Evaluation and Extension of OPC UA Publish/Subscribe MQTT Binding

Hannes Raddatz*, Eman Mahmoud*, Fabian Hölzke*, Peter Danielis†, Dirk Timmermann* and Frank Golatowski*
* Institute of Applied Microelectronics and CE
† Department of Computer Science University of Rostock, Rostock, Germany
Email: hannes.raddatz@uni-rostock.de

***Abstract*—Currently, Open Platform Communication Unified Architecture (OPC UA) is nominated as a reference standard to meet all Industry 4.0 demands with one protocol, enabling an efficient communication between devices in industrial systems. Recently, OPC UA has been extended by a publish/subscribe pattern to support multicast communication since there is a need for an efficient data aggregation and distribution to cope with the increasing number of controllers, sensors, and measured values in industries. Message Queuing Telemetry Transport (MQTT) is the most widely used publish/subscribe communication protocol and is based on a central broker for message exchange. In this paper, the communication latency of OPC UA end devices exchanging data over MQTT as a broker-based middleware is investigated and compared to the client/server-based communication. In particular, we analyze the use case of one-to-many communication for a crane model from the material handling domain. We provide a prototype implementation of the OPC UA MQTT extension and our evaluation results show the reduced traffic overhead and communication latency compared to client/server-based communication.**

***Index Terms*—OPC UA, publish/subscribe, OPC UA part 14, MQTT, material handling, Industry 4.0, open62541**

I. Introduction

The development of the Industrial IoT in the context of the Industry 4.0 trend requires the standardization and sharing of data from individual components of industrial companies. The large number of industrial mechatronic systems, automation tools, and industrial communication systems represents a new challenge in this regard [1]. The requirements placed on Industry 4.0 components or systems to achieve horizontal and vertical interoperability over the entire life cycle are flexibility, adaptability, transparency, and many others [2]. Currently, Open Platform Communication Unified Architecture (OPC UA) is developing as a reference standard that fulfills all Industry 4.0 requirements with one protocol thereby offering the flexibility of chosing from different communication protocols like HTTP, TCP, and others [3]. OPC UA is supported by a reasonable amount of PLCs on the industrial market. Despite the fact that OPC UA starts to replace legacy communication protocols in facilities and offers the advantage of semantic data modelling, the common communication patterns are remained unchanged. Therefore, one-to-one connections, also known as client/server or request-response scheme are still the majority. Consequently, such connections are also used in cases where many devices are interested in information of a single device, or a monitor device is gathering specific information from all controllers in a network. These cases are also known as one-to-many and many-to one communication. Their combination is known as many-to-many communication. This demand has also been realized by the OPC Foundation and consequently, they published the OPC UA standard amendment part 14, the publish/subscribe extension [4]. This paper investigates the communication between OPC UA end devices over Message Queuing Telemetry Transport (MQTT) as a broker-based middleware to exchange data from the OPC UA address space in term of message latency, targeting the use case one-to-many and proposing further extensions to support functionality of this communication pattern.

The remainder of this paper is organized as follows. Section II discusses related work and Section III introduces OPC UA part 14. Section IV presents the developed OPC UA-MQTT binding. Section V describes results from the testbed evaluation before Section VI concludes the paper.

II. Related Work

In terms of the time spent to send and receive messages and the message size, the work in [5] compares the performance of MQTT and OPC UA separately, as opposed to our approach. It was mainly observed that MQTT is more suitable for the distribution of messages when there is a large number of subscribers that are interested in receiving the same topic. Hence, it is advisable to use MQTT as extension to UPC UA to support this use case which we do in this paper. In [6], the performance and resource usage of the most common protocols in the field of industrial automation and IoT are evaluated. In addition to OPC UA and MQTT, Data Distribution Service (DDS) and Robot Operating System (ROS) are analyzed. Unlike our approach, the work again evaluates OPC UA and MQTT separately and concludes that OPC UA's major strength is the semantic description of the address space while MQTT is a lightweight publish/subscribe protocol which focuses on a small footprint and low network bandwidth usage. Interestingly, the authors state that UPC UA publish/subscribe does not include any QoS mechanism so it should be combined with MQTT, which supports additional QoS principles (re-sending of messages, caching of topic data

on the broker), which motivates the contribution in our paper. In [7], the authors evaluate the performance of MQTT in three different distributions of publishers and subscribers, i.e., system architectures under various traffic load scenarios. They arrive at the conclusion that the underlying network layers such as TCP and IP have a stronger impact on message latency than the system architecture. Again, they propose to combine the semantic publish/subscribe capability of MQTT with the information model of OPC UA to replace fieldbusses in industrial applications, which supports our contribution. The authors propose an OPC UA publish/subscribe implementation also using the open62541 software development kit in [8]. Their focus is on implementing it in the binary message format with brokerless transport over time-sensitive networking-enabled Ethernet thereby achieving realtime while our focus is on non-realtime networks using the broker-based lightweight MQTT protocol. The work in [9] also examines the performance of OPC UA as a client/server variant and contrasts it with various publish/subscribe options. Unlike in our paper, the communication latencies that can be achieved are not based on measurements, but on expert estimates. However, the work offers an interesting overview of which OPC UA implementation is suitable for which application (Enterprise, human-machine interface, control-to-control, field level). Performance bottlenecks for OPC UA publish/subscribe communication using typical application profiles from industrial automation are investigated in [10]. However, the authors mainly analyze resource utilization rather then latencies and use switches with Internet Group Management Protocol (IGMP) support for multicast filtering, i.e., no application-layer broker such as an MQTT broker.

III. OPC UA Publish Subscribe Amendment

To prepare OPC UA for the future, the publish/subscribe extension of OPC UA specification part 14 [4] was released to support many-to-many, one-to-many and many-to-one communication. However, the OPC Foundation is limiting the publish/subscribe extension to the use case one-to-many. Such a communication pattern consists of up to three entity roles. The publisher provides its data to the network, while the subscriber is the consumer of this information. Both entities are defined to be decoupled, maybe also without having knowledge about the existence of each other. Publish/subscribe protocols are subdivided into broker-based and broker-less middleware. A broker-based approach uses the third entity role, the broker, to link data publisher and consumer. Both register at the broker with their specific interest, such as topic or queue of interest. The broker forwards data sent by a publisher to all interested subscribers. The second type, the broker-less middleware, delegates the broker functionality to the underlying network technology and requires in general apriori knowledge, such as UDP multicast addresses. The amendment can be interpreted to be protocol agnostic. However, for the broker-based middleware, MQTT and Advanced Message Queuing Protocol (AMQP) are highlighted and basic mapping information is provided. UDP multicast is described as a candidate for broker-less middleware.

An MQTT broker offers the optional feature to retain a message and forward it later to the subscribers, supporting the decoupling of publisher and subscriber. However, the broker functionality is out of scope of part 14.

The amendment discusses also different types of data encoding for publish/subscribe communication. To send data to the cloud or to analytics systems, JavaScript Object Notation(JSON) encoding is highlighted, especially the combination of MQTT/AMQP with JSON is presented as a common use case. A binary data encoding that is already used in the common OPC UA client/server communication pattern may also be supported by publish/subscribe extensions. The combination of UDP multicast and binary encoding, also known as UA Datagram (UADP), is recommended for frequent data exchange in production environments.

Applying JSON encoding in a scenario where only devices with an OPC UA stack exist is not very advantageous because the payload of the message is increased and OPC UA data processing takes place after reception anyway. In case of data distribution by an OPC UA publisher to MQTT subscribers with no additional OPC UA stack, JSON can be a valid choice for cloud access or data processing when no OPC UA stack is required or possible.

Following, components are described that are defined in part 14 to extend OPC UA by publish/subscribe capabilities [11]. Component details that are beyond the scope of this paper are not mentioned due to the high number of available parameters in the OPC UA publish/subscribe extension.

- DataSet (DS): List of name-value pairs that contain real values of entities in the OPC UA address space. A name-value pair is referred to as DataSet field.
- DataSetMetaData (DSMD): Additional semantic information about the content of DS, such as the full name of variables and its data types. Transmitted only on demand or change of published data.
- Published DataSet (PDS): A structure that contains the DSs to be collected from the OPC UA address space for publication. Further, it contains configuration and semantic meta information (DSMD), properties, IDs of standardized DataSets and filter functions.
- DataSetMessage (DSM): Contains the PDS and additional header information.
- NetworkMessage (NWM): Final message sent by the publisher that can contain multiple DSMs and additional information, such as publisher ID.

The publisher holds in Figure 1 a list of predefined PublishedDataSets and PubSubConnections and manages the publication of these data sets over the available connections. A PubSubConnection defines the transport protocol and middleware to be used. Further, it contains the WriterGroups that create the NetworkMessages. A WriterGroup can manage multiple DataSetWriters and holds information for the creation of a single NetworkMessage, such as the publishing interval. A DataSetWriter creates a DataSetMessage out of a single

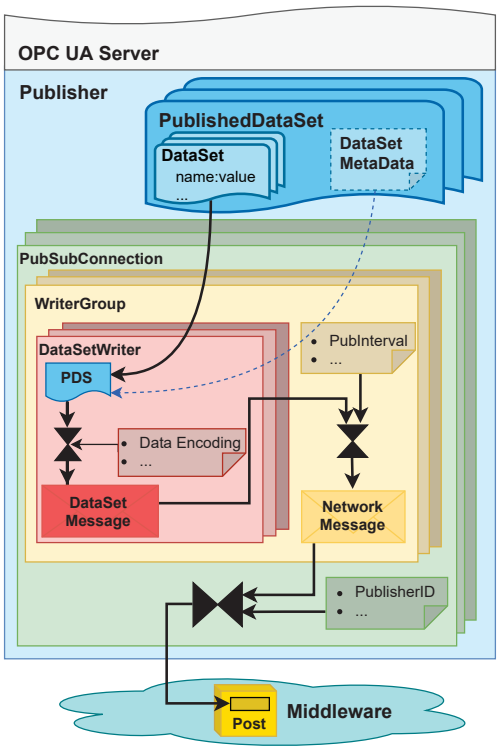

Fig. 1. Publisher components of OPC UA standard part 14.

PDS using a specific data encoding. One PDS can be used by multiple DataSetWriters.

The subscriber receives in Figure 2 a NetworkMessage via a secific middleware and forwards it internally to the PubSubConnection component responsible for this middleware. The PubSubConnection on subscriber side contains multiple ReaderGroups. These groups orchestrate several DataSetReaders and hold some common information. A DataSetReader is responsible to receive and filter NetworkMessages and decode the included DSMs with the help of the DataSetMetaData. The extracted DataSets are finally processed according to parameters defined in the component SubscribedDataSet, such as storing the received values in objects of the local address space. Finally, the dashed line in the Figures 1 and 2 represent the acyclical exchange of the DSMD information. According to OPC UA part 14, OPC UA servers and clients can be both publisher and subscriber. However, we question the role of the client as publisher. The amendment specifies that a publish/subscribe implementation should support at least one transport profile. Such a profile is a combination of specific mappings for DSM, NWM, and transport protocol. Examples for a transport profile are UADP binary data encoding (includes definition of header fields for DSM and NWM) over UDP multicast or JSON encoding (with DSM and NWM details) over MQTT (TCP). Publisher and subscriber can therefore agree on a common transport profile. However, this involves direct communication between both entities. An idea to solve this issue is proposed in Section IV-A.

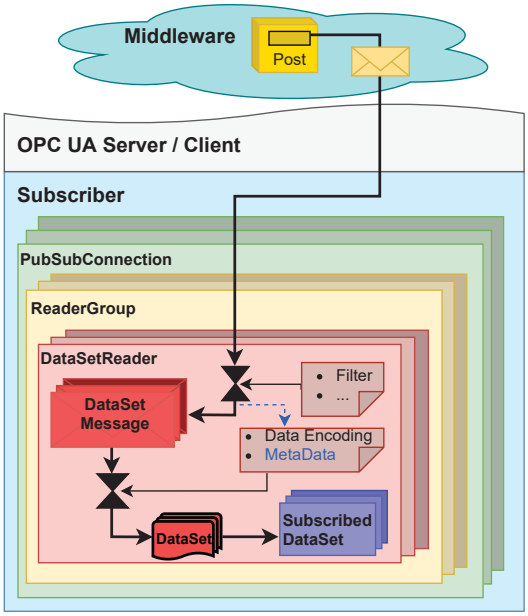

Fig. 2. Subscriber components of OPC UA standard part 14.

IV. OPC UA - MQTT Binding

OPC UA supports a subscription mechanism for the client/server pattern, eliminating the need to frequently request the same data to receive updates. This mechanism is still a one-to-one connection, realized by a separate connection for each interested client. This requires to create an individual session and may encrypt context for each client that may leads to a notable resource consumption, given a certain number of interested clients. Assuming resource constrained devices in a facility, this behaviour may represent a bottleneck in certain use cases for the application of OPC UA in large networks.

OPC UA part 14 was published to counter this issue and adopt best practices from the IoT domain, such as decoupling of end devices and asynchronous communication. By applying the publish/subscribe extension, the described bottleneck can be avoided with the help of a broker-based middleware by shifting the resource-intensive session management to a broker device with sufficient resources. Alternatively, this task can be delegated to the underlying network infrastructure by using a broker-less middleware like UDP multicast.

Since MQTT is currently the most known communication protocol applying the publish/subscribe pattern, we used it to realize a prototypical implementation of the OPC UA publish/subscribe extension. OPC UA provides the advantage of a semantic data model that adds meta information to the actual data to support the interpretation of data on the receiver side. However, MQTT is advertised as a rather simple protocol, being agnostic of the content it is transmitting. Therefore, there is a need to bridge the gap between these protocols by encapsulating the OPC UA data (actual data and associated meta information) before sending it as a payload of a MQTT message in order to reconstruct the information at the subscriber. Although protocol mappings are described in part

14, only the UADP has been elaborated in detail, while MQTT and AMQP are described only superficially. It is helpful that the standard outlines the binary and JSON encoding more precisely, since a complete mapping from OPC UA to MQTT is currently missing. The OPC UA standard part 14 only states that in case of a broker-based middleware the data exchange via the broker should be parameterized according to specifications such as queue name, routing key, and topic.

To realize a data distribution over MQTT, the WriterGroup needs to be modified since it holds the primary configuration flags to create the DSM. We defined the MQTT topic as one of these parameters. Therefore, a WriterGroup is limited to one MQTT topic and creates only NWMs with this topic in the defined publication interval. This behaviour is compliant with the respective definitions in part 14.

To be able to add non-OPC UA devices to the network in future, we changed the data encoding from binary encoding to JSON encoding. According to part 14, this parameter is part of the DataSetWriter. Using another data encoding than binary encoding adds the effort to distribute this convention to all subscribers before the middleware-based communication channel can be used.

The NWM created by the WriterGroup is finalized with information of the PubSubConnection entity, such as its PublisherID. The message is now ready to be transmitted via a function that realizes the MQTT publish task. However, a registration or initial connection establishment is required before an MQTT publisher can send messages to a broker. After this process, the NWM can be send to the broker according to the specified publishing interval. While the NWM is the payload of the MQTT message, the MQTT topic, defined in the WriterGroup, is stored in the dedicated header field of the MQTT message. The QoS level is also defined in the MQTT header and can be realized as an additional parameter of the WriterGroup. The goal of this message building process is to create an MQTT message that can be consumed by any standard-compliant MQTT broker. Therefore, we did not modified the broker in our scenario.

The message processing on the subscriber side is similar to the one on publisher side. The subscriber needs to register at the broker to declares its interest in a specific topic. This requires again an agreement out of band between publisher and subscriber to exchange information via a common topic. This can either be realized in combination with the settlement on a common data encoding or by a convention, such as a Companion Specification (a common subset of device data).

Once receiving the MQTT message from the broker, the OPC UA subscriber forwards the incoming MQTT message to the PubSubConnection that has registered at the broker. Here, the NMW is extracted from the MQTT message and delegated to all ReaderGroups. In contrast to a WriterGroup, the ReaderGroup simply forwards the received NWM withour modification to its DataSetReaders. As shown in Figure 2, a DataSetReader filters the NWM by a predefined set and disassembles the NWM to DSMs. In a next step, the predefined JSON encoding and the pre-exchanged DSMD are used to

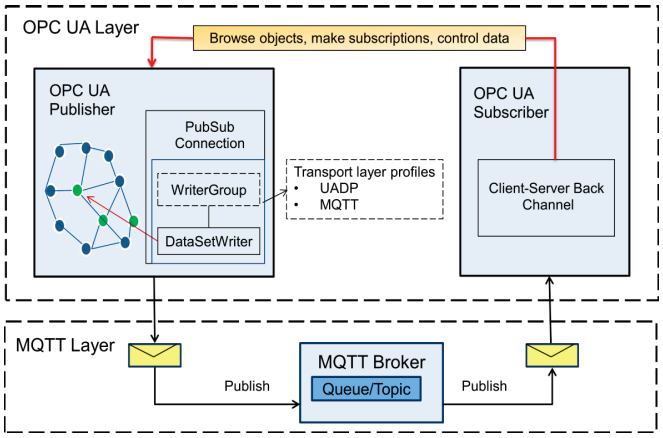

Fig. 3. Concept of remote publication mechanism

decode the DSM in one step to a PDS and further into its DataSets. As a last step, the DataSets are processed according to instructions saved as so called subscribed DataSets. The most simple example of such an instruction is the logging of this data or a print to standard output of the application. This last step can also be used to integrate the received information into the local address space of the subscriber device, assuming an OPC UA server.

Since our proposed OPC UA-MQTT binding is an early version, security is currently out of scope. However, one can rely on MQTT built-in security with the advantage of a rather simple configuration of the existing approach MQTT standard. This approach requires trust in the broker since the communication is only secured on the subsections publisher-broker and broker-subscriber. On the other hand, a complete end-to-end security between publisher and subscriber would require shared key material between all subscribers.

Currently, the exchange of DSMD is not fully integrated. A simple solution could be the definition of a separate topic at the broker.

A. Remote Publication Mechanism

Although the proposed concept used to enable a broker based publish/subscribe data exchange is convenient for providing an efficient data distribution using MQTT, it adds constraints for the subscriber regarding the client features of OPC UA, such as the ability to control and browse the data provided by the publisher. The reason behind these constraints is based mainly on the new release part 14 of OPC UA standard, the decoupling of publisher and subscriber as well as the one way principle of publish/subscribe pattern. Hence, the subscriber has no control over the data published by the publisher. Therefore, we propose the idea to extend the publish/subscribe concept in order to circumvent this limitation. The subscriber shall be able to control the data published by the publisher.

An temporary feedback channel using client/server pattern between subscriber and publisher allows the subscriber to browse and modify the data in the address space of the publisher. The feedback channel is illustrated in Figure 3. In a next step, a new object is defined in the address space of

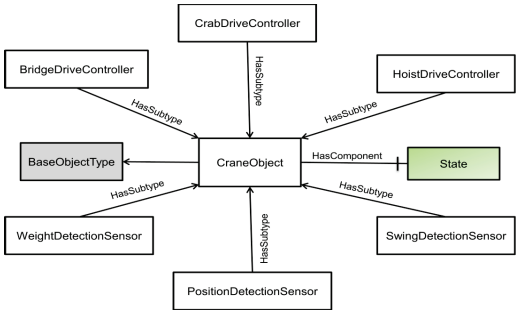

Fig. 4. Simple OPC UA data model of a material handling crane.

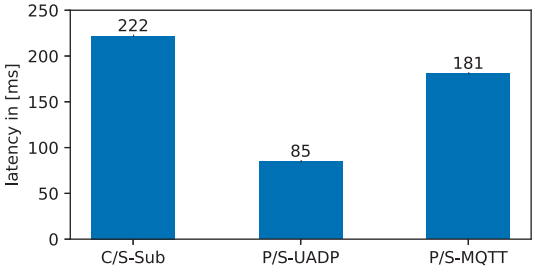

Fig. 5. Average of latency measurements.

the publisher. The object contains a *PublicationList* method that requires an array as input argument. The subscriber can invoke this method by using the feedback channel and provide an array with NodeIds of all data it is interested in to be published via publish/subscribe communication. Consequently, the publisher defines a new PDS, and adds DS fields according to the NodeId array. The last step is the modification of the WriterGroup and DataSetWriter that are part of the publish/subscribe extension.

V. Evaluation and Results

This section contains a description for the testbed, scenarios and measurements used to evaluate the implementation of the proposed concept. As a basis we used the open source OPC UA implementation open62541 for C programming language. We created the data structure of a crane from the material handling domain in the address space of the server, see Figure 4. Selected variables of this model are published via different communication channels to evaluate our solution.

The testbed consists of a 100 Mbps full duplex switch and two end devices represented by Raspberry Pi 3 development boards. A router with DHCP and NTP servers is connected to the switch to provide IP addresses and time synchronization to the end devices.

We conducted measurements to evaluate the OPC UA performance using the three different communication channels client/server subscription, publish/subscribe using UDP multicast (UADP), and publish/subscribe using MQTT. First, the performance of OPC UA is evaluated regarding the overall transmission time of data form the OPC UA address space for the three different kinds of communication mentioned before. In a next measurement, we focus on the OPC UA performance using the MQTT channel and investigate the influence of different messages sizes on the communication.

A. Comparison of OPC UA communication channels

In this experiment, we measure the time required to exchange data between pure OPC UA client and server (subscription) as well as between OPC UA publisher and subscriber extensions in order to evaluate the performance for the different communication channels. The communication partners are located on different end devices in the network.

For the subscription via the client/server pattern, the measurement starts when the subscription request of the client is confirmed by the server. The measurements stops when the client receives a notification message from the server. The OPC UA server responds immediately after subscription with a first notification containing the current value of the subscribed variable.

In case of OPC UA publish/subscribe pattern with UADP, we measure the latency between subscriber and publisher using UDP as with OPC UA binary data encoding. Since this communication represents a broker-less middleware, the publisher sends its data to a predefined multicast address, while the subscribers listen to this network address. The measurement starts when the publisher sends its data and stops after the subscriber receives the published message.

The measurement of OPC UA publish/subscribe using MQTT with JSON data encoding is conducted as follows. We apply the open source MQTT broker from the Mosquitto project in our testbed. Therefore, three devices are involved in the data exchange. The publisher publishes and the subscribers subscribe to a specific MQTT topic at the broker. We defined two timing measurements in the communication flow. The first one is taken directly after the publisher sends a message to the broker. The second timestamp is taken when the subscriber received the forwarded message.

The average transmission time for each communication channel is presented in Figure 5. The delay to exchange a message using the client/server pattern is quite high compared to the time required to send the same message using one of the publish/subscribe channels. The communication between the publisher and subscriber using MQTT requires more time to exchange a message than the UADP variant. One reason is the delay introduced by the broker to process and forward the message. The UDP-based communication channel does not require this processing time as there is no broker in between.

B. Influence of message sizes on OPC UA communication via MQTT

The timestamps created in this scenario are basically similar to the former test with MQTT. However, we added two measurements on the broker. The first timestamp defines the point in time shortly after the broker received the published message. The second one is created right after the broker has forwarded the message to the subscriber. Figure 6 depicts an overview of these timing measurements.

Figure 7 shows the different averaged delay for every part of the communication between publisher and subscriber. The time

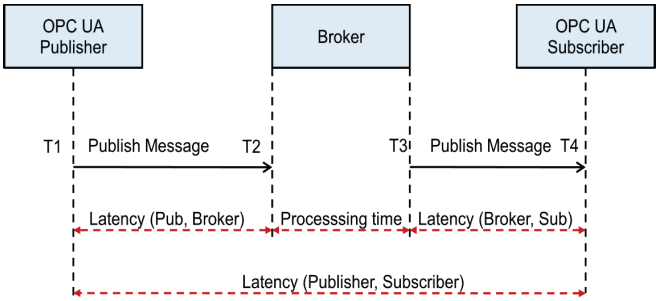

Fig. 6. Overview of latency measurements.

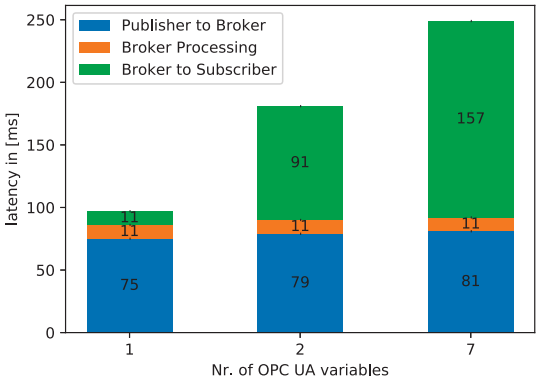

Fig. 7. Latency by number of transmitted OPC UA variables

required to receive the published message by the subscriber is increasing progressively as the message length increases. The latency between broker and subscriber shows an unexpected growth in transmission time regarding the size of message when compared to publisher-broker delay. The delay generated by the broker during processing and filtering the published message is almost constant for all message sizes. This confirms that MQTT is agnostic of the payload and its task is limited to filtering the incoming message based on the topic.

VI. Conclusion

The OPC UA standard with amendment part 14 provides a comprehensive solution for data distribution in industrial networks. It has the ability to map onto many communication protocols such as UDP and MQTT, improving the flexibility of industrial production and the interoperability between industrial systems. Additionally, it solves the scalability issue by supporting one-to-many communication with its new publish/subscribe part 14. In this work, OPC UA with its semantic data model has been extended by MQTT following the OPC UA publish/subscribe amendment to enable a broker-based data exchange. As both OPC UA and MQTT have different messaging concepts, the data mapping issue between OPC UA and MQTT has been solved in this work. In addition, we have proposed a remote publication mechanism as an extended approach. Using this mechanism, subscribers have the ability to control the data published via MQTT. Our evaluation showed that OPC UA using client/server subscription creates the largest delay for a data exchange between client and server. OPC UA publish/subscribe using UDP has advantages to send data in a local area network, as it generates a small overhead and the transmission time is lower when compared to our broker-based middleware using MQTT. On the other hand, OPC UA Publish/Subscribe using MQTT has the advantage of aggregating data in an effective way and also sending it into the cloud. Furthermore, the broker-less approach requires special network hardware such as IGMP switches, which can lead to high costs. A possible use case could be a monitoring device that is not bound to latency requirements and aggregates sensor information across the entire facility to send it to the cloud.

In future work, we will investigate the performance of this implementation in a larger network with multiple subscribers and test it against the other OPC UA communication channels.

Acknowledgement

This work has been achieved in the European ITEA project ”OPTimised Industrial IoT and Distributed Control Platform for Manufacturing and Material Handling” (OPTIMUM) and has been funded by the German Federal Ministry of Education and Research (BMBF) under reference number 01IS17027. We want to thank all partners in the OPTIMUM project for the stimulating discussions and their contributions to the project. All project partners can be found on https://itea3.org/project/optimum.html.

References

[1] P. Drahoš, E. Kučera, O. Haffner, and I. Klimo, “Trends in industrial communication and opc ua,” in *2018 Cybernetics & Informatics (K&I)*. IEEE, 2018, pp. 1–5.

[2] M. Schleipen, S.-S. Gilani, T. Bischoff, and J. Pfrommer, “Opc ua & industrie 4.0-enabling technology with high diversity and variability,” *Procedia Cirp*, vol. 57, pp. 315–320, 2016.

[3] H. Renjie, L. Feng, and P. Dongbo, “Research on opc ua security,” in *5th IEEE Conference on Industrial Electronics and Applications*, 2010.

[4] OPC Foundation, “OPC Unified Architecture Specification, Part14: PubSub,” 2018.

[5] M. S. Rocha, G. S. Sestito, A. L. Dias, A. C. Turcato, and D. Brandão, “Performance comparison between opc ua and mqtt for data exchange,” in *Workshop on Metrology for Industry 4.0 and IoT*. IEEE, 2018.

[6] S. Profanter, A. Tekat, K. Dorofeev, M. Rickert, and A. Knoll, “Opc ua versus ros, dds, and mqtt: performance evaluation of industry 4.0 protocols,” in *Proceedings of the IEEE International Conference on Industrial Technology (ICIT)*, 2019.

[7] P. Nenninger, M. Gierl, and R. Kriesten, “A contribution to publish-subscribe based communication in industrial applications,” in *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. IEEE, 2019, pp. 0424–0429.

[8] J. Pfrommer, A. Ebner, S. Ravikumar, and B. Karunakaran, “Open source opc ua pubsub over tsn for realtime industrial communication,” in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1. IEEE, 2018, pp. 1087–1090.

[9] A. Eckhardt, S. Müller, and L. Leurs, “An evaluation of the applicability of opc ua publish subscribe on factory automation use cases,” in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1. IEEE, 2018, pp. 1071–1074.

[10] A. Burger, H. Koziolek, J. Rückert, M. Platenius-Mohr, and G. Stomberg, “Bottleneck identification and performance modeling of opc ua communication models,” in *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, 2019, pp. 231–242.

[11] W. Mahnke, *OPC Unified Architecture OPC UA PubSub*. Unified Automation, 2018. [Online]. Available: https://industrie40.vdma.org/documents/4214230/27125486/4_PubSub_Mahnke_1542713548405.pdf