Smart Grid Communication Infrastructure Comparison

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

Published in:
Proceedings of 8th IEEE International Conference on Smart Grid Communications

Publication date:
2017

Document Version
Peer reviewed version

Citation (APA):
Abstract—Communication between Distributed Energy Resources and aggregators is required 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 connections include 1, 10, 100 and 100 Mbit/s. The results show that there are better alternatives to XMPP and Web Services middleware and XML serialization as advocated for by the prevalent communication standards, and gives guidance in choosing the best software and hardware depending on the use case.

Keywords—Smart Grid; Internet of Things; Infrastructure; Processing Unit; Raspberry Pi; Beagle Bone Black; Data Connection; Bandwidth; Communication; RMI; XML-RPC; CORBA; ICE; Web Services; OPC UA; XMPP; WAMP; YAMI4; ZeroMQ; Serialization; XML; JSON; YAML; FST; Key; JAXB; Jackson; XStream; ProtoStuff; Gson; Genson; SnakeYaml; MegPack; Smile; ProtoBuf; BSON; Hessian; CBOR; Avro;

I. INTRODUCTION

Communication between Distributed Energy Resources (DERs), and aggregators is a requirement to improve the efficiency of power use in the future Smart Grid and to support more reliable and robust operation.

The need for these improvements comes from the increase in intermittent renewable energy, primarily solar and wind power [1], which is problematic with a traditional load following power grid, requiring energy storage and a production following Smart Grid.

The Smart Grid will partly be achieved by DERs providing power system ancillary services, such as primary frequency control for stability and load shifting and shedding for a production following power grid.

Some of these ancillary services require a high probability of delivery of measurements and control commands within a short timeframe, which depends on the processing unit of the DERs, the data connection between the DERs, the communication middleware, and the serialization used with the middleware.

The state of the art by previous papers [2] [3] [4], and the authors’ earlier work [5] [6] [7], is that they investigate the performance and characteristics of middleware and serialization for communication, but do not investigate the impact on performance by the processing units of the DERs and the data connection between the DERs.

The aim of this paper is 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 attempt 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 include 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 and serialization, with different processing units and data connections, especially compared to the middleware and serialization recommended by prevalent communication standards for Smart Grids, including the IEC 61850 [8], OpenADR [9] and CIM [10].

II. METHODS

The tests were performed in Java using Oracle JDK 1.8.0_111, on 10 setups which combine a pair of processing units and a data connection.

The processing units included consists 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 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)

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

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

A message in this context means a set of measurements or a control schedule (as describe 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 second period:

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

The tests are performed with 3 messaging patterns. Request-Reply for polling of measurements, Push-Pull for control commands, and publish-subscribe for receiving measurements when they are made:

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

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

To measure the time a message is sent and received, to get the latency, the same clock is used by 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 [8] data model (1 logical node per message, except the request message of Request-Reply, which is an IEC 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 millisecond per message (fig. 2).

While to get throughput speeds of 280 – 510 messages per second, and a latency below 3 milliseconds per message, adequate processing units like the Raspberry Pi’s or Dells 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 of fewer than 140 messages per second and a latency above 4 milliseconds 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 milliseconds per message.

The results show that a slow data connection of 1 Mbit/s limits the performance with all the processing units, while 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 Dells to be utilized fully. The 1 Gbit/s data connection require even stronger processing units than the ones tested to be utilized.

The results also show that the Dell’s need a 100 Mbit/s data connection, to perform optimally, while 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 1Mbit/s.

For the messaging patterns, the results show that when the data connection is fast, the Request-Reply pattern performs better, while 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 milliseconds 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 regards to throughput, ICE, ZeroMQ, and WAMP perform the best with a utilization above 70 percent, while Web Services, XMPP, and OPC UA have a utilization below 31 percent.

On latency, ICE, ZeroMQ, RMI and YAMI4 generally do an excellent job, with utilization above 70 percent, while Web Services, XMPP, and OPC UA has a utilization below 30 percent. But to get the full picture of how the middleware perform, fig. 6. and 7. shows the throughput and latency utilization respectively, for the middleware across the setups.

While 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 not as simple.

The throughput for ZeroMQ and WAMP varies a lot depending on the setup, but is generally good, while YAMI4 and RMI perform modestly on all setups.

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

While the latency for ZeroMQ, RMI and YAMI4 is relatively stable, for WAMP it is very similar to 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 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 of fewer than 1.1 milliseconds 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) has a throughput and latency utilization above 90 percent, all XML serializers are below 20 percent, all JSON serializers except 1 are faster than the XML serializers and binary serializers are faster than string serializers.

Fig. 11. and 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 much strong 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, which do better with slow data connections.
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 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 choice for reference.

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, while it does not support Publish-Subscribe which could hurt the performance in real world cases.

ZeroMQ and YAMI4 have varying throughput performance, while 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).

Though it should be considered 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, while 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, while 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 like 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 previous showed Request-Reply to do better than Push-Pull and Publish-Subscribe, is now different depending on the setup, with the choice still depending 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 implications 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 is 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 be costly 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. Conclusion

The results show that there are better alternatives to using XMPP & 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, but the characteristics of the middleware should 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).

The 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 of less than 1 millisecond 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.

REFERENCES