache, a distributed storage system.

Source [6] evaluates diverse data serialization formats. The gap in this research is its focus on microcontrollers with certain constraints, which may not be generalizable to all IoT devices or distributed systems.

Source [7] evaluates serialization formats' efficiency and performance in distributed systems, focusing on IoT sensor networks. Protocol Buffers or Apache Thrift were the most efficient means of encoding information based on the information provided.

Source [8] evaluates several schema-driven and schema-less binary serialization specifications that are JSON-compatible, including but not limited to ASN.1, Apache Avro, Microsoft Bond, Cap'n Proto, FlatBuffers, and others.

Complementing these findings, [9] evaluates the performance impact of different communication protocols (REST, gRPC, and Thrift) in microservices, focusing on network, CPU, and memory utilization alongside response times. Thrift and gRPC outperformed REST based on response time and system resource efficiency, attributed to their compact binary serialization formats and efficient protocol designs.

The study [10] investigates JSONBinPack's efficiency, particularly in schema-driven mode, and directly aligns with our research. The study suggests that JSONBinPack outperforms traditional JSON and binary serialization formats based on space efficiency.

Research in [11] provides comparative experiments involving HDVM, Redis, and Protobuf for JSON data serialization, assessing performance metrics to demonstrate Protobuf's efficiency.

The paper [12] focuses on Apache Arrow and its Arrow Flight protocol.

The research [13] details the impact of SOAP serialization on communication efficiency, particularly in web services using HTTP and JMS protocols. The study's limitations include not considering the effect of network conditions, not testing other serialization formats like JSON or Protocol Buffers, and focusing only on SOAP messages.

Research conducted in [14] explores the efficiency of serialization formats in distributed systems, focusing on IoT devices.

The study [15] evaluates a wide range of JSON-compatible binary serialization formats. Schema-driven specifications, especially ASN.1 PER Unaligned and Apache Avro (unframed) are identified as the most space-efficient.

Additionally, for our research, it is essential to understand the inner workings of various optimized formats, like Protocol Buffers [16], to understand better the scenarios they are suited for [17].

A recent study [18] highlighted that exploring alternative web archival formats, specifically Parquet and Avro, demonstrated significant performance improvements over the traditional WARC format. The study [19] emphasizes the unique advantages of HatRPC's hint-accelerated approach in optimizing Thrift RPC services over RDMA transport.

Findings in [20] suggest that Cap'n proto is faster than Flatbuffers in serialization/deserialization time. Source [21] suggests that MessagePack (MsgPuck) excels and beats other libraries with formats like, e.g., Flatbuffers and NanoPB (Protobuf). It has been suggested that special hardware like FPGAs can be used, as it has mentioned in [22]. To conduct benchmarks both for serialization/deserialization speed and distributed delay measurements, we will use the model and part of the methodology we have designed in the study [23]. The study [24] underlines the importance of efficient resource utilization for improving system performance.

Considering the gap in current literature and the rapid evolution of serialization technologies, we aim to provide new insights into the influence of serialization format choice on inter-service communication latency. This study will examine the feasibility of achieving a 50 % reduction in latency compared to JSON.

III. SCOPE OF WORK AND OBJECTIVES

The objective of this study is to assess whether the latency of inter-service communication can be decreased by at least 50 %, compared to JSON, by utilizing alternative binary and textual serialization formats. To achieve this, we will break the objective into smaller parts. Firstly, we will measure serialization/deserialization speed to better understand better the influences of constituent parts underlying the overall latency metric. Then, we will conduct a distributed benchmark to measure latency for each selected format in a distributed environment. The last step is to perform a comparative analysis of the gathered metrics and draw conclusions.

IV. METHODOLOGY

For the serialization and deserialization speed benchmarks, we will present two separate results tables: the first one showing results from epochs 1 through 15 and the second from epochs 16 to 30. The first test will operate with smaller message sizes from ~0.5 kB to ~4 kB, and the second benchmark will have message sizes from ~4 kB to ~30 kB.

The distributed benchmark will be conducted for the first 15 epochs and will feature one results table, message sizes from ~0.5 kB to ~4 kB, number of messages, N_e, is gradually increasing with each epoch, and is determined as $N_e = 2000 \times e$, where e is an epoch number. The median reduction $R_{f,m}$ in comparison with JSON is calculated as follows:

$$R_{f,m} = \frac{V_{f,m} - V_{json,m}}{V_{json,m}} , \qquad (1)$$

where m is a metric in the set {serialization speed s, deserialization speed d, latency l}, $V_{json,m}$ is the median value of the metric m for JSON and $V_{f,m}$ represents the median value of the metric m for a given serialization format f. Results will be presented in descending order by $R_{f,m}$.

V. SERIALIZATION SPEED BENCHMARK

Let us have a look at the serialization speed benchmark results presented in Table 1 and Table 2. Flatbuffers format consistently outperforms other protocols at all message sizes in this benchmark, maintaining a median below 2 µs/op across all tested message sizes. This finding contradicts the results presented in [3], which suggests that Protobuf has a faster serialization speed.

Table 1

Serialization time distribution (µs/op), epoch 1-15

Format	Mean	Max	R_s(%)
Flatbuffers	0.22	0.5	97.59
Protocol Buffers	2.80	8.04	89.54
Cap'n (unpacked)	0.80	1.84	89.54
Avro	2.62	5.56	61.66
CBOR	3.71	7.59	45.58
Smile	5.05	10.7	32.44
Thrift	5.11	10.75	17.69
Json	6.59	13.91	0.00
Cap'n (packed)	7.70	16.39	-1.34
MessagePack	7.53	14.01	-32.98
BSON	11.07	21.83	-76.94
XML	13.33	26.4	-92.76
AmazonIon	14.26	29.54	-116.09
YAML	67.65	139.53	-866.49

While not as performant as Flatbuffers, Protobuf demonstrates excellent efficiency, particularly at smaller message sizes, but has a significant increase with larger messages, suggesting that Protobuf is optimized for small to medium-sized messages. Thrift shows higher µs/op for smaller messages than Avro and Protobuf. However, a study presented in [9] suggests that the Thrift protocol outperformed others due to rapid serialization and deserialization of the packets for communication.

Though, in this study, we are not evaluating protocol efficiency, we argue that the sheer serialization speed of the format might not be a main contributor to Thrift protocol performance.

Cap'n Proto (unpacked) performs comparably to Protobuf and Flatbuffers for smaller messages but shows a larger increase in µs/op with larger messages. This suggests that while Cap'n Proto is designed for speed and efficiency, it may not be as well-suited for larger messages as Flatbuffers. Our testing scenario results contradict the results in [20], where it was suggested that Cap'n Proto has a slight speed advantage over FlatBuffers.

BSON shows inferior results compared to JSON, with serialization times reaching 21 µs/op for smaller messages and 52.17 µs/op for larger ones.

We may notice that Avro performs relatively well compared to Protobuf and Thrift across a range of large message sizes, as it has been seen in Table 2. Avro's serialization time increase is more gradual than Protobuf's, which sees a more dramatic growth as message sizes grow. Although Protobuf's times are higher than Avro's for larger messages, it is still within a competitive range for smaller messages.

Table 2

Serialization time distribution (µs/op), epoch 16-30

Format	Median	Mean	R_s(%)
Flatbuffers	1.52	1.61	95.42
Cap'n (unpacked)	5.51	6.35	83.40
Avro	14.36	14.93	56.75
CBOR	17.9	18.74	46.08
Smile	24.07	25.81	27.50
Thrift	25.73	25.03	22.50
Protocol Buffers	26.86	27.23	19.10
MessagePack	30.02	32.44	9.58
Json	33.2	36.78	0.00
Cap'n (packed)	41.9	43.48	-26.20
BSON	52.17	55.76	-57.14
XML	64.89	68.09	-95.45
AmazonIon	74.54	84.92	-124.52
YAML	351.93	379.37	-960.03

Additionally, in this scenario, Cap'n (unpacked) performs inferior to Flatbuffers, which contradicts the results presented in [20].

Notably, in tests on larger messages from epochs 16-30, schema-less CBOR outperformed the schema-based binary format Thrift, with a median speed of 17.9 compared to Thrift's 25.73.

Cap'n Proto (packed) presents moderate serialization times for smaller messages but shows the largest relative increase in time as message sizes increase. Let's compare this with the result in Table 1. The us efficiency trend of CBOR and Smile is consistent even as the message size scales up. In contrast, the relative inefficiency of BSON and Cap'n Proto (packed) becomes more pronounced with larger messages.

MessagePack remains a middle-ground option but is not superior to Protobuf as opposed to the results mentioned in [21], though it was much closer with larger messages. Overall, as it has been expected, textual formats performed much worse than binary formats, especially when compared to Zero-copy serialization formats like FlatBuffers and Cap'n Proto.

VI. DESERIALIZATION SPEED BENCHMARK

Let us examine the results of the deserialization speed benchmark with smaller test messages first presented in Table 3.

Flatbuffers exhibit exceptionally low deserialization times, consistently outperforming other formats across all epochs. This confirms findings from [3], where it was suggested that Flatbuffers wins in deserialization time compared to Protobuf

Cap'n Proto (unpacked) follows closely with a median time of 0.06 µs/op and an R_d=99.1 %. Like Flatbuffers, it provides extremely efficient data access without a deserialization step, which contributes to its high performance.

Table 3

Deserialization time distribution (µs/op), epoch 1–15

Format	Median	Mean	R_d(%)
Flatbuffers	0.02	0.02	99.7
Cap'n (unpacked)	0.06	0.06	99.1
Protobuf	0.88	3.61	87.1
Thrift	1.15	2.97	83.1
Avro	2.39	5.60	64.9
Cap'n (packed)	3.92	6.80	42.4
Smile	4.44	7.42	34.7
CBOR	4.69	7.81	31.0
MessagePack	6.49	11.18	4.6
JSON	6.8	13.66	0.0
BSON	10.26	20.15	-50.9
XML	19.36	38.32	-184.7
AmazonIon	25.22	41.65	-270.9
YAML	65.21	118.46	-859.0

The results for medium messages in Table 4. follow the same pattern as for smaller messages, showing that Flatbuffers and Cap'n Proto (unpacked) consistently excel, with minimal increases in deserialization times.

Table 4

Deserialization time distribution (µs/op), epoch 16-30

Format	Median	Mean	R_d(%)
Flatbuffers	0.03	0.03	99.96
Cap'n (unpacked)	0.1	0.10	99.86
Thrift	18.09	18.72	74.04
Protobuf	25.17	26.06	63.88
Avro	33.12	35.61	52.48
Cap'n (packed)	35.04	37.01	49.72
Smile	35.83	37.62	48.59
CBOR	39.05	39.07	43.97
MessagePack	48.24	52.89	30.78
JSON	69.69	76.20	0.00
BSON	102.06	109.94	-46.45
XML	214.64	1545.24	-207.99
AmazonIon	245.89	264.31	-252.83
YAML	583.51	578.50	-737.29

VII. DISTRIBUTED BENCHMARK FRAMEWORK

For this benchmark, we will employ three Java Virtual Machine (JVM) threads that act as distinct components of a distributed system; they will be referred to as Thread A, B, and C.

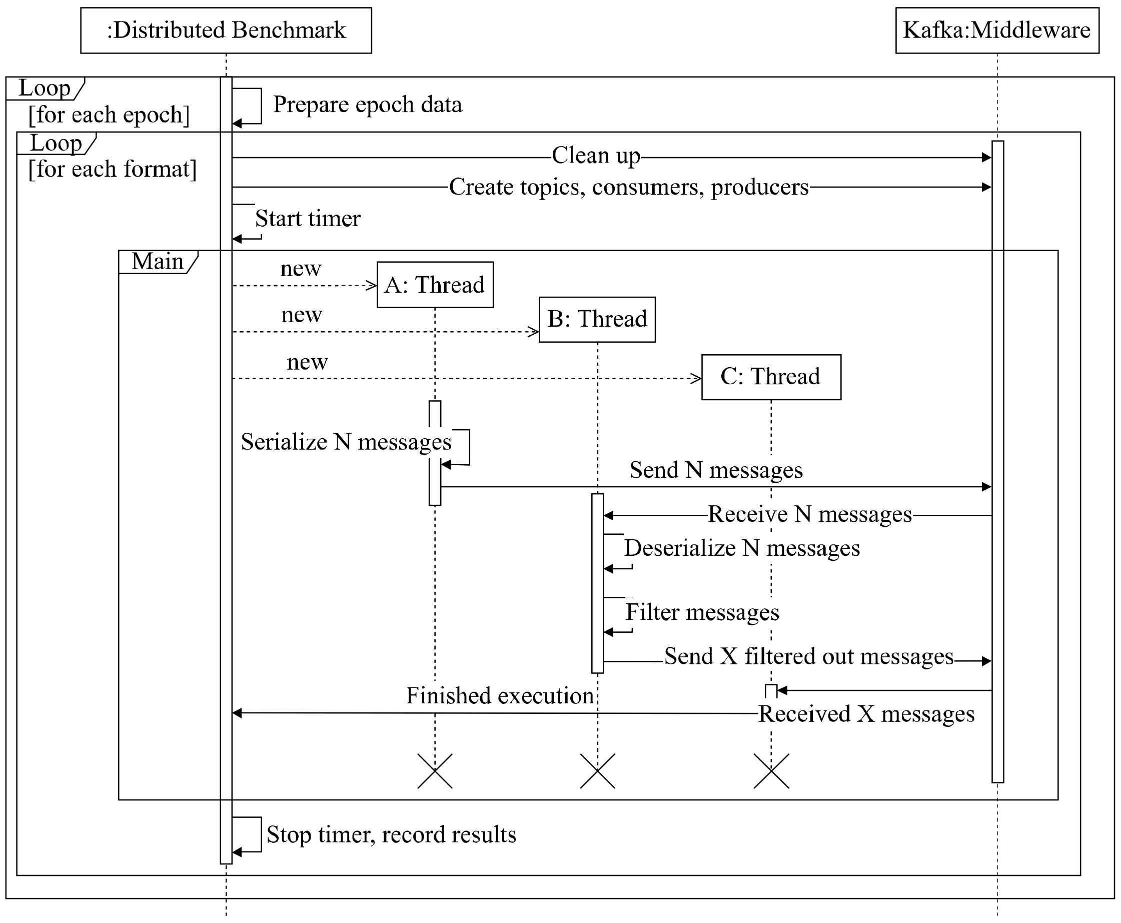


Fig. 1. A designed test scenario for distributed benchmark

These threads are designed to imitate the roles of producers and consumers within a distributed architecture, utilizing Kafka as the middleware for message passing. A designed test scenario is depicted as a UML sequence in Fig. 1.

VIII. DISTRIBUTED BENCHMARK

Now, let us analyze the resulting lag metrics for several key formats, presented in Table 5. Avro, Protobuf, and Thrift showed the best performance in this distributed benchmark, significantly reducing median delay in distributed communication scenarios compared to JSON, a traditional serialization format. Avro decreased median delay by R_l=84.3 %, Protobuf by 82.4 %, and Thrift by 80.4 %. Cap'n (packed) was able to reduce median latency by 70.8 %. Flatbuffers decreased median latency by 17.1 %, and MessagePack, CBOR, and Smile decreased median latency by 10-13 %. The rest of the formats did not decrease latency significantly (<10 %) or increased compared to JSON. For example, YAML increased median latency by 14.5 %. In future research, one might consider improving simulation by including additional test scenarios, message complexity, message nesting variations, and serialization formats like Microsoft Bond, PSON, and UBJSON.

Table 5

Distributed simulation latency distribution

Format	Mean (ms)	Median (ms)	R_l(%)	Range (ms)
Avro	1566.5	1526.0	84.3	2326
Protobuf	1750.5	1664.4	82.4	2278
Thrift	1948.5	2035.6	80.4	3085
Cap'n (packed)	2904.5	3075.6	70.8	11749
Flatbuffers	8255.5	8411.8	17.1	11512
Smile	8744.5	8634.6	12.2	12588
MessagePack	8814	8803.6	11.5	12700
CBOR	8900.5	8570.7	10.6	11811
Cap'n(unpacked)	9103.5	8667.1	8.6	5090
AmazonIon	9122.5	9141.4	8.4	12300
BSON	9338.5	8986.2	6.2	12318
XML	9864	9545.2	1.0	13048
JSON	9959	8929.6	0.0	15476
YAML	11399	11447	-14.5	12898

VIII. CONCLUSION

Our findings confirm that transitioning to an alternative data serialization format can yield a substantial latency reduction, surpassing R_l=50 % when compared to traditional JSON. This result is particularly relevant for systems where real-time performance is critical, as each millisecond gained or lost in latency can significantly impact overall system efficiency. Our distributed benchmark analysis further highlights that message compactness plays a more decisive role in reducing distributed latency than the raw speed of serialization and deserialization processes. This insight underscores the importance of format selection based on message structure and overhead, rather than focusing solely on serialization/deserialization speeds.

Additionally, our research indicates that schema-based binary formats such as Avro, Protocol Buffers, and Thrift outperform schema-less formats when the goal is to minimize latency in distributed systems.

References

[1] Marii, B., Zholubak, I., (2022). Features of Development and Analysis of REST Systems. *Advances in Cyber-Physical Systems*, vol. 7, no. 2, 121–129. DOI: 10.23939/acps2022.02.121.

[2] Weerasinghe, S., Perera, I., (2024). Optimized Strategy in Cloud-Native Environment for Inter-Service Communication in Microservices. *International Journal of Online and Biomedical Engineering*, vol. 20, no. 01, 40–57.
DOI: 10.3991/ijoe.v20i01.44021.

[3] Proos, D. P., Carlsson, N., (2020). Performance Comparison of Messaging Protocols and Serialization Formats for Digital Twins in IoV. In *2020 IFIP Networking Conference (Networking)*, Paris, France, 10–18. [Electronic resource]. – Available at: https://ieeexplore.ieee.org/document/9142787 (Accessed: 03/22/2024).

[4] Buono, V., Petrovic, P., (2021). Enhance Inter-service Communication in Supersonic K-Native REST-based Java Microservice Architectures. URN: https://urn.kb.se/resolve?urn=urn:nbn:se:hkr:diva-22135

[5] Morschel, L., (2020). dCache – Efficient Message Encoding For Inter-Service Communication in dCache: Evaluation of Existing Serialization Protocols as a Replacement for Java Object Serialization. *EPJ Web Conf.*, vol. 245, 05017. DOI: 10.1051/epjconf/202024505017.

[6] Friesel, D., Spinczyk, O., (2021). Data Serialization Formats for the Internet of Things. In *Electronic Communications of the EASST*, vol. 20, 1–4. DOI: https://doi.org/10.14279/tuj.eceasst.80.1134.

[7] Luis, Á., Casares, P., Cuadrado-Gallego, J. J., Patricio, M. A., (2021). PSON: A Serialization Format for IoT Sensor Networks. In *Sensors*, vol. 21, no. 13, 4559. DOI: 10.3390/s21134559.

[8] Viotti, J. C., Kinderkhedia, M., (2022). A Survey of JSON-compatible Binary Serialization Specifications. DOI: 10.48550/arXiv.2201.02089.

[9] Kumar, P. K., Agarwal, R., Shivaprasad, R., Sitaram, D., Kalambur, S., (2021). Performance Characterization of Communication Protocols in Microservice Applications. In *International Conference on Smart Applications, Communications and Networking (SmartNets)*, 1–5. DOI: 10.1109/SmartNets50376.2021.9555425.

[10] Viotti, J. C., Kinderkhedia, M., (2022). Benchmarking JSON BinPack. DOI: 10.48550/ARXIV.2211.12799.

[11] Huang B., Tang Y., (2021). Research on optimization of real-time efficient storage algorithm in data information serialization. *PLoS ONE*, vol. 16, no. 12, e0260697. DOI: 10.1371/journal.pone.0260697.

[12] Ahmad, T., Ars, Z. A., Hofstee, H. P., (2022). Benchmarking Apache Arrow Flight - A wire-speed protocol for data transfer, querying and microservices. DOI: 10.48550/arXiv.2204.03032.

[13] Dauda, A. B., Adam, M. S., Mustapha, M. A., Mabu, A. M., and Mustafa S., (2020). "Soap serialization effect on communication nodes and protocols,"
DOI: 10.48550/ARXIV.2012.12578.

Evans D., (2020). Energy-Efficient Transaction Serialization for IoT Devices. *Journal of Computer Science Research*, vol. 2, no. 2, 1–16. DOI: 10.30564/jcsr.v2i2.1620.

[14] Viotti, J. C., Kinderkhedia, M., (2022). “A Benchmark of JSON-compatible Binary Serialization Specifications,” DOI: 10.48550/ARXIV.2201.03051.

[15] Protocol Buffers Version 3 Language Specification. [Electronic resource]. – Available at: https://protobuf.dev/reference/protobuf/proto3-spec/ (Accessed: 03/22/2024).

[16] Currier, C., (2022). Protocol Buffers. In *Mobile Foren-Ïsics – The File Format Handbook: Common File Formats and File Systems Used in Mobile Devices*, Springer International Publishing, 223–260. DOI: 10.1007/978-3-030-98467-0 9.

[17] Wang, X., Xie, Z., (2020). The Case For Alternative Web Archival Formats To Expedite The Data-To-Insight Cycle. In *Proceedings of the ACM/IEEE Joint Conference on Digital Libraries in 2020.* In JCDL ’20. New York, NY, USA: Association for Computing Machinery, 177–186. DOI: 10.1145/3383583.3398542.

[18] Li, T., Shi, H., Lu, X., (2021). HatRPC: hint-accelerated thrift RPC over RDMA. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, in SC ’21. New York, NY, USA: Association for Computing Machinery, pp. 1–14. DOI: 10.1145/3458817.3476191.

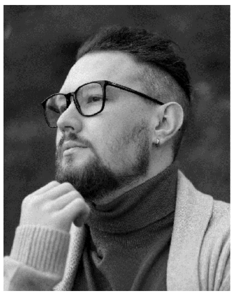
[19] Sorokin, K., (2023). “Benchmark comparing various data serialization libraries," [Electronic resource]. – Available at: https://github.com/thekvs/cpp-serializers. (Accessed: 03/22/2024).

[20] Hamerski, J. C., Domingues, R. P., Moraes F. G., Amory A., (2018). “Evaluating Serialization for a Publish-Subscribe Based Middleware for MPSoCs," in *25th IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, Bordeaux, France, pp. 773–776, DOI: 10.1109/ICECS.2018.8618003.

[21] Peltenburg, J., Hadnagy, Á., Brobbel, M., Morrow, R., Al-Ars Z., (2021). Tens of gigabytes per second JSON-to-Arrow conversion with FPGA accelerators. In *2021 ICFPT*, 1–9. DOI: 10.1109/ICFPT52863.2021.9609833.

[22] Maltsev, E., Muliarevych, O., (2024). Beyond JSON: Evaluating Serialization Formats for Space-Efficient Communication. *Advances in Cyber-Physical Systems*, vol. 9, no. 1, 9-15. DOI: 10.23939/acps2024.01.009.

[23] Kniazhyk, T., Muliarevych O., (2023). Cloud Computing With Resource Allocation Based on Ant Colony Optimization. *Advances in Cyber-Physical Systems*, vol. 8, no. 2, 104–110. DOI: 10.23939/acps2023.02.104.



Maltsev Eduard obtained his Master’s in Computer Engineering, specializing in Computer Systems and Networks, from Lviv Polytechnic National University in 2013. In 2021, he became a Certified Cloud Architect and is currently working towards a Ph.D. in Computer Engineering.

Dr. Riaz Ul Amin completed his PhD in Computing Science at the University of Glasgow. He is currently a Postdoctoral Research Fellow at Edinburgh Napier University, UK, specializing in Applied AI in Distributed Computing.