Smart Grid Communication Infrastructure Comparison–For Distributed Control of Distributed Energy Resources Using Internet of Things Devices

Bo Petersen, Henrik Bindner, Bjarne Poulsen, and Shi You
Technical University of Denmark, DTU Electrical Engineering & DTU Compute, Kgs. Lyngby, Denmark
Email: bo@petersen.dk, hwbi@elektro.dtu.dk, bjpo@dtu.dk, sy@elektro.dtu.dk

***Abstract*—Communication between distributed energy resources and aggregators is necessary to improve the efficiency of power use and solve stability issues. For the communication, the probability of delivery for measurements and control commands determines the possible power system services. The probability of delivery is determined by the processing units, data connection, middleware, and serialization. The comparison is made based on multiple experimental setups to test the performance of different middleware and serialization with different processing units and data connections in a Smart Grid context. The hardware includes Beagle Bone, Raspberry Pi, and Dell laptop processing units, and the data connection bandwidths are 1, 10, 100, and 1000 Mbit/s. The results show that there are better alternatives to the Extensible Messaging and Presence Protocol (XMPP) and Web Services middleware and XML serialization as advocated for by the prevalent communication standards. This paper gives guidance in choosing the best software and hardware for communication depending on the use case.**

***Index Terms*—smart grid, communication, infrastructure, middleware, serialization**

I. Introduction

Communication between distributed energy resources (DERs) and aggregators is required to improve the efficiency of power use in future smart grids and to support more reliable and robust operations.

The need for these improvements comes from the increase in intermittent renewable energy, primarily solar and wind power, which is problematic with a traditional load following power grid, thus requiring energy storage and a production-following smart grid.

The power grid will be made production following by DERs providing power system ancillary services, such as primary frequency control for stability, and by moving production and consumption, using services such as load shifting and shedding.

Some of these ancillary services require a high probability of delivery of measurements and control commands within a short timeframe, which is determined by the processing units of the DERs, the data connection between the DERs, the communication middleware, and the serialization used for the communication.

The state-of-the-art by previous papers [1]-[3], and the authors' earlier work [4]-[6], is that they investigated the performance and characteristics of middleware and serialization for communication but did not investigate the impact on performance by the processing units of the DERs and the data connection between the DERs.

This paper aims to determine the combined performance of middleware and serialization with different processing units and data connections, to determine the impact of the processing units and data connections, and to determine the best combination of processing units, data connection, middleware, and serialization.

The authors' earlier work covers the performance and characteristics of middleware and serialization separately, the combined performance, and also includes the arguments for including these communication middleware and serialization formats/libraries, which is therefore not covered in this paper.

The hypothesis of the paper is that the probability of delivery of measurements and control commands can be improved by the choice of middleware, serialization, processing unit, and data connection, especially compared to the middleware and serialization recommended by prevalent communication standards for smart grids, including IEC 61850 [7], OpenADR [8], and CIM [9].

II. Method

The tests were performed in Java using Oracle JDK 1.8.0_111 on ten setups which combine a pair of processing units and a data connection.

The processing units included consist of a pair of the following:

- Beagle Bone Black.
- Raspberry Pi 3 (model B).
- Dell Latitude E6520 laptop.

The data connections used by limiting the bandwidth

Manuscript received August, 2017; revised December 14, 2017.

doi: 10.18178/ijeetc.7.1.7-14

of the network from 1 Gbit/s using the Linux Traffic Control subsystem are as follows:

- 1 Mbit/s.
- 10 Mbit/s.
- 100 Mbit/s.
- 1 Gbit/s (only for Dell laptops).

The Beagle Bone and Raspberry Pi are only capable of 100 Mbit/s, and the Dell laptop is capable of 1 Gbit/s; therefore, the 1 Gbit/s data connection is only tested with the Dell laptops.

For each combination of processing units and data connections, the tests are performed with every combination of 10 middleware and 25 serializers included based on the earlier work of the authors'.

A message in this context means a set of measurements or a control schedule (as described in IEC 61850-7-{4, 420})

The tests measure the number of messages received (throughput) and the average time it takes to send a message (latency) within a 10 s period:

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

The tests are performed using three messaging patterns: Request–Reply for measurement polling, Push–Pull for sending control commands, and Publish–Subscribe for delivering measurements when they are made:

Each test is run 10 times on every combination of processing unit and data connections with every combination of middleware and serialization for every messaging pattern, measuring throughput, and latency.

To measure the time a message is sent and received to get the latency, the same clock is used, using the same processing unit to measure both, which require the measurements and control commands to be sent twice, with the result being half the time between being sent and received.

The data model used for measurements and control commands is the IEC 61850 [7] data model (one logical node per message, except the request message of Request–Reply, which is an IEC 61850 path string).

III. Results

A. Setup

With strong processing units like the Dell's and a fast data connection of 100+ Mbit/s, throughput speeds of 1000 messages per second can be reached (Fig. 1), with a latency below 1 ms per message (Fig. 2).

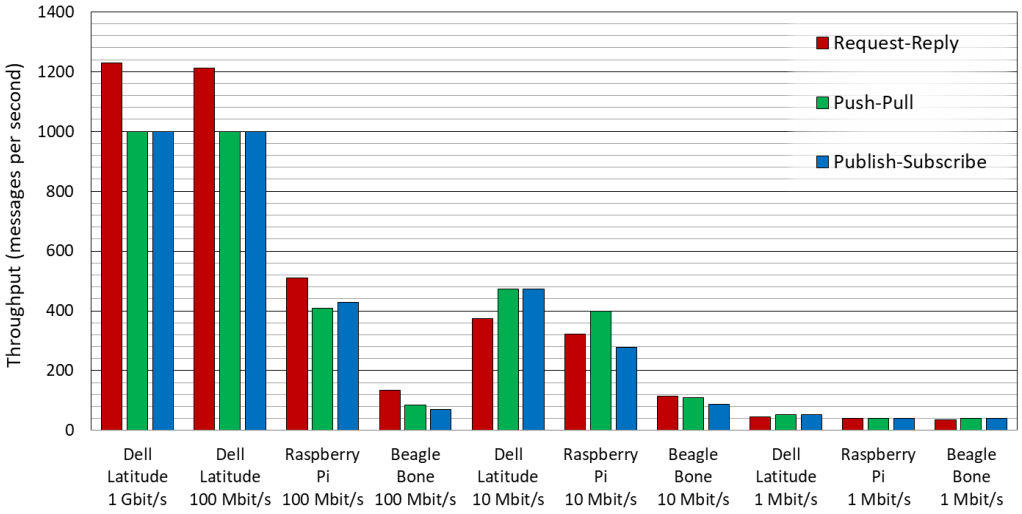

Figure 1. Maximum average throughput by setup and pattern for all combinations of middleware and serialization.

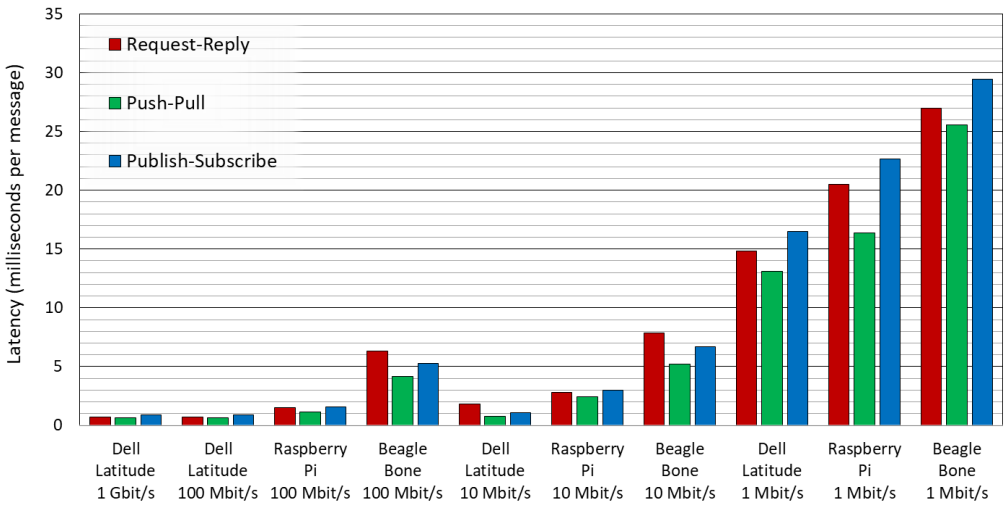

Figure 2. Minimum average latency by setup and pattern for all combinations of middleware and serialization.

To get throughput speeds of 280–510 messages per second and a latency below 3 ms per message, adequate processing units like the Raspberry Pi's or Dell's are needed, along with a data connection of at least 10 Mbit/s.

Using less powerful processing units like the Beagle Bone's will result in throughput fewer than 140 messages per second and a latency above 4 ms per message.

With a slow data connection of 1 Mbit/s, the throughput is below 50 messages per second, and the latency is above 13 ms per message.

The results show that a slow data connection of 1 Mbit/s limits the performance of all processing units, a data connection of 10 Mbit/s allows the Raspberry Pi's and Dell's to perform much better, and a fast data connection of 100 Mbit/s requires strong processing units like the Dell's to be utilized fully. The 1 Gbit/s data connection requires even stronger processing units than the ones tested.

The results also show that the Dell's need a 100 Mbit/s data connection to perform optimally, the Raspberry Pi's need a 10 Mbit/s data connection, and the Beagle Bone's only need a data connection a little faster than 1 Mbit/s.

For the messaging patterns, the results show that, when the data connection is fast, the Request–Reply pattern performs better, and Push–Pull and Publish–Subscribe do better when the processing units are stronger compared to the data connection.

B. Middleware

Fig. 3 shows the throughput, and Fig. 4 shows the latency of each middleware by each setup. They show that, for the two strongest setups with the strongest processing unit (Dell) and fast data connections (100+ Mbit/s), all middleware, except XMPP, Web Services, and OPC UA, perform well on throughput with 600+ messages per second, and the same middleware, except for XML-RPC and WAMP, also perform well on latency with less than 2.1 ms per message.

For each middleware, Fig. 5 shows the average utilization for all setups, with the maximum utilization defined as the utilization of the fastest middleware on each setup, for both throughput and latency.

When comparing the performance of communication middleware across the setups with regard to throughput, ICE, ZeroMQ, and WAMP perform the best with a utilization above 70%, and Web Services, XMPP, and OPC UA have a utilization below 31%.

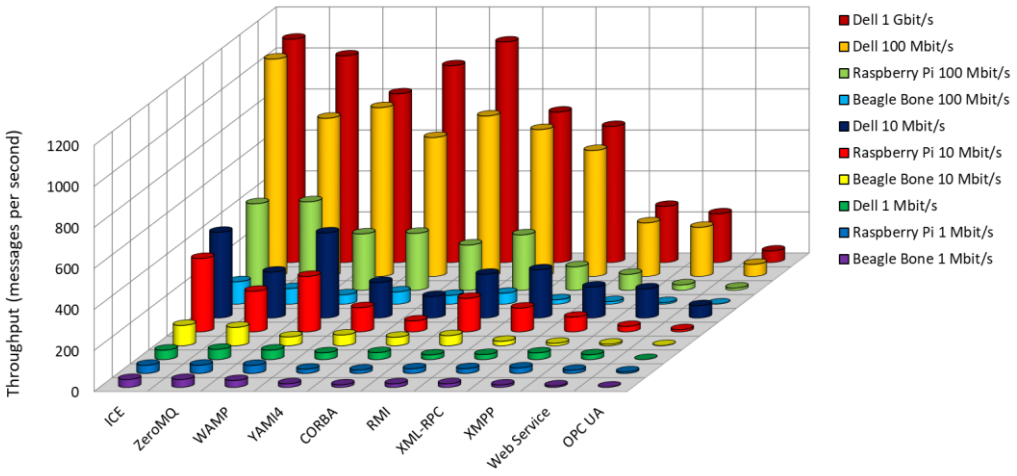

Figure 3. Maximum average throughput by setup and middleware for all serialization and patterns.

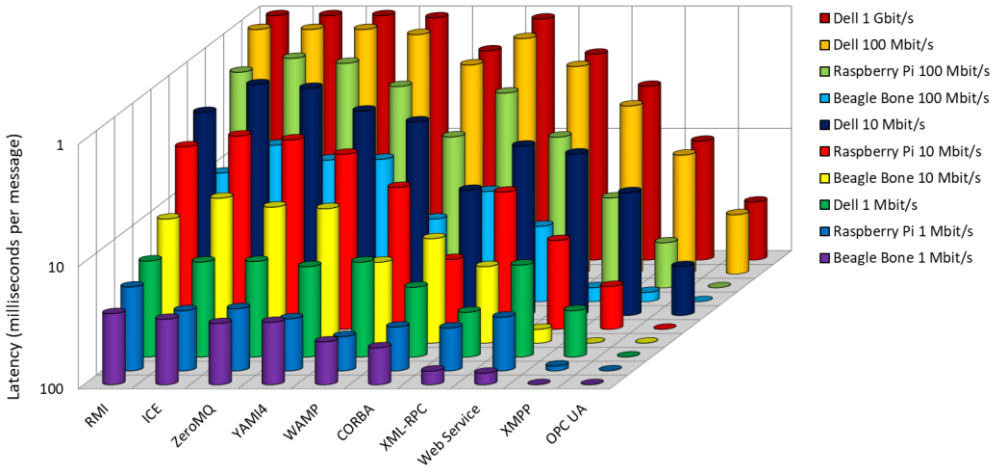

Figure 4. Minimum average latency by setup and middleware for all serialization and patterns.

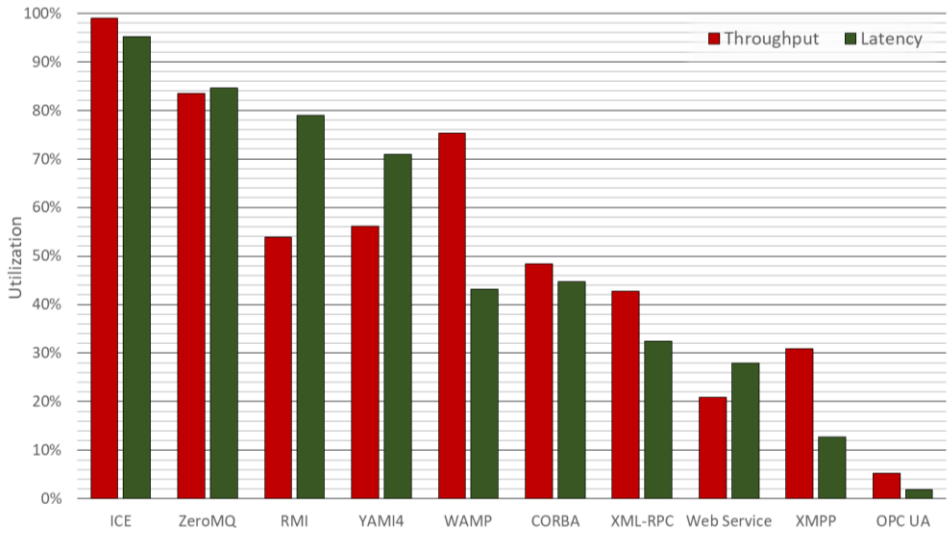

Figure 5. Middleware throughput and latency utilization.

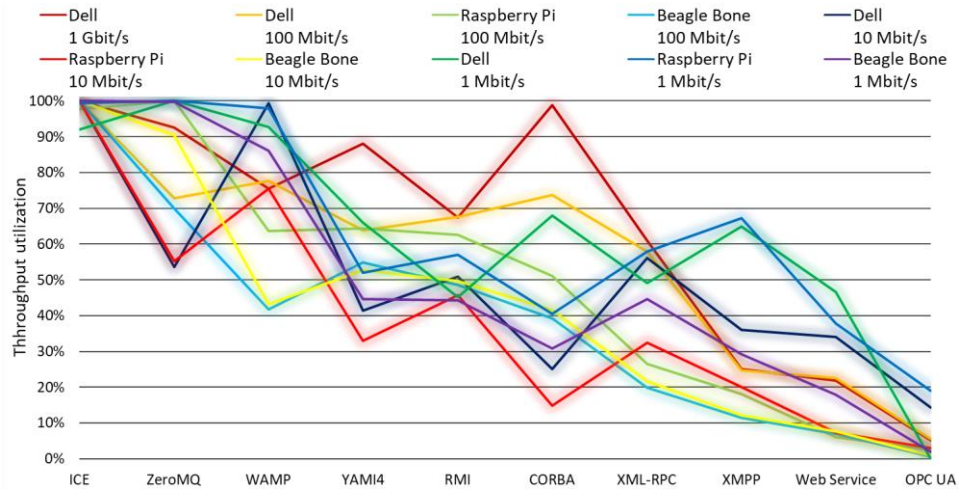

Figure 6. Middleware throughput utilization.

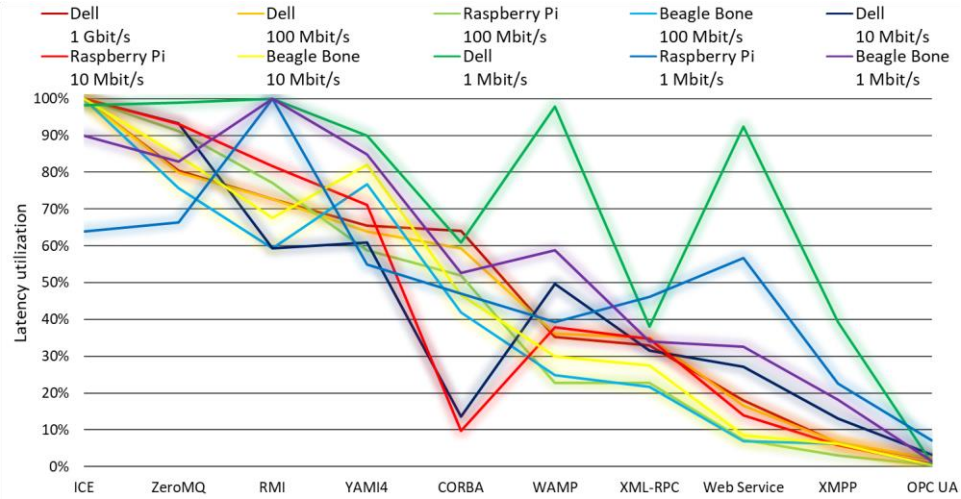

Figure 7. Middleware latency utilization.

On latency, ICE, ZeroMQ, RMI, and YAMI4 generally do an excellent job, with utilization above 70%, and Web Services, XMPP, and OPC UA have a utilization below 30%. To get the full picture of how the middleware perform, Fig. 6 and Fig. 7 show the throughput and latency utilization, respectively, for the middleware across the setups. Although ICE performs the best and OPC UA performs the worst in almost all cases for both throughput and latency, the performance of the other middleware is more fluctuating.

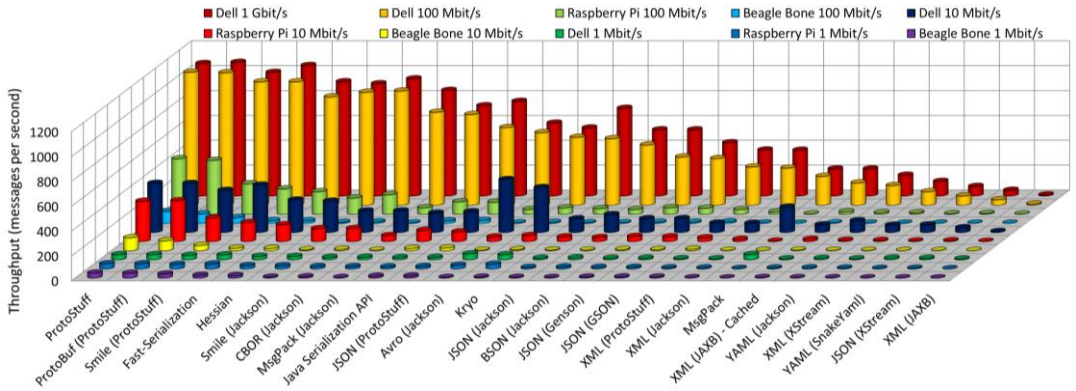

Figure 8. Maximum average throughput by setup and serialization for all middleware and patterns.

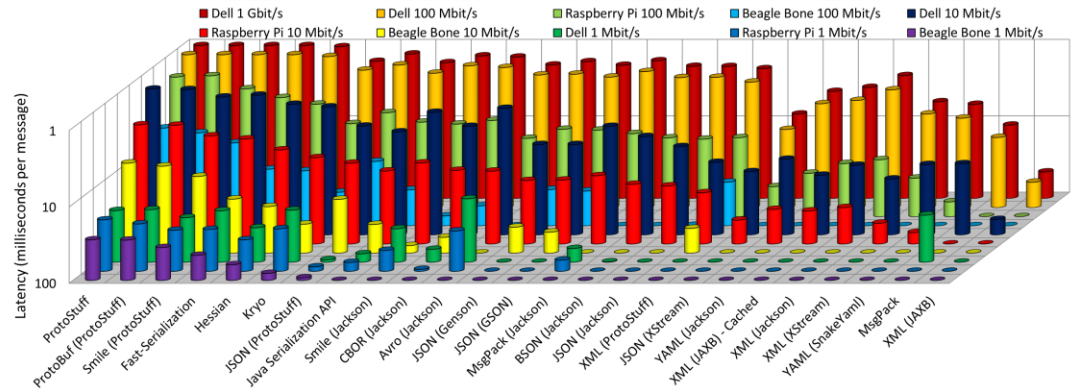

Figure 9. Minimum average latency by setup and serialization for all middleware and patterns.

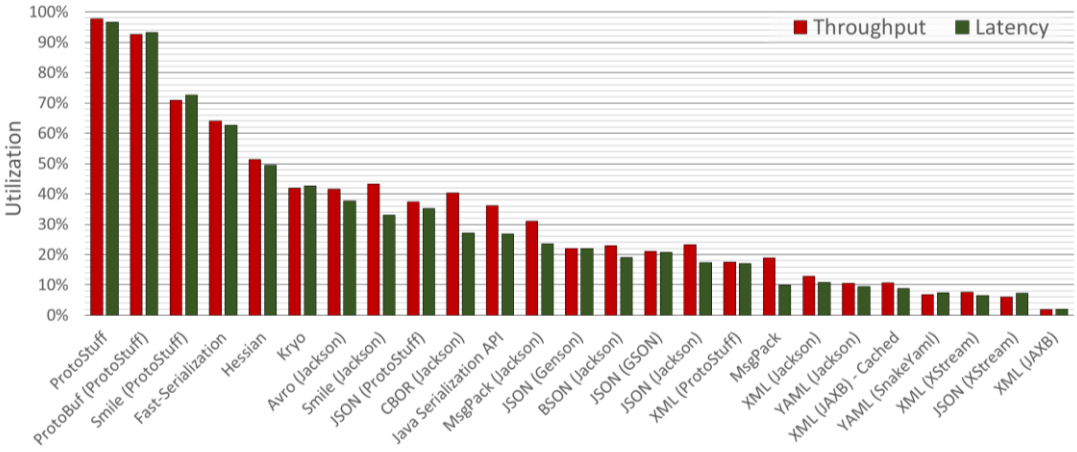

Figure 10. Serialization throughput and latency utilization.

The throughputs of ZeroMQ and WAMP differ a lot depending on the setup, but are generally good, and YAMI4 and RMI perform modestly on all setups.

Web Services and XMPP generally do not perform well compared to the other middleware, but when the processing units are very strong compared to the data connection, they do better.

Although the latency for ZeroMQ, RMI, and YAMI4 is relatively stable, for WAMP it is very similar to that for Web Services and XMPP, which is not as good.

C. Serialization

The throughput and latency for each type of serialization are shown in Fig. 8 and Fig. 9, respectively, for each setup.

ProtoStuff, ProtoBuf (ProtoStuff), Smile (ProtoStuff), Fast-Serialization, Hessian, Smile (Jackson), and CBOR (Jackson) reach throughputs of 870+ messages per second, and except for Smile (Jackson) and CBOR (Jackson), they achieve a latency fewer than 1.1 ms per message on the two fastest setups (Dell 100+ Mbit/s).

Fig. 10 shows the average throughput and latency utilization of all serialization for all setups, which shows that only ProtoStuff and ProtoBuf (ProtoStuff) have throughput and latency utilization above 90%, all XML serializers are below 20%, all JSON serializers, except 1, are faster than the XML serializers, and binary serializers are faster than string serializers.

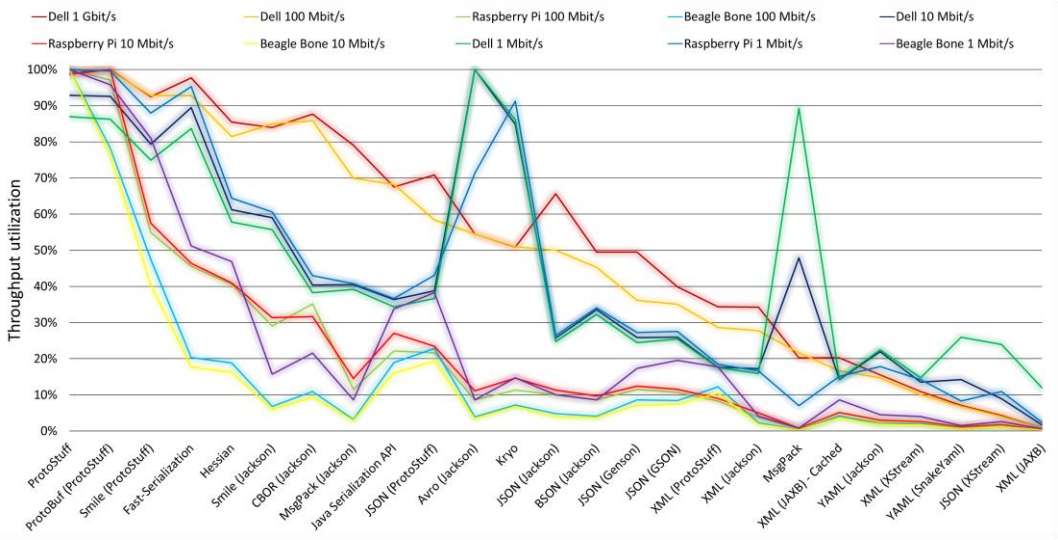

Figure 11. Serialization throughput utilization.

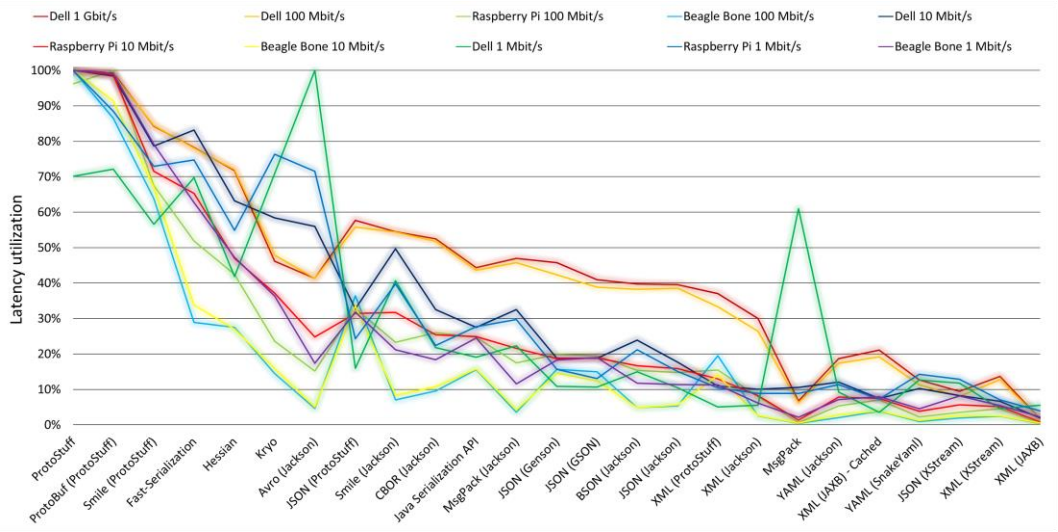

Figure 12. Serialization latency utilization.

Fig. 11 and Fig. 12 show the relative performance of the serializers for the setups on throughput and latency, respectively.

It is quite interesting that Avro (Jackson) performs really well on throughput and latency when the processing unit is stronger than the data connection, as in the case of the Dell 1 Mbit/s setup. This is because of the small size of the messages generated by Avro.

IV. Discussion

A. Communication Standards

The results show that XMPP and Web Services have low throughput and high latency compared to the other middleware, that JSON is a better alternative to XML, and that binary serializers are even faster.

This means that, for distributed systems where the XMPP and Web Services server needs to run on the processing units of the DERs, there are better alternatives to XMPP, Web Services, and XML.

As the prevalent communication standards currently advocate for XMPP and Web Services middleware and XML serialization, they should consider other, newer middleware and serialization, limit their scope to centralized systems or only use these middleware and this serialization as the reference choice, and recommend using other middleware and serialization.

B. Guidance

For the setups, the best combinations are Dell 100 Mbit/s, Raspberry Pi 10 Mbit/s, and Beagle Bone 1 Mbit/s, which makes the best use of the available processing unit and data connection, and the choice between them depends on the required performance.

The best performing middleware choices are ICE, ZeroMQ, YAMI4, and WAMP.

ICE really does an excellent job on performance, but it does not support Publish–Subscribe, which could hurt the performance in real-world cases.

ZeroMQ and YAMI4 have varying throughput performance, and WAMP has low latency performance.

For serialization, JSON generally performs better than XML and binary performs even better, with the two best

performing serializers being ProtoStuff and ProtoBuf (ProtoStuff).

We should consider that the serializers that produce compact output could have better performance with very strong processing units, slow data connections, and long distances between DERs because of the messages being transmitted faster over the Internet. Avro (Jackson) and MsgPack produce the most compact output.

C. Previous Results

Compared to the results of the authors' previous work on middleware, ZeroMQ, YAMI4, and ICE still have the best performance, and the performance of WAMP over the setups is better than when measured only on the Raspberry Pi 100 Mbit/s setup used for the earlier work for throughput, but worse for latency. The difference is that, with multiple setups, the varying performance of ZeroMQ, YAMI4, and WAMP can now be seen, and ICE leads in performance with multiple setups.

For serialization, ProtoStuff and ProtoBuf (ProtoStuff) still have a clear performance advantage with multiple setups, and Smile (ProtoStuff) and Fast-Serialization still have the 3rd and 4th best performance, with their varying performance for the setups now clearly visible. The most interesting result with multiple setups is that compact serializers such as Avro (Jackson) and MsgPack have a clear advantage when the processing units are much stronger than the data connection.

The performance of the messaging patterns, which previously showed Request–Reply doing worse than Push–Pull and Publish–Subscribe, is now different depending on the setup, with the choice still dependent on the use case.

D. Economics

Without considering who will be paying for the processing unit and data connection, the manufacturer, owner, or aggregator, the economic implication for the processing unit is a price up to 200$ at present, with a Raspberry Pi Zero costing 5$, a Beagle Bone Black costing 60$, a Raspberry Pi 3 costing 40$, and a Nvidia Jetson TK1 costing 190$.

The implications of adding up to 200$ to the price of the DER are probably small, as the price of most DERs is at least 10.000$, whereas the implications of adding a monthly subscription fee for the internet connection could result in negative customer feedback, becoming costly over time depending on the country and the data usage.

The idea of having a processing unit and data connection is that the owner earns money by providing flexibility and ancillary services either directly to the System Operators or through an aggregator.

Then, the question becomes how much more money can be earned depending on the services that can be provided, which depends on the processing unit and data connection.

It should be considered that using compact serialization with a strong processing unit could improve the performance with low bandwidth data connections, which might save money over the lifetime of the DER, especially in countries with expensive data connections.

V. Conclusions

The results show that there are better alternatives to using XMPP and Web Services for middleware and XML for serialization as advocated for by the prevalent communication standards.

ICE, ZeroMQ, YAMI4, and WAMP are good choices for middleware, with ICE providing the best performance. The characteristics of the middleware should, however, also be taken into consideration.

For serialization, JSON performs better than XML, and binary serializers perform even better, with the obvious tradeoff of not being human-readable. The serializers with the best performance by far are ProtoStuff and ProtoBuf (ProtoStuff).

Comparison of setups shows that there is a clear correlation with stronger processing units and faster data connections providing better performance, but only when the data connection fits with the processing unit in performance, which means that the best combinations are Beagle Bone 1 Mbit/s, Raspberry Pi 10 Mbit/s, and Dell 100 Mbit/s, except when using a compact serializer, which requires a stronger processing unit compared to the data connection.

The results show the performance provided depending on the processing unit and data connection, and when this is linked to the required performance for the services required by the power grid, a cost–benefit analysis could show the return on investment for processing units and data connections.

The correlation between the processing unit and data connection on performance can be used to avoid spending money on strong processing units or data connections without getting a clear benefit on performance.

The results show that a throughput of 1000 message per second and a latency less than 1 ms per message can be achieved with strong processing units and a fast data connection, which should give an idea of which services can be provided by the DERs.

Future work should be done on the impact on the performance of sending measurements and control commands over the internet, by comparing the distance to the impact on throughput and latency.

Acknowledgment

Sponsored by the project, PROActive INtegration of sustainable energy resources enabling active distribution networks (PROAIN)

References

[1] M. Albano, L. L. Ferreira, L. M. Pinho and A. R. Alkhawaja, "Message-oriented middleware for smart grids," *Computer Standards & Interfaces*, vol. 38, pp. 133-143, 2015.

[2] L. Qilin and Z. Mintian, "The state of the art in middleware," *Information Technology and Applications (IFITA)*, vol. 1, pp. 83-85, 2010.

[3] A. Dworak, M. Sobczak, F. Ehm, W. Sliwinski, and P. Charrue, "Middleware trends and market leaders 2011," in *Proc.*

International Conference on Accelerator and Large Experimental Physics Control Systems, vol. 111010, 2011.
[4] B. Petersen, H. Bindner, S. You, and B. Poulsen, "Smart grid serialization comparison," in *Proc. Science and Information Organization (SAI) Computing Conference*, London, 2017, in press.
[5] B. Petersen, H. Bindner, S. You, and B. Poulsen, "Smart grid communication middleware comparison," in *Proc. International Conference on Smart Cities and Green ICT Systems (SmartGreens)*, Porto, 2017, in press.
[6] B. Petersen, H. Bindner, S. You, and B. Poulsen, "Smart grid communication comparison," in *Proc. International Conference on Innovative Smart Grid Technologies*, Torino, 2017, in press.
[7] R. E. Mackiewicz, "Overview of IEC 61850 and benefits," in *Proc. Power Systems Conference and Exposition*, Atlanta, 2006, pp. 623-630.
[8] C. McParland, "OpenADR open source toolkit: Developing open source software for the smart grid," in *Proc. Power and Energy Society General Meeting*, San Diego, 2011, pp. 1-7.
[9] M. Uslar, S. Rohjans, M. Specht, and J. M. G. V ázquez, "What is the CIM Lacking?" in *Proc. Innovative Smart Grid Technologies Conference Europe (ISGT Europe)*, Gothenburg, 2010, pp. 1-8.

Bo Petersen is currently with the Center for Electric Power and Energy at the Technical University of Denmark as a Ph.D. student. He received his B.Sc. and M.Sc. degree in software engineering from the Technical University of Denmark. His research interests cover software architecture for the integration of renewable energy and management of distributed energy resources in the Smart Grid.

Henrik W. Bindner is a senior scientist and the head of the Energy System Operation and Management Research Group in the Center for Electric Power and Energy (CEE) at the Technical University of Denmark (DTU).
He received his M.Sc. degree in electrical engineering from the Technical University of Denmark, Lyngby, Denmark, in 1988. Since 1990, he has been with the Risø National Laboratory for Sustainable Energy, Roskilde, Denmark. He has been focusing on developing technologies and control schemes enabling DER units to participate in the control of the power system. One of his main activities has been participation in establishing the experimental facility SYSLAB.

Bjarne Poulsen received his M.Sc. from Aalborg University and Ph.D. from the Technical University of Denmark. He has many years of industrial experience and has worked as an associate professor at the Technical University of Denmark since 2001. His research interests is within industrial distributed IT systems, with focus on the energy sector. He worked on the implementation of the IEC 61400-25 standard, and participated in the research projects Virtual Power Plant and EDISON.

Shi You received his M.Sc. and Ph.D. in electrical engineering from Chalmers Institute of Technology, Sweden in 2007 and DTU (Technical University of Denmark) in 2010 respectively. Currently, he is a research scientist at the Center for Electric Power and Energy, Department of Electrical Engineering at DTU. His main research interests include market-based control, management and integration of distributed energy resources, planning, operation and management of active distribution networks and integrated energy systems.