Smart Grid Communication Comparison

Distributed Control Middleware and Serialization Comparison for the Internet of Things

Petersen, Bo Søborg; Bindner, Henrik W.; Poulsen, Bjarne; You, Shi

Published in:
Proceedings of 7th IEEE International Conference on Innovative Smart Grid Technologies

Link to article, DOI:
10.1109/ISGTEurope.2017.8260268

Publication date:
2017

Document Version
Peer reviewed version

Smart Grid Communication Comparison

Distributed Control Middleware and Serialization Comparison for the Internet of Things

Bo Petersen, Henrik Bindner, Bjarne Poulsen, Shi You
DTU Electrical Engineering, DTU Compute
Technical University of Denmark
2800 Lyngby, Denmark
{bspet, hwbi, sy}@elektro.dtu.dk, bjpo@dtu.dk

***Abstract*—To solve the problems caused by intermittent renewable energy production, communication between Distributed Energy Resources (DERs) and system operators is necessary. The communication middleware and serialization used for communication are essential to ensure delivery of the messages within the required timeframe, to provide the necessary ancillary services to the power grid. This paper shows that there are better alternatives to using Web Services and XMPP as middleware and that there are better alternatives than using XML for serialization. The paper also gives guidance at choosing the best communication middleware and serialization format/library, aided by the authors' earlier work, which investigates the performance and characteristics of communication middleware and serialization independently. Given the performance criteria of the paper, ZeroMQ, YAMI4, and ICE are the middleware that performs the best, and ProtoBuf (ProtoStuff), and ProtoStuff are the serialization that performs the best.**

Keywords—Smart Grid; Internet of Things; Communication Middleware; RMI; XML-RPC; CORBA; ICE; Web Services; OPC UA; XMPP; WAMP; YAMI4; ZeroMQ; Serialization; XML; JSON; YAML; FST; Kryo; JAXB; Jackson; XStream; ProtoStuff; Gson; Genson; SnakeYaml; MsgPack; Smile; ProtoBuf; BSON; Hessian; CBOR; Avro

I. INTRODUCTION

With the future Smart Grid, being production following instead of a traditional power grid, which is load following, communication to Distributed Energy Resources (DERs) is necessary to efficiently use the power in the power grid when it is available, and keep the power grid stable.

The reason the power grid needs to be production following is because of the intermittent production of the increasing share of renewable energy from sources like solar and wind power [1].

The ancillary power system services, that need to be provided by the DERs to efficiently use the power when it is available, and keep the grid stable, such as primary frequency control, require that the probability of delivery of measurements and control commands within a given time frame is as high as possible, which is determined by the data connection, the processing unit in the DERs, the communication middleware, and the serialization.

Sponsored by the project, **PROA**ctive **IN**tegration of sustainable energy resources enabling active distribution networks (**PROAIN**).

The data connection and the processing units in the DERs depend on the owners and the manufacturer of the DERs respectively and is outside the scope of the paper.

The paper focuses on the performance of the communication middleware combined with the serialization, which strongly affects the probability of delivery of the measurements and control commands within the given timeframe.

The current state of the art by previous papers [2] [3] [4] including the earlier papers "Smart Grid Communication Middleware Comparison" [5] and "Smart Grid Serialization Comparison" [6] investigate the performance of the communication middleware and serialization individually.

This paper combines the communication middleware from the earlier paper "Smart Grid Communication Middleware Comparison" [5] with the serialization formats/frameworks from the earlier paper "Smart Grid Serialization Comparison" [6], with the arguments for including these communication middleware and serialization formats/frameworks found in the earlier papers.

The aim of this paper is to determine the performance of combining the communication middleware and the serialization previously investigated to show the combined performance of the combinations, and to show if the combined performance is as expected.

This paper does not cover the characteristics of the middleware and the serialization formats/libraries, as they are covered in the earlier papers.

The hypothesis of the paper is that with careful consideration for choosing the right communication middleware and serialization format/library, the performance and therefore probability of delivery of measurements and control commands within the given timeframe can be vastly improved. Especially compared to the recommendations by prevalent communication standards for Smart Grids, including the IEC 61850 [7], OpenADR [8] and CIM [9] standards.

II. METHODS

The tests were performed in Java using Oracle JDK 1.8.0_111, on a pair of Raspberry Pi 3's (Model B) with a 1

Gbit/s data connection, but the Raspberry Pi's only have 100 Mbit/s network interfaces.

The middleware and serialization framework/libraries included in this comparison are taken from the earlier papers "Smart Grid Communication Middleware Comparison" [5] and "Smart Grid Serialization Comparison" [6], which consists of 10 middleware and 25 serializers.

The measurements consist of the average number of messages sent from one device to another (throughput), and the average time it takes for a message to get from one device to another (latency), summarized as:

- Throughput (messages per second)
- Latency (milliseconds per message)

The messaging patterns measured consist of Request-Reply, which is used for legacy systems to retrieve measurements, without knowing when it is available. Push-Pull that is used to send control commands, and Publish-Subscribe used in newer systems, where a device subscribes to measurements and gets them when they are available, summarized as:

- Request-Reply
- Push-Pull
- Publish-Subscribe

The results come from executing each test, for each pattern, and each combination of middleware and serialization, 10 times and taking the average of the measurements.

Each test takes an IEC 61850 [7] class (logical node) with randomly generated data, de-/serializes it using the current serializer and transfers it using the current communication middleware, using the current messaging pattern.

The tests are performed by giving the sending device 10 seconds to send and then counting the number of messages received for 10 seconds on the receiving device, from the time the first message arrives.

Throughput is measured by using a server on one device and a client on the other, with Request-Reply going from the client to the server and back, Push-Pull going from the client to the server, and Publish-Subscribe going from the server to client, with the final receiving device measuring the number of messages received during the 10 second interval.

With Push-Pull and Publish-Subscribe the sending and receiving devices are not the same, which is a problem for measuring time, with different clocks, therefore the data is sent back again to the original sending device, so the same clock is used for both time measurements, and the time is then divided by 2. For Request-Reply, the sending and receiving device is always the same device, but the data was still sent again, and divided by 2, to make sure the results are comparable to the other patterns.

For Request-Reply, only a tiny request message is sent with the full message payload being returned which differs from the approach from the earlier paper on communication middleware.

III. Results

The throughput tests using Request-Reply (Fig. 1) shows that ICE and ZeroMQ perform great, YAMI4, RMI, and CORBA perform very well, XML-RPC and WAMP perform adequately, and Web Services, XMPP, and especially OPCUA struggles to keep up.

They also show that the 6 best serializers that perform better than adequately include only 1 string serializer, which is JSON (ProtoStuff), which uses an incompatible JSON format. In addition, the 3 serializers that perform the best are all part of the ProtoStuff library. On the other hand, the 12 serializers that perform the worst include all XML serializers, and result in a reduction in throughput of at least 3 times compared to the 3 best, with the 7 best middleware.

When looking at the throughput tests using Push-Pull (Fig. 2), ZeroMQ, WAMP, ICE, and YAMI4 perform great, CORBA performs well, XMPP and XML-RPC perform adequately, and RMI, Web Services, and OPCUA struggle to keep up, especially Web Services and OPCUA.

For the serializers, the worst 16 serializers include all XML serializers and all string serializers except for JSON (ProtoStuff). The 2 best serializers are part of the ProtoStuff library and perform at least 7 times better than the worst 16 serializers, for the 4 best middleware.

The Publish-Subscribe throughput tests (Fig. 3) show that XMPP and again especially OPCUA perform terribly, while ZeroMQ really shows its performance advantage in these tests.

Even fewer serializers perform well for Publish-Subscribe, only 7 of the 25, with all string serializers performing badly. Only 2 serializers perform great, both of which are part of the ProtoStuff framework, and perform at least 3 times better than the worst 18 serializers for the 3 best middleware.

The latency tests for all messaging patterns (Fig. 4-6) show that using OPCUA, XMPP or Web Services as middleware, or XML (JAXB) or MsgPack for serialization ruins the latency. For these middleware, for all patterns, the fastest combination is at least 13 times slower than the fastest combination for all middleware. While for these serializers, for all patterns, the best combination is 70 times slower than the fastest combination for all serializers.

The Request-Reply tests (Fig. 4) show that the latency for 45 percent of the combinations is below 20 milliseconds, for 29 percent it is below 10 milliseconds, and for 5 combinations, it is below 2 milliseconds.

The Push-Pull tests (Fig. 5) show that the latency for half the combinations is below 20 milliseconds, for a third, it is below 10 milliseconds and for 12 combinations it is below 2 milliseconds.

The Publish-Subscribe tests (Fig. 6) shows that the latency for a third of the combinations is below 20 milliseconds, that 20 percent of the combinations is below 10 milliseconds and that for 2 combinations it is below 2 milliseconds.

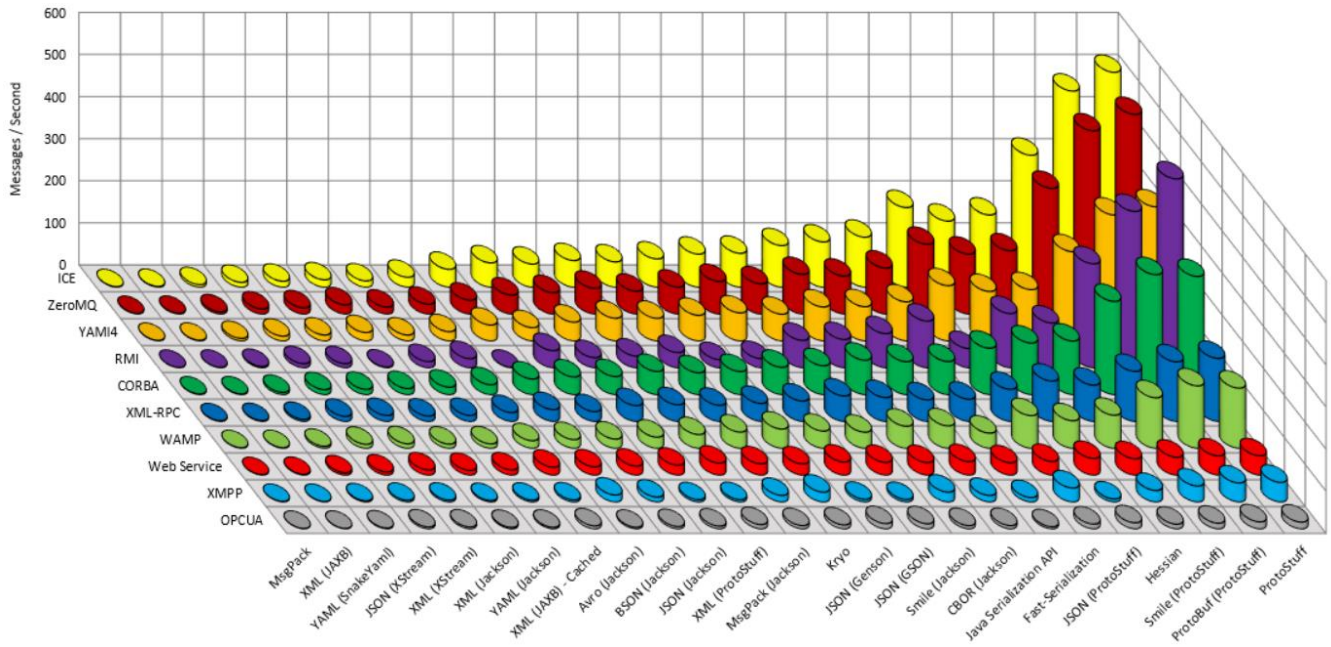

Fig. 1. Request-Reply throughput ordered by the sum of middleware and serialization throughput.

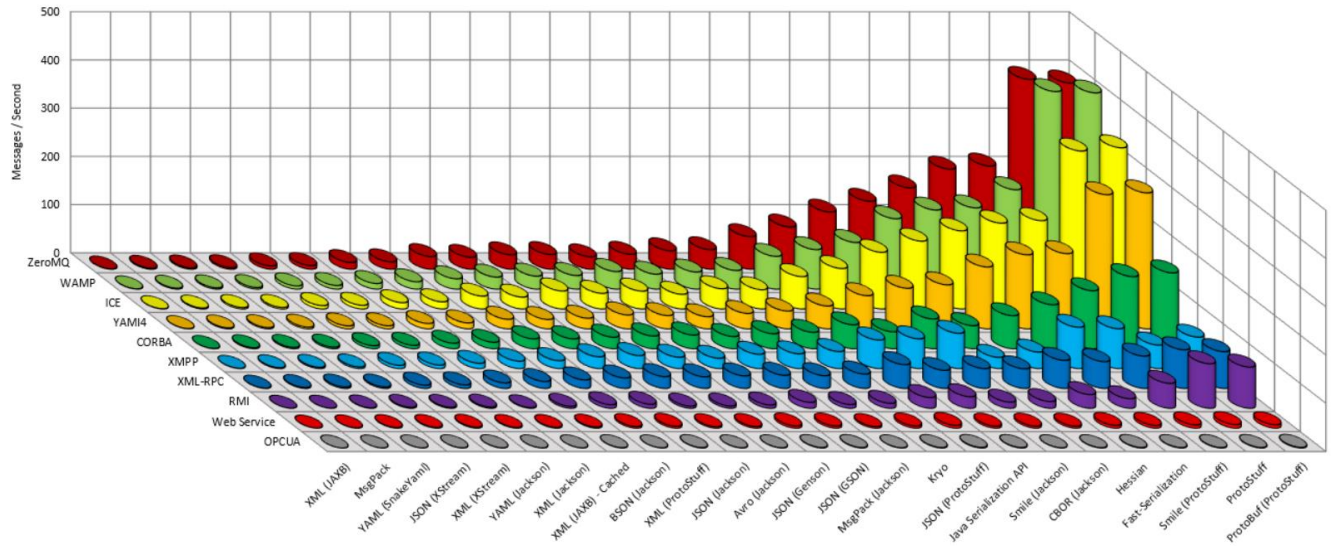

Fig. 2. Push-Pull throughput ordered by the sum of middleware and serialization throughput.

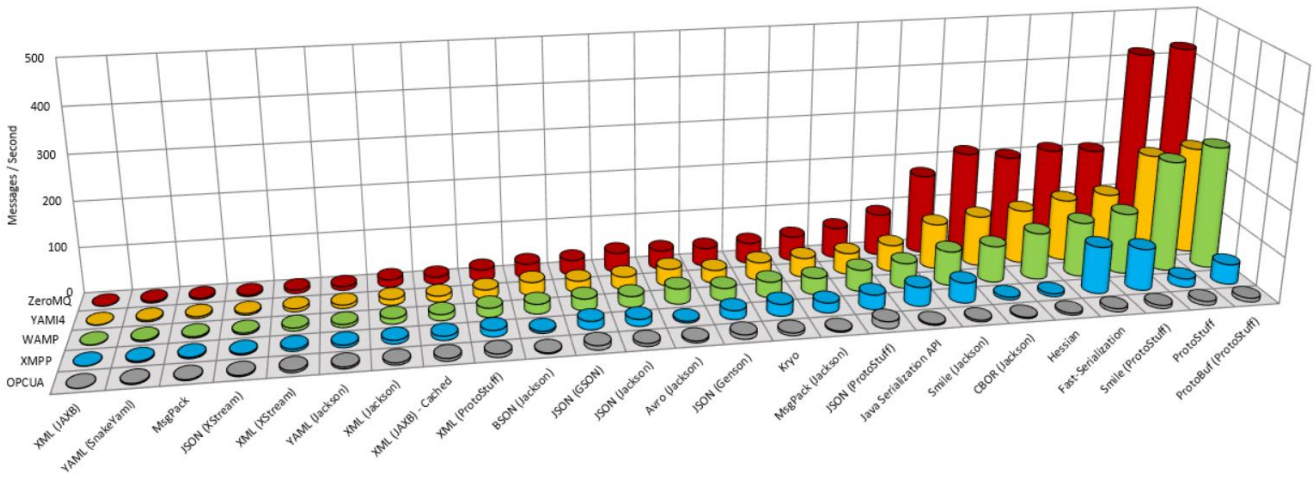

Fig. 3. Publish-Subscribe throughput ordered by the sum of middleware and serialization throughput.

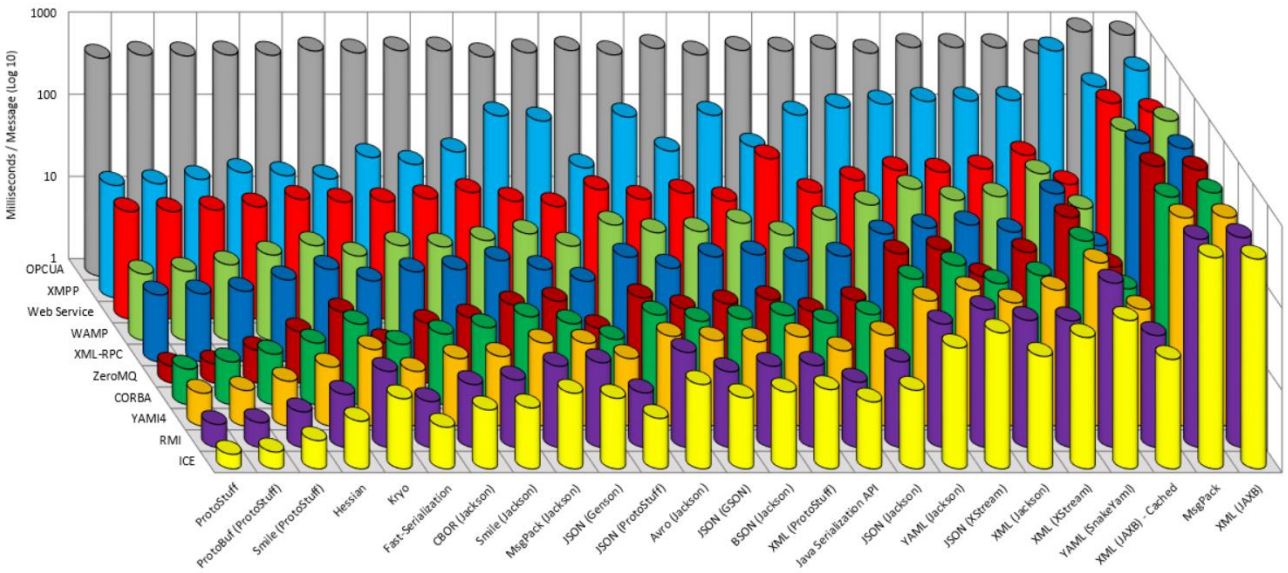

Fig. 4. Request-Reply latency ordered by the sum of middleware and serialization latency.

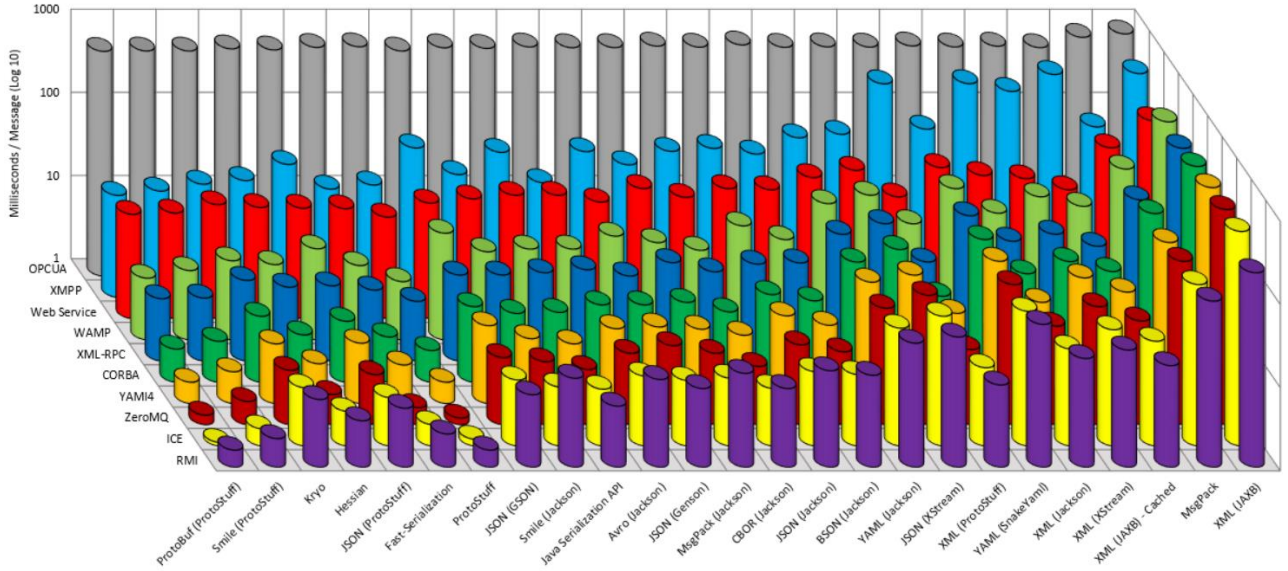

Fig. 5. Push-Pull latency ordered by the sum of middleware and serialization latency.

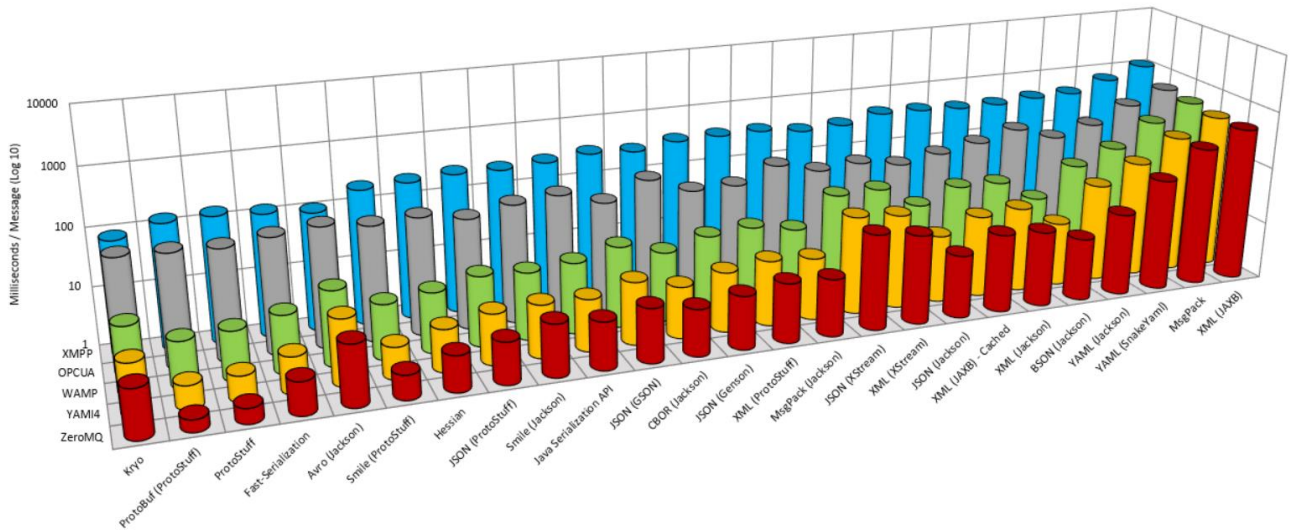

Fig. 6. Publish-Subscribe latency ordered by the sum of middleware and serialization latency.

IV. Discussion

A. Communication Standards

One thing that is clear from the performance measurements of throughput using all messaging patterns is that Web Services and XMPP are far from being the best middleware available.

The performance measurements of latency for all messaging patterns also show that using XMPP with all serializers or Web Services with the fastest serializers results in the latency being much higher than with all other middleware except for OPCUA.

This should be proof enough that there are much better alternatives for middleware than using Web Services or XMPP as recommended by the prevalent communication standards, at least in the case of the processing unit being a Raspberry Pi and a 100 Mbit/s data connection.

For serialization, the throughput and latency tests using all messaging patterns show that JSON is generally a better alternative to using XML, binary serializers are generally faster than string serializers are, and even though they show that the serialization format is more important than the library, the library still makes a big difference, especially for JSON.

This shows that there are better alternatives for serializers than the ones used by the prevalent communication standards, which are all based on XML.

B. Guidance

The choice of middleware and serialization makes a huge difference in both throughput and latency performance, with the best combinations having a throughput that is at least 8 times higher than the average, and a latency that is at least 75 times faster than the average, for all messaging patterns.

The best results for middleware come from using ZeroMQ, YAMI4 or ICE, with the first two, also enabling Publish-Subscribe. In addition, WAMP performs quite well for Push-Pull and Publish-Subscribe on throughput and does adequately on latency for the same messaging patterns, making it a good alternative for newer systems not using Request-Reply.

For serialization, the best results come from using ProtoBuf (ProtoStuff) or ProtoStuff, with the advantage of ProtoBuf (ProtoStuff) being compatible with Google ProtoBuf serializers. Smile (ProtoStuff) and Fast-Serialization, do however do a really good job and are good alternatives.

C. Real World Cases

It is quite interesting that a throughput of 500 messages per second for Request-Reply and 400 messages per second for Push-Pull and Publish-Subscribe can be achieved using IEC 61850 logical node classes.

Moreover, the fact that an average latency of under 2 milliseconds for all messaging patterns can be achieved with the right combination of middleware and serialization is quite promising.

D. Compared to Previous Results

Compared to the performance results from the paper "Smart Grid Communication Middleware Comparison" [5], which compares middleware using random binary and string data, of different sizes, the message sizes used for this paper range from 2 to 13 kB, and the time used for serialization is now part of the performance measurements. This takes away the big advantage with Request-Reply that ZeroMQ, ICE, and RMI had for small message sizes.

The results for middleware compared to earlier (Fig. 7) are as expected, with ZeroMQ, YAMI4 and ICE still leading in performance, but when looking at the difference between Request-Reply and the other messaging patterns, the difference is huge. The previous results showed that the other patterns are up to 10 times faster than Request-Reply, which because of the increased message sizes and the change from sending a full message in both directions for Request-Reply to only sending it back, makes the difference between the patterns quite small, with Request-Reply being the fastest.

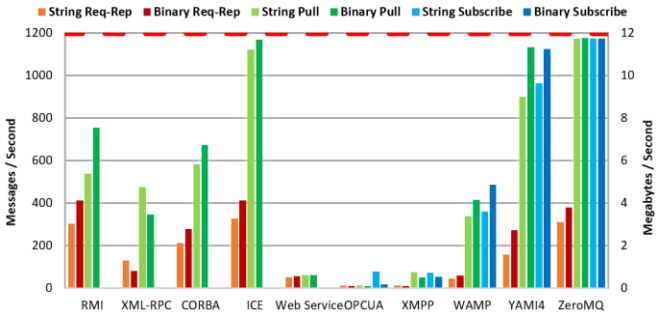

Fig. 7. Middleware throughput (10 kilobyte messages). [5]

For getting measurement data, Publish-Subscribe still has the advantage over Request-Reply, that it gets data when it is available instead of having to do polling to see if new data is available.

For serialization, the results compared to those from the earlier paper "Smart Grid Serialization Comparison" [3] (Fig. 8), show that the speed of the serializers is much more important than the size of the resulting serialized output. This is especially clear when looking at the bad performance of MsgPack and Avro (Jackson), which produce very small messages but are slower than most other binary serializers are.

Generally, the results compared to the previous results are as expected with ProtoBuf (ProtoStuff) and ProtoStuff being the fastest, but one thing that is quite surprising is that Kryo does significantly worse than Fast-Serialization, and is no longer one of the fastest serializers.

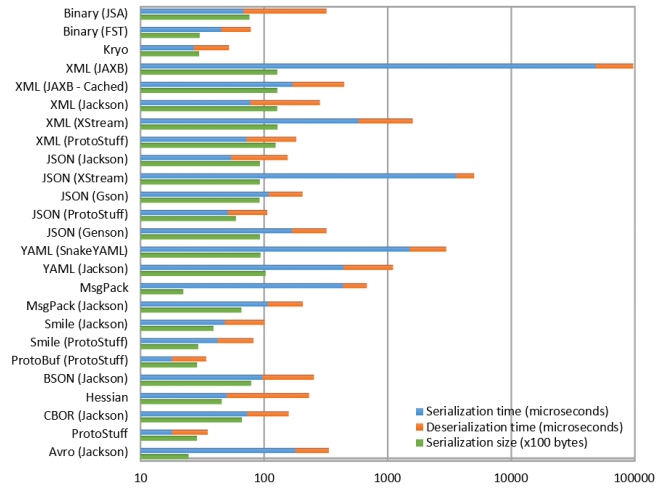

Fig. 8. Serialization time & size. [3]

V. Conclusion

The paper shows that there are much better middleware alternatives than XMPP & Web Services and that there are much better serialization alternatives than XML. All of which are advocated for by prevalent communication standards.

The choice of middleware makes a significant difference in performance, with ZeroMQ, YAMI4, and ICE being the best, and being much better than XMPP and Web Services. The eventual choice should be determined by the characteristics of the middleware, which can be found in the earlier paper “Smart Grid Communication Middleware Comparison” [5].

The choice of serialization is even more important than the choice of middleware, especially because of the loss of performance with most serializers compared to using ProtoBuf (ProtoStuff) and ProtoStuff, which are by far the best choices, with a Raspberry Pi and a 100 Mbit/s data connection.

The throughput measurements of up to 400 messages per second, and the latency measurements below 2 milliseconds, both for all messaging patterns, show how fast measurements and control commands could be delivered for distributed control systems. However, it should be considered that the distance between the actual devices will be longer, and therefore the throughput and latency will not be as good as for these tests, though a lot better than for a centralized control system with much longer distances.

Compared to previous results the same middleware perform the best, with the only real difference being that Request-Reply performs much better with the current tests than for the earlier tests.

For serialization, it was previously unknown whether the serializers with the smallest output size or the ones with the fastest serialization time would have the best performance when sending measurements and control commands using communication. For the current setup, the serialization time is much more important than the size of the message, but the chance of a future DER having a processing unit with the performance of a Raspberry Pi is a lot bigger than it having a 100 Mbit/s data connection to the other DERs.

Therefore, an important thing to investigate in the future is the impact of different processing units and different data connections, which could shift the balance for serialization between the importance of the size of the output and the serialization time.

With the current setup, the potential impact of compression would most likely be that the best results would come from using no compression, as the time used for compression would offset the gain from smaller messages, however, if the processing unit was stronger compared to the data connection, the results would probably be quite different.

When using the results, it should be considered that the objects that are serialized and sent are IEC 61850 logical node classed filled with random data for the required properties, which add a lot of overhead compared to more compact and primitive objects.

The memory used by the middleware and serialization is outside the scope of this paper, but it can be seen for middleware and serialization individually from the earlier papers, along with data loss.

References

[1] "Scientific American," [Online]. Available: http://blogs.scientificamerican.com/plugged-in/renewable-energy-intermittency-explained-challenges-solutions-and-opportunities/. [Accessed 27 09 2016].

[2] M. Albano, L. L. Ferreira, L. M. Pinho, A. R. Alkhawaja, "Message-oriented middleware for smart grids," in Computer Standards & Interfaces 38: 133-143., 2015.

[3] L. Qilin, L. Mintian, "The state of the art in middleware," in Information Technology and Applications (IFITA), 2010.

[4] A. Dworak, M. Sobczak, F. Ehm, W. Sliwinski, P. Charrue, "Middleware trends and market leaders 2011," in Conf. Proc.. Vol. 111010. No. CERN-ATS-2011-196. 2011., 2011.

[5] B. Petersen, H. Bindner, S. You, B. Poulsen, "Smart Grid Communication Middleware Comparison," in SmartGreens, Porto, 2017, in press.

[6] B. Petersen, H. Bindner, S. You and B. Petersen, "Smart Grid Serialization Comparison," in SAI Computing Conference, London, 2017, in press.

[7] R. E. Mackiewicz, "Overview of IEC 61850 and Benefits," IEEE PES Power Systems Conference and Exposition, Atlanta, GA, pp. 623-630, 2006.

[8] C. McParland, "OpenADR open source toolkit: Developing open source software for the Smart Grid," IEEE Power and Energy Society General Meeting, San Diego, CA, pp. 1-7, 2011.

[9] M. Uslar, S. Rohjans, S. Specht, J. M. G. Vázquez, "What is the CIM lacking?," IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT Europe), Gothenburg, pp. 1-8, 2010.