Smart Grid Communication Middleware Comparison
Distributed Control Comparison for the Internet of Things

Bo Petersen[1], Henrik Bindner[1], Bjarne Poulsen[2] and Shi You[1]
[1]*DTU Electrical Engineering, Technical University of Denmark, Anker Engelunds Vej 1, 2800, Kgs. Lyngby, Denmark*
[2]*DTU Compute, Technical University of Denmark, Anker Engelunds Vej 1, 2800, Kgs. Lyngby, Denmark*

Keywords: Smart Grid, Internet of Things, Communication Middleware, RMI, XML-RPC, CORBA, ICE, Web Services, OPC UA, XMPP, WAMP, YAMI4, ZeroMQ.

Abstract: Communication between Distributed Energy Resources (DERs) is necessary to efficiently solve the intermittency issues caused by renewable energy, using DER power grid auxiliary services, primarily load shifting and shedding. The middleware used for communication determines which services are possible by their performance, which is limited by the middleware characteristics, primarily interchangeable serialization and the Publish-Subscribe messaging pattern. The earlier paper "Smart Grid Serialization Comparison" (Petersen *et al*. 2017) aids in the choice of serialization, which has a big impact on the performance of the communication as a whole. This paper identifies the dis-/advantages of the different middleware, shows that there are better alternatives to Web Services and XMPP, and gives guidance in choosing the most appropriate middleware depending on the context. YAMI4 and ZeroMQ are generally the strongest candidates for Smart Grid distributed control, but WAMP should also be considered in the future.

SCITEPRESS
SCIENCE AND TECHNOLOGY PUBLICATIONS

1 INTRODUCTION

With an increased share of Renewable Energy in the future Smart Grid, the problems caused by Renewable Energy producing energy intermittently have to be solved to ensure an efficient and reliable supply of energy.

The most efficient solution to solve these problems is to match the energy consumption to the production by moving the consumption and production of energy. This is done by controlling the DERs, requiring communication to exchange measurements and react to control commands.

The communication middleware used is important for the success of communication measured by the probability of delivery within the timeframe defined by the power grid service offered by the DERs. Which in the case of frequency corrections and load shedding is milliseconds to minutes, while for load shifting it is minutes to days.

In the context of the Internet of Things, with hardware constrained System on Chip (SoC) devices and low bandwidth data connections for the DERs, the use of efficient middleware is essential for achieving a high probability of delivery within short timeframes.

The use of certain middleware is advocated for by the prevalent communication standards, including IEC 61850 (Mackiewicz 2006), OpenADR (McParland 2011) and the Common Information Model (Uslar *et al*. 2010).

An important part of the communication is the serialization used with the middleware, covered in the previous paper "Smart Grid Serialization Comparison" (Petersen *et al*. 2017).

The current state of the art is a handful of papers (Albano *et al*. 2015) (Qilin and Mintian 2010) (Dworak *et al*. 2011). These are limited by the available middleware at the time, the limited number of middleware compared, the lack of Smart Grid characteristics considered and the lack of recommendations for the choice of middleware.

The hypothesis of this paper is that there are better alternatives than the middleware advocated for by the prevalent communication standards, especially considering constrained SoC devices and low bandwidth data connections.

The aim of the paper is to compare the possibilities of middleware primarily for distributed control, to show the dis-/advantages of a broad range of middleware, and provide guidance in choosing the most appropriate middleware for the given use case.

Petersen, B., Bindner, H., Poulsen, B. and You, S.
Smart Grid Communication Middleware Comparison - Distributed Control Comparison for the Internet of Things.
DOI: 10.5220/0006303302190226
In *Proceedings of the 6th International Conference on Smart Cities and Green ICT Systems (SMARTGREENS 2017)*, pages 219-226
ISBN: 978-989-758-241-7
Copyright © 2017 by SCITEPRESS – Science and Technology Publications, Lda. All rights reserved

2 METHODS

The comparison is done in Java, as most middleware is available for Java.

2.1 Middleware Choices

Choosing the best composition of communication middleware for the comparison is important to give the best guidance in choosing the right middleware and to show that there are better alternatives to the middleware advocated for by the prevalent communication standards.

Web Services (W3C 2016) and XMPP (XSF 2016) are included because of these communication standards. Jetty (Eclipse 2016) Web Services are used, because Jetty is one of the most prominent embedded Java web servers, and an embedded web server is a requirement for distributed systems.

Vysper (Apache Mina 2016) and Smack (Realtime Ignite 2016) have been used for XMPP as Vysper is the only embedded Java XMPP server, and Smack is one of the most widely used Java XMPP clients.

OPC UA is included primarily because of its heavy use in industrial automation and because of a number of scientific articles (Lehnhoff *et al*. 2011) (Srinivasan *et al*. 2013) proposing its use with IEC 61850. Prosys OPC UA (Prosys 2016) is used because it is one of the few mature Java OPC UA SDK's.

Oracle RMI (Oracle 2016), Apache XML-RPC (Apache 2016) and Oracle CORBA (OMG 2016) are included because of their heavy use in distributed systems, along with ZeroC ICE (ZeroC 2016) which is a mature modern middleware with promising performance.

As oppose to the previously mentioned middleware, which are well-established mature technologies and have been in use for years, a number of new modern middleware have been included.

ZeroMQ (iMatix 2016) have been included to show the capabilities of message queue middleware while avoiding the use of a broker (used by most other message queue middleware), which is ill-suited for distributed systems. JeroMQ (JeroMQ 2016) was chosen because it is the only native Java implementation of ZeroMQ and still has excellent performance.

WAMP (Tavendo 2016) is included to show the capabilities of using Web Sockets. WAMP is used for Web Sockets because it adds an API layer for Request-Reply and Publish-Subscribe, as oppose to using raw binary Web Sockets. Jawampa (Matthias247 2016) is used because it is the only native Java implementation.

Inspirel YAMI4 (Inspirel 2016) is included because it is a really interesting project that is built specifically for cyber-physical systems with a strong performance, message prioritization, and restricted memory consumption specifically designed for constrained SoC devices.

2.2 Performance Comparison

For the quantitative performance comparison for Smart Grids, three messaging patterns (Request-Reply, Push-Pull, and Publish-Subscribe) are used.

Request-Reply (figure 1) is used with older middleware to poll for measurement data, without knowing when new measurements are available.

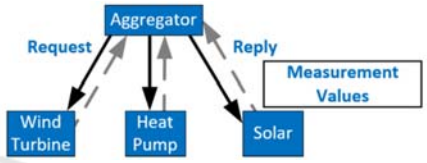

Figure 1: Request-Reply messaging pattern.

Push-Pull (figure 2) is used to send control commands to a device, preferably asynchronously, with only an acknowledgment of receipt returned.

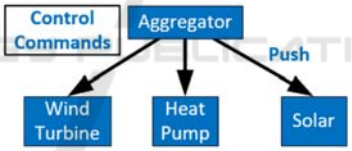

Figure 2: Push-Pull messaging pattern.

Publish-Subscribe (figure 3) is used to subscribe to measurement data, with the data returned when new data is available, which makes this pattern much more efficient than Request-Reply.

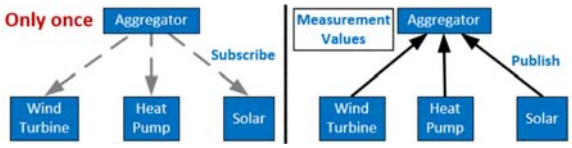

Figure 3: Publish-Subscribe messaging pattern.

For middleware supporting Publish-Subscribe, a combination of Publish-Subscribe and Push-Pull should be used for measurement data retrieval and delivery of control command, while for the other middleware a combination of Request-Reply, and Push-Pull, must be used instead.

Three different message sizes (1 kB, 10 kB, and 20 kB) are used for the comparison. They are chosen to cover the range of message sizes generated by serialization of IEC 61850 data model classes from the previous paper "Smart Grid Serialization Comparison" (Petersen *et al*. 2017), which generate output in the range between 2 and 12 KB.

Both string and binary message types are used, because serialization creates either string or binary output, with some middleware handling binary messages more efficiently and some only supporting string messages.

The performance measurements primarily consist of the average number of messages that can get from one device to another during a unit of time (throughput) and the average time it takes to get a message from one device to another (latency).

While measuring the throughput and latency, the package loss is measured in the form of the percentage of messages not received, and the memory is measured by the consumption during the whole test run for a given middleware.

To summarize, performance is measured for each middleware for the following:

- Throughput by size (1 kB, 10 kB, 20 kB), message type (string, binary), and messaging pattern (Request-Reply, Push-Pull, Publish-Subscribe).
- Latency by size (1 kB, 10 kB, 20 kB), message type (string, binary), and messaging pattern (Request-Reply, Push-Pull, Publish-Subscribe).
- Package loss by messaging pattern (Request-Reply, Push-Pull, Publish-Subscribe)
- Memory use by server and client.

The test was performed with two Raspberry Pi 3's (model B), with one running the server, as a DER, supplying measurement values, receiving control commands, and the other running the client, as an aggregator, getting measurement values, sending control commands.

The devices are connected by a 1 Gbit Ethernet, with 100 Mbit network interfaces, which ensures that the only limiting factor for throughput is the devices.

Because of the limitation of the 100 Mbit bandwidth, the theoretical maximum bandwidth utilization allows for 12500 messages of 1 kB/s, which is 12.5 MB/s.

The data loss is measured by the percentage of the total amount of messages not delivered for the 6 tests (String 1 kB, String 10 kB, String 20 kB, Binary 1 kB, Binary 10 kB and Binary 20 kB) for each messaging pattern (Request-Reply, Push-Pull, and Publish-Subscribe).

The memory consumption is measured for each middleware, by taking the memory used after setting up the tests, but before initializing the middleware and running the tests, and comparing it to the memory consumption after all tests.

2.3 Characteristics Comparison

The qualitative characteristics comparison compares the capabilities of the middleware and development related characteristics that should be considered along with the performance of the middleware.

One thing that is particularly important for certain Smart Grid use cases is message prioritization to ensure that control commands can get through even with a high amount of traffic.

The messaging patterns supported by the middleware are very important for use cases with high network utilization and a requirement for fast control command delivery.

Interchangeable serialization is important because middleware that supports it generally has a higher throughput and lower latency because serialization that is more efficient can be used.

Middleware that can run on SoC devices and scale to a high degree of traffic, because of their limited consumption of memory, is essential for distributed control systems, which use constrained SoC devices.

For the development related characteristics, the available resources in the form of documentation, the development effort needed, the size of the community (mailing lists, Q & A's, tutorials, etc.) and the license are important to consider.

To sum up, the following characteristics are compared:

- Message prioritization
- Messaging patterns
- Interchangeable serialization
- SoC scalability
- Resource quality
- Development effort
- Community size
- License

3 RESULTS

When interpreting the results, and deciding on the middleware to use it is important to first consider the characteristics of the middleware and then the performance needed for the given context.

3.1 Performance Comparison

The average measured throughput seen in figure 4-9, shows that binary data is more efficient for middleware that supports it and that Publish-Subscribe is much more efficient than Request-Reply.

Figure 10-15 shows the average latency, which is in addition to serialization, except for XML-RPC, XMPP, and WAMP that already serializes the messages.

One of the most interesting performance results is that the bandwidth utilization is quite stable between 10 kB and 20 kB message sizes for all messaging patterns and message types, which can be seen by comparing the throughput in MB/s between figure 5 & 6.

It should be noted that for OPC UA and XMPP the test could only be run with 100 iterations because of the memory consumption, which caused them to crash with 1000 iterations.

The performance of ICE, YAMI4 and ZeroMQ is especially impressive and for large messages, they all reach the limits of the network bandwidth at around 12 MB/s with overhead.

It should also be noted that Request-Reply has to transmit messages from the client to the server and then back, which doubles the average latency for the network, and reduces the theoretically possible throughput compared to the other messaging patterns.

The performance also shows how the middleware that does not support interchangeable serialization

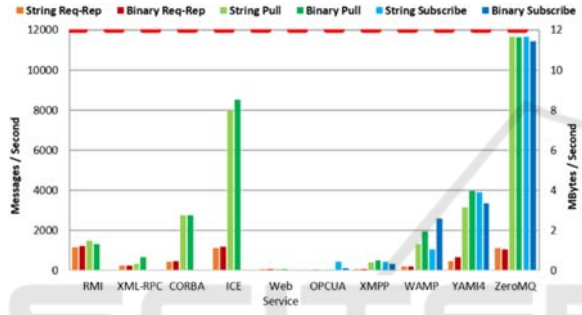

Figure 4: Throughput (1 kB messages).

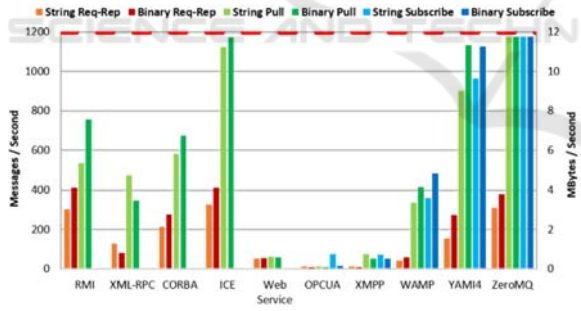

Figure 5: Throughput (10 kB messages).

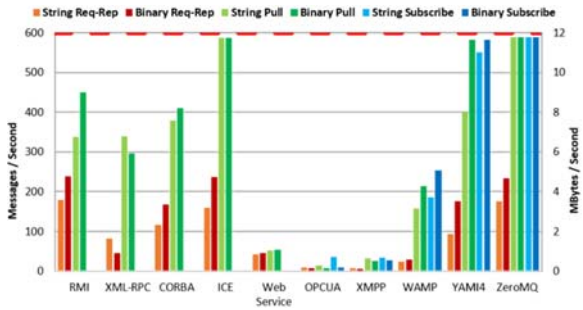

Figure 6: Throughput (20 kB messages).

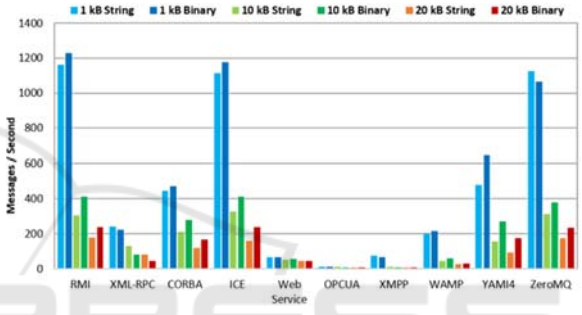

Figure 7: Throughput (Request-Reply pattern).

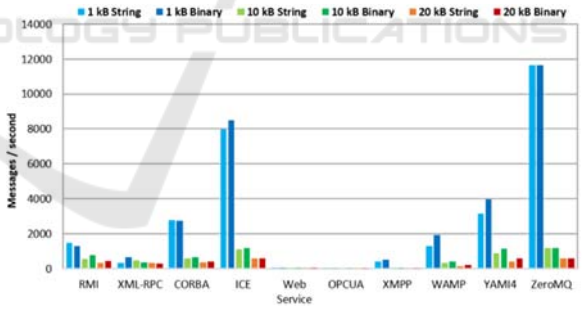

Figure 8: Throughput (Push-Pull pattern).

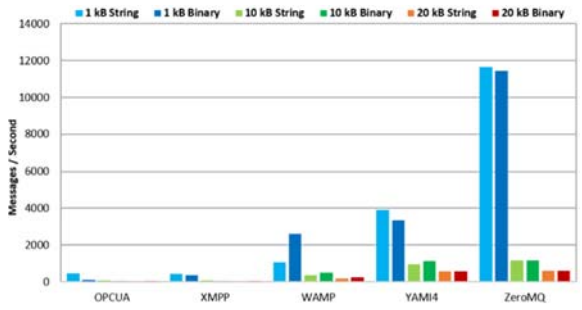

Figure 9: Throughput (Publish-Subscribe pattern).

(XML-RPC, XMPP, and WAMP) pays the price for serializing the already serialized data.

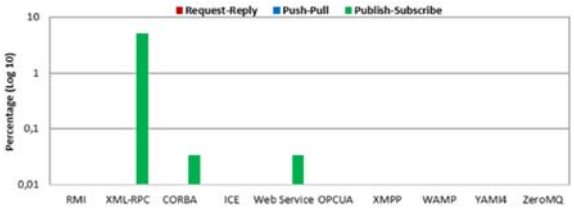

Figure 16: QoS Package loss.

The data loss can be seen in figure 16. Most middleware delivered all messages during the performance test. The test also shows that there is no data loss for Request-Reply and Publish-Subscribe, only for Push-Pull, and only for XML-RPC, CORBA and Web services, which would not be the case without a stable high-bandwidth data connection.

The memory consumption of the middleware is shown in figure 17, which is important to run the middleware on hardware constrained SoC devices.

Most middleware use less than 20 MB for the server and client, with 3 of them using less than 2 MB, which is quite impressive. XMPP uses much more memory than the other middleware, which is especially problematic seeing as it could only be tested with 100 iterations because of its memory consumption, which is also the case for OPC UA. On the other hand, RMI, CORBA, and YAMI4 use almost no memory, which makes them particularly well suited for running on SoC devices.

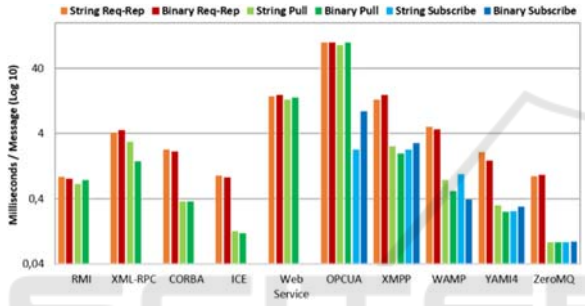

Figure 10: Latency (1 kB messages).

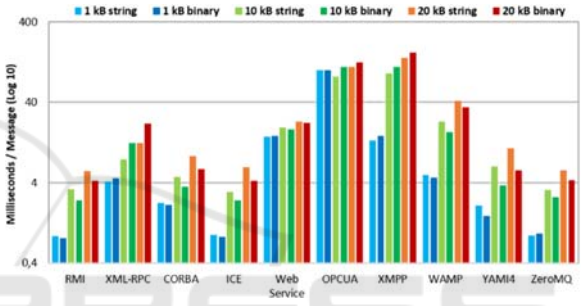

Figure 13: Latency (Request-Reply pattern).

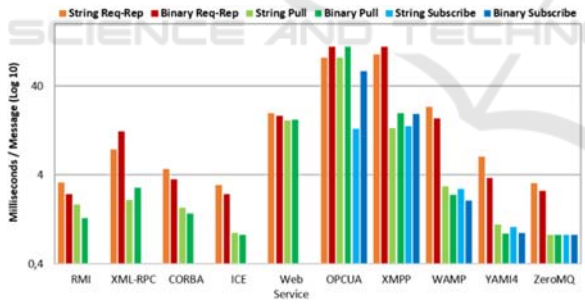

Figure 11: Latency (10 kB messages).

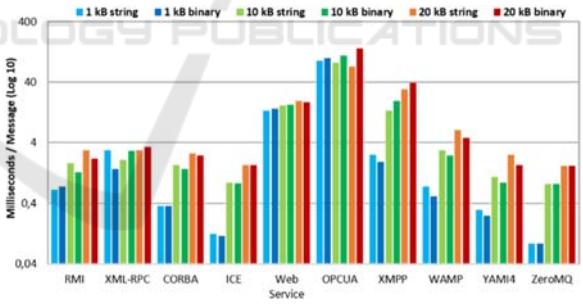

Figure 14: Latency (Push-Pull pattern).

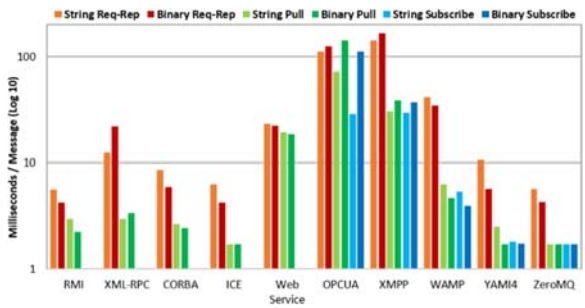

Figure 12: Latency (20 kB messages).

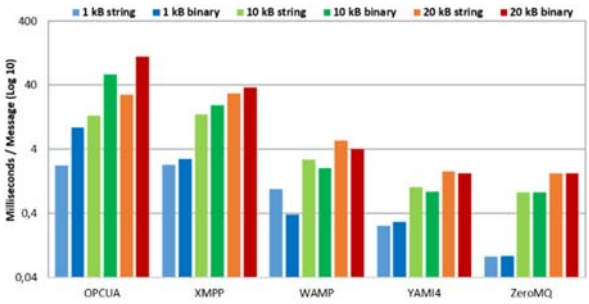

Figure 15: Latency (Publish-Subscribe pattern).

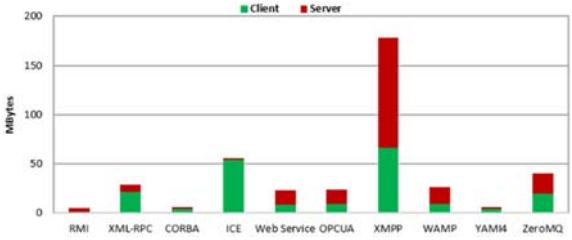

Figure 17: Memory use.

3.2 Characteristics Comparison

The comparison of middleware characteristics (table 1) shows what the middleware are capable of natively.

Only YAMI4 natively support prioritization of messages, which makes it especially suited for use cases with a large bandwidth utilization for measurement data exchange, and a requirement for fast control command delivery.

Publish-Subscribe is only supported by half the middleware, while all middleware, except RMI, support Push-Pull asynchronously, both of which are required to make efficient communication possible and all support Request-Reply.

Interchangeable serialization is supported by all middleware except XML-RPC and XMPP (which only support XML), and WAMP (which only support JSON and MessagePack).

Determining whether a middleware is scalable on SoC devices, is quite subjective and for the test, SoC scalability is based on whether they can do 1000 iterations on a Raspberry Pi 3, which OPC UA and XMPP cannot.

The quality of the available resources (manual, tutorials, examples), the required development effort (based on implementation effort for the comparison), and the size of the community (based on StackOverflow.com and Google search) are quite subjective and should be judges based on the given use case.

The license of the middleware can be decisive in the choice of middleware. Luckily, the only middleware that is closed source and only available with a paid license is OPC UA, while the only middleware that requires a paid license for commercial use are ICE and YAMI4.

4 DISCUSSION

When choosing the middleware, the first thing to consider is the middleware characteristics, which should be used to limit the number of middleware candidates, and the performance comparison should then be used to find the best candidates for the use case.

4.1 Characteristics Comparison

The license is especially important for commercial products, and the SoC scalability is essential for using the middleware on SoC devices.

The development characteristics (resource quality, development effort, and community size) is especially important for small projects, but also for bigger projects, because of maintainability.

Interchangeable serialization is very important to achieve the highest throughput and lowest latency, but require the serialization to be chosen carefully.

The Publish-Subscribe messaging pattern is necessary for a high degree of measurement data exchange, and for use cases where the DER getting

Table 1: Middleware characteristics.

	Message Prioritization	Messaging Patterns	Interchangeable Serialization	SoC Scalability	Resource quality	Development effort	Community size	License
RMI	No	*Req.-Rep.* Sync. Push-Pull	**Yes**	**Yes**	*Medium*	*Medium*	**High**	**Oracle BCL**
XML-RPC	No	*Req.-Rep.* *Push-Pull*	No	**Yes**	Low	**Very low**	*Medium*	**Apache v2**
CORBA	No	*Req.-Rep.* *Push-Pull*	**Yes**	**Yes**	**High**	**Low**	**High**	**Oracle BCL**
ICE	No	*Req.-Rep.* *Push-Pull*	**Yes**	**Yes**	**High**	**Low**	Low	*GPLv2*
Web Services	No	*Req.-Rep.* *Push-Pull*	**Yes**	**Yes**	**High**	*Medium*	**Very high**	**Apache v2**
OPCUA	No	**All**	**Yes**	No	Low	Very high	*Medium*	Commercial
XMPP	No	**All**	No	No	Low	High	**Very high**	**Apache v2**
WAMP	No	**All**	No	**Yes**	*Medium*	**Very low**	Very low	**Apache v2**
YAMI4	**Yes**	**All**	**Yes**	**Yes**	**High**	**Very low**	Very low	*GPLv3*
ZeroMQ	No	**All**	**Yes**	**Yes**	**High**	**Low**	**High**	**MPLv2**

For easier reading: **Big Advantage**, **Advantage**, *Neutral*, Disadvantage, Big Disadvantage.

data does not know how often data is sampled by the DER supplying the measurement data. Also, the Push-Pull pattern, with asynchronous push, is important for use cases requiring middleware supporting low latency control command.

Prioritization is important for getting control commands delivered within the given timeframe when large amounts of measurements are being exchanged.

4.2 Performance Comparison

With the comparison of binary and string message types, the increase in throughput for the middleware is up to 40 percent for the majority of the tests as shown in figures 4 - 9. But the real gain from using binary messages comes from the smaller sizes produced by the binary serializers which are up to 5 times smaller, than the corresponding string serializers, as shown by the earlier paper "Smart Grid Serialization Comparison" (Petersen *et al*. 2017).

Because of the stable bandwidth utilization, the gain from the messages being up to 5 times smaller with binary serialization means that the throughput is increased by up to 5 times. In addition to the up to 40 percent increase in throughput because of the middleware being faster with serialization, the total gain from using binary serialized data with an interchangeable serialization middleware is up to 7 times higher throughput.

The gain from using modern middleware also comes from them supporting the Publish-Subscribe messaging pattern which results in around 2-3 times higher throughput which is shown in figures 4 - 6 by comparing the throughput of Publish-Subscribe for the 5 middleware that support it to the throughput of Request-Reply for all the middleware.

This gain in throughput for Publish-Subscribe is in addition to the advantage of avoiding the problems with Request-Reply polling of measurement data, which include using the wrong polling interval, which will either cause a loss of measurements or waste bandwidth by getting the same measurements more than once. Even when using the correct polling interval, a few messages will be lost, because of the communication request not being executed with the exact same interval as the hardware polling.

The average latency shown in figures 10-15 show how long it takes for a control command to get to a DER on average, but it is only a small part of the latency of sending a control command over the Internet, as opposed to the comparison, which uses Ethernet. Still the results show that asynchronous Push-Pull improves the latency by 3-4 times, which can clearly be seen in figure 11, where the limit of the bandwidth is not reached and the messages are big enough for the results to be clear.

The results also show that interchangeable serialization, like with throughput, also improves the latency by about 2 times for messages of half the size, which is seen in figure 14, by comparing the 10 kB messages to the 20 kB messages. Which means that if the message size is reduced by 5 times, then the latency is improved by 5 times, in addition to the gains from the middleware having faster latency for binary messages.

The data loss of the compared middleware is minimal and should not affect most use cases, but in those few affected cases, middleware with data loss, should off course be avoided, which includes XML-RPC, CORBA, and Web Services, as shown in figure 16. This is however only for Push-Pull when a stable high-bandwidth data connection is used.

The memory consumption is important for use cases using SoC devices and for scaling up to very high throughput use cases. It should be noted that the measured memory consumption for OPC UA and XMPP are for 100 iterations, which means that if they could be run with 1000 iterations they would use a lot of memory. However, even excluding this difference, the memory consumption differs by at least a factor of 10, as shown in figure 17.

4.3 Guidance

When choosing whether to use middleware advocated for by the prevalent communication standards (Web Services and XMPP), it should be considered that they have terrible performance, to begin with. Especially when considering that Web Services does not support Publish-Subscribe and XMPP does not support interchangeable serialization, which is extremely problematic with low bandwidth data connections and high throughput use cases.

The fact that these standards are moving from Web Services to XMPP, makes the choice even easier with SoC devices because XMPP can only be used for use cases with low traffic where the rest of the control system uses very little memory, and then still risks failure due to running out of memory.

YAMI4 and ZeroMQ have a strong performance and advantages in characteristics, primarily Publish-Subscribe that makes them strong candidates to use for Smart Grid control system.

ZeroMQ has better performance than YAMI4 and can be used for commercial products, while YAMI4 has lower memory consumption and QoS prioritization, which makes YAMI4 better suited for

distributed control on SoC devices with low bandwidth data connections, and ZeroMQ better suited for centralized or hierarchical control on strong hardware with high bandwidth data connections.

WAMP should also be considered because it uses Web Sockets, which is an emerging web standard, which is being broadly used, and even though it has lower performance, does not support prioritization and interchangeable serialization, it does support MessagePack which is a quite efficient serialization format and might support either more serialization formats or interchangeable serialization in the future. When it is matured and for use cases not requiring prioritization, it could possibly be one of the best choices.

5 CONCLUSIONS

The paper shows that using message based middleware in the form of YAMI4 or ZeroMQ has excellent performance, and provide the best characteristics, while other papers (Albano *et al.* 2015) just state that message based middleware is the obvious choice for Smart Grid communication because of it being message based by nature.

The paper shows the results of comparing a large carefully chosen range of middleware, including modern middleware, considering Smart Grid requirements, the impact of serialization and SoC devices, for distributed control with recommendations for the choice of middleware.

Future work could be done by combining serialization and communication middleware to show the impact of combinations of the two, and to run performance tests on high and low bandwidth data connections, using constrained and more capable hardware.

REFERENCES

Petersen, B., Bindner, H., Poulsen, B., You, S. (2017). Smart Grid Serialization Comparison. In *SAI Computing Conference (unpublished)*, London, 2017.

Mackiewicz, R. E. (2006). Overview of IEC 61850 and Benefits. *In IEEE PES Power Systems Conference and Exposition, Atlanta, 2006.* pp. 623-630.

McParland, C. (2011), OpenADR open source toolkit: Developing open source software for the Smart Grid. *In IEEE Power and Energy Society General Meeting, San Diego, 2011.* pp. 1-7.

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

Albano, M., Ferreira, L. L., Pinho, L. M., Alkhawaja, A. R. (2015). Message-oriented middleware for smart grids. *In Computer Standards & Interfaces.* 2015, 38: 133-143.

Qilin, L., Mintian, L. (2010). The state of the art in middleware. *In Information Technology and Applications (IFITA).* 2010.

Dworak, A., Sobczak, M., Ehm, F., Sliwinski, W., Charrue, P. (2011). Middleware trends and market leaders 2011. *In Conf. Proc.. Vol. 111010. No. CERN-ATS-2011-196. 2011.*, 2011.

W3C, 2016, Web Services [Online]. Available: http://www.w3.org/2002/ws/. [Accessed 25 11 2016].

XSF, 2016, XMPP [Online]. Available: https://xmpp.org/. [Accessed 25 11 2016].

Eclipse, 2016, Jetty [Online]. Available: http://www.eclipse.org/jetty/. [Accessed 24 11 2016].

Apache Mina, 2016, Vysper [Online]. Available: https://mina.apache.org/vysper-project/. [Accessed 24 11 2016].

Realtime Ignite, 2016, Smack [Online]. Available: https://www.igniterealtime.org/projects/smack/. [Accessed 24 11 2016].

Lehnhoff, S., Mahnke, W., Rohjans, S., Uslar, M. (2011). IEC 61850 based OPC UA Communication-The Future of Smart Grid Automation. *In 17th Power Systems Computation Conference (PSCC 2011).* 2011. Stockholm.

Srinivasan, S., Kumar, R., Vain, J. (2013). Integration of IEC 61850 and OPC UA for Smart Grid automation. *In 2013 IEEE Innovative Smart Grid Technologies-Asia (ISGT Asia).* 2013.

Prosys, 2016, OPC UA [Online]. Available: https://www.prosysopc.com/products/opc-ua-java-sdk/. [Accessed 25 11 2016].

Oracle, 2016, RMI [Online]. Available: http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html. [Accessed 25 11 2016].

Apache, 2016, XML-RPC [Online]. Available: https://ws.apache.org/xmlrpc/. [Accessed 25 11 2016].

OMG, 2016, CORBA [Online]. Available: http://www.corba.org/. [Accessed 25 11 2016].

ZeroC, 2016, ICE [Online]. Available: https://zeroc.com/. [Accessed 25 11 2016].

iMatix, 2016, ZeroMQ [Online]. Available: http://zeromq.org/. [Accessed 25 11 2016].

JeroMQ, 2016, JeroMQ [Online]. Available: https://github.com/zeromq/jeromq. [Accessed 25 11 2016].

Tavendo, 2016, WAMP [Online]. Available: http://wamp-proto.org/. [Accessed 25 11 2016].

Matthias247, 2016, Jawampa [Online]. Available: https://github.com/Matthias247/jawampa. [Accessed 25 11 2016].

Inspirel, 2016, YAMI4 [Online]. Available: http://www.inspirel.com/yami4/. [Accessed 25 11 2016].