An OPC UA PubSub Implementation Approach for Memory-Constrained Sensor Devices

Quang-Duy Nguyen, Patrick Bellot, Pierre-Yves Petton

An OPC UA PubSub Implementation Approach for Memory-Constrained Sensor Devices

Quang-Duy NGUYEN
LTCI, Télécom Paris
Institut Polytechnique de Paris
91120, Palaiseau, France
quanguyen@telecom-paris.fr

Patrick BELLOT
LTCI, Télécom Paris
Institut Polytechnique de Paris
91120, Palaiseau, France
patrick.bellot@telecom-paris.fr

Pierre-Yves PETTON
Departement physique du système Ferroviaire
SNCF - Direction Innovation et Recherche
93210 Saint-Denis, France
pierreyves.petton@sncf.fr

***Abstract*—Open Platform Communications Unified Architecture (OPC UA) comprises 14 specifications to deploy an industrial system with reliability, security, and interoperability. While realizing this standard, the devices of the considering industrial system must have enough capabilities in computation and storage. It is a challenge in the Industrial Internet of Things (IIoT), a scenario in which field-level devices can be memory-constrained sensor devices. Tailoring OPC UA to fit such devices requires advanced programming skills; otherwise, system developers must simplify or remove some essential features of OPC UA. This paper presents another implementation approach to tailor OPC UA PubSub, a specification for the publish-subscribe messaging pattern, into memory-constrained sensor devices. This approach, titled OPC UA PubSub-C, proposes using a remote OPC UA server as a configurator to operate large-memory-footprint tasks for field-level devices.**

***Index Terms*—Industry, IoT, OPC UA, PubSub, Sensor Device, Memory-constrained, Configuration**

I. Introduction

Industrial Internet of Things (IIoT), together with other innovations in the industry such as Cyber-Physical Systems (CPS), form Industrie 4.0 [1]. Technically speaking, the Internet of Things (IoT) is a set of physical objects with communication capabilities, ranging from resource-constrained sensor devices to high-computational servers, that exchange data throughout internet connections. The physical objects in the IIoT are devices in the industry. One main concern of the IIoT is to enable memory-constrained sensor devices to work in industrial environments. Memory-constrained sensor devices are electronic devices equipped with one or several sensors that aim to generate sensed data and send them to data consumers. In general, they have random data access (RAM) memory sizes ranging from a few kilobytes (KB) to a few megabytes (MB) and a few MB of read-only memory (ROM). With such a minimal amount of memory, they can only work with small-footprint requirement software programs. It is a challenge for many standards in the industry, which are reliable, secure, and solid, but also require resources to afford such advantages.

Open Platform Communication Unified Architecture (OPC UA) is a widely used standard in the industry. It consists mainly of 14 specifications that provide conventions to deploy industrial devices, so they can collaborate with reliability, security, and interoperability. A system based on the OPC UA specifications is called an OPC UA system. The strong point of OPC UA is that it supports multiple interoperability levels [2]. It provides network protocols, security policies, and data formats solutions, which are fundamental elements for technical and syntactic interoperability. Also, OPC UA supports semantic interoperability by providing the address space and information model mechanisms. In detail, an address space is a collection of OPC UA nodes. They represent resources of an OPC UA system following the schema of an information model [3], [4]. OPC UA nodes can be accessed and discovered from the other devices. Another strong point of OPC UA is its scalability. It provides many profiles corresponding to different features, functionalities, or scenarios [5]. Of which, each profile is a group of conformance units. A conformance unit defines the minimum needed facets of OPC UA that must be carried out in system development.

Two roles engaged in an OPC UA system are data provider and data consumer. A data provider manages an address space, generates data, and sends them to one or several data consumers. Depending on the messaging pattern used in an OPC UA system, the two roles have different titles. With the request-response, data providers are servers, and data consumers are clients. With the publish-subscribe, data providers are publishers, and data consumers are subscribers. OPC UA specification 14, also called OPC UA PubSub, is dedicated to the publish-subscribe messaging pattern [6].

The advantage of OPC UA implies two drawbacks for memory-constrained sensor devices, which play the role of data providers. Concerning interoperability, an information space containing thousands of nodes can consume a significant amount of memory [7]. Concerning scalability, OPC UA designs some profiles for memory-constrained embedded devices, such as Nano Embedded Device Server Profile[1] (nano profile) or Micro Embedded Device Server Profile[2] (micro profile). However, implementing these profiles involves removing some OPC UA facets, such as a server with the nano profile or micro profile has no security in default.

This paper proposes an implementation approach to overcome the two above drawbacks. In a limited scope, this ap-

[1]http://opcfoundation.org/UA-Profile/Server/NanoEmbeddedDevice2017
[2]http://opcfoundation.org/UA-Profile/Server/MicroEmbeddedDevice2017

proach addresses only OPC UA PubSub. The idea is to create a version of OPC UA PubSub specified for memory-constrained sensor devices while respecting all OPC UA specifications, without reducing the information model or removing OPC UA facets. This implementation approach, called OPC UA PubSub-C, produces an OPC UA PubSub system that uses a configurator to configure all memory-constrained sensor devices and manage one unified information model for them. This system is called an OPC UA PubSub-C system. Even more, the configurator is encouraged to handle other features of an OPC UA PubSub system defined in specification 14.

The rest of this paper is organized as follows. Section II presents the background of this paper: the publish-subscribe communication pattern, OPC UA PubSub, and the memory consumption factors of an OPC UA PubSub system. Next, Section III focuses on this research's contribution: the OPC UA PubSub-C implementation approach. Section IV details a proof-of-concept experiment and analyzes its results. Section V presents some other OPC UA implementations and compares them with OPC UA PubSub-C. Finally, Section VI summarizes the content of this paper and opens a discussion.

II. Background

Publish-subscribe messaging pattern, with its advantages of selective distribution of message and real-time data offering, becomes an effective communication method in the IoT [8]. OPC foundation provides its version of this messaging pattern dedicated to OPC UA, called OPC UA PubSub, presented in specification 14. OPC UA PubSub proposes two modes of work: broker-less and broker-based. Each mode requires different resources, then costs differently.

A. Publish-Subscribe Messaging Pattern

The publish-subscribe messaging pattern is an asynchronous data exchange mechanism, in which some devices perform as subscribers and some as publishers. A subscriber needs to subscribe to a data source only once and receive newly generated data as soon as available. Data sources are parts of the publisher's side. Two communication modes between publishers and subscribers are the broker-based mode and the broker-less mode. The broker-based mode requires another computing device playing as a broker. The broker has two features. The first feature is to manage the information of data sources and the details required to create links between publishers and subscribers. This information is called a topic. When a subscriber subscribes to a topic, it is equivalent to when the subscriber subscribes to a data source related to the topic. The second feature of a broker is to collect data from publishers and forward them to subscribers. In the scope of this paper, the first feature is called topic management, and the second feature is called data forwarding. Therefore, it is possible to say that a broker provides two features: topic management and data forwarding. A typical example of the broker-based mode is the MQTT protocol. The broker-less mode uses the multicast approach to spread new data to subscribers. Instead of using topics and a broker as in the broker-based mode, publishers manage a list of subscribers' addresses or use a default multicast address domain of the TCP/IP stack. In this sense, the broker-less mode can profit from the default network infrastructure.

In the IIoT, one typical publisher can be a field-level device that equips one or several sensors. It is also called a sensor device. Each sensor of a sensor device is a data source. A subscriber is a computer that consumes sensed data collected from sensor devices. In the broker-based mode, a broker manages topics and forwards data between sensor devices and data consumers. This broker can be another computer or a gateway having enough capabilities in computation and storage. In the broker-less mode, a data consumer can request sensor devices to add its information, such as the address of the data consumer, into a custom multicast group on the sensor devices' side. Otherwise, when a data consumer and sensor devices agree to communicate through some default multicast addresses of the TCP/IP stack, the data consumer only needs to listen to these multicast addresses.

B. OPC UA PubSub System

OPC UA PubSub adopts the concept of the publish-subscribe messaging pattern. Also, this specification presents the other specific features of OPC UA. Figure 1 illustrates the prototype of an OPC UA PubSub system. A publisher manages an address space containing nodes representing its resources. In general, the address space contains a group of standard nodes defined by OPC UA and a group of nodes defined in a specific use case. The node identifiers of the former group link to a namespace titled namespace zero (ns0). Thus, these nodes are also known as namespace zero nodes. The nodes in the latter group can be called custom-specific nodes. New generated data from a node and its additional information are represented by a dataset field. The dataset writer encodes the dataset field into a dataset. Then, the dataset is put into a message. One message may contain several datasets. The message is sent from the publisher to the message-oriented middleware (MOM), which is a broker in the broker-based mode or a device supporting the multicast mechanism in the broker-less mode. Then, MOM forwards the message to one or several subscribers. When the message arrives at a subscriber, the dataset reader of the subscriber processes the received message. In addition to the above work, an OPC UA PubSub system also requires three features. First, registration management is a mechanism for publishers to be available in the system so that subscribers can query the list of publishers. Second, dataset metadata management is a mechanism to exchange dataset metadata which includes the semantic of a message. Third, security key management is a mechanism to distribute security information for both publishers and subscribers.

The broker-less mode of OPC UA PubSub uses the UDP transport protocol. Moreover, it defines UA Datagram Protocol (UADP) message mapping and proposes to use binary encoding to serialize data into UADP messages. After serialization, a UADP message is the payload of a UDP message.

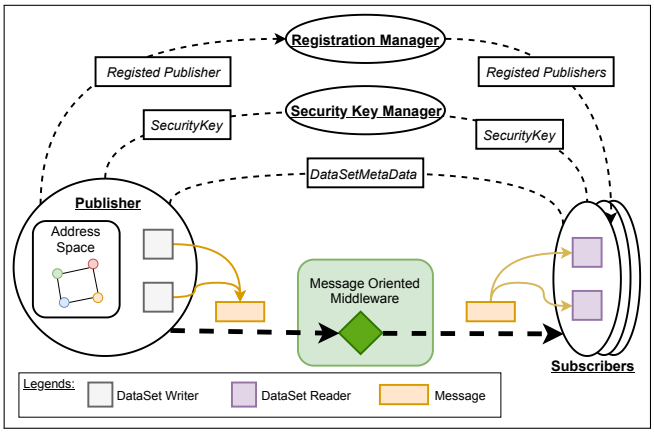

Fig. 1: The prototype of an OPC UA PubSub system

The broker-based mode of OPC UA PubSub reuses the MQTT and AMQP protocols. Therefore, OPC UA PubSub can profit from the advantages of these two protocols and can use MQTT and AMQP brokers available in the market. Moreover, OPC UA PubSub provides also two profiles called PubSub MQTT UADP profile[3] and PubSub AMQP UADP profile[4] that allows combining them with binary UADP message format.

C. Memory Consumption Factors of OPC UA

While reducing the memory footprint of an application, it is necessary to find the memory consumption factors. Each OPC UA communication mode produces a different memory footprint. This subsection compares all three OPC UA communication modes, including the OPC UA request-response mode, OPC UA PubSub broker-based mode, and OPC UA PubSub broker-less mode. There are three criteria to discuss.

First, the address space on the data provider side is the main factor for the memory cost. Loading a thousand nodes of an address space requires a significant amount of RAM [7]. At this point, there is no difference between the three communication modes: their systems need to manage an address containing namespace zero and custom-specific nodes.

Second, maintaining connection sessions consumes RAM. In the OPC UA request-response mode, the communication between a server and a client relies on TCP protocol. Thus, they must grant memory to manage their connection sessions. In the OPC UA PubSub broker-based mode, both MQTT and AMQP rely on TCP protocol to maintain connections between a broker and other components of a system. In this sense, this communication mode consumes memory for connection sessions. Different from the above, the OPC UA PubSub broker-less mode uses UDP protocol. UDP protocol is connectionless, then it has no waste for connection sessions.

Third, the pre-allocated buffer for temporary stocking messages consumes RAM. It is necessary to calculate the size of each message format. In general, the size of a binary format message is much smaller than the size of a JSON or XML format message with the same content. All three communication modes support binary message format. Moreover, the OPC UA request-response mode also supports XML format, and the OPC UA broker-based mode also supports JSON format.

Table I summarizes the memory-consumption factors that affects the three OPC UA communication modes. The first row, from the second column, lists the three OPC UA communication modes. The first column, from the second row, lists the criteria corresponding to three categories of factors. The intersection box between a communication mode and a criterion is the detail factor that consumes memory.

TABLE I: Memory consumption factors of three OPC UA communication modes

Factor \ Mode	**Request-response**	**PubSub broker-based**	**PubSub broker-less**
Address space	Namespace zero and custom-specific nodes		
Connection management	Sessions	MQTT or AMQP sessions	
Pre-allocated buffer	Binary size or XML size	Binary size or JSON size	Binary size

III. OPC UA PubSub-C

OPC UA PubSub-C is a new approach to implement OPC UA PubSub systems with memory-constrained sensor devices as publishers. OPC UA PubSub-C stands for OPC UA PubSub with a configurator. A configurator is a new role in OPC UA PubSub-C. The key of OPC UA PubSub-C relies on three criteria. First, all configurations and parameters are specified and simplified for sensor devices. Second, OPC UA PubSub-C systems use a configurator to manage a unified information model for all sensor devices. Third, a configurator could support other features required in specification 14, including registration management, dataset metadata management, security key management, topic management, and data forwarding.

A. Configurations and Parameters for Sensor Devices

A sensor device consists of several electronics units: a microprocessor, memory, power supply, analog-to-digital converter (ADC), network transceiver, and one or several sensors [9]. Of which, a sensor is a unit that measures a physical property of a phenomenon. It is necessary to distinguish between a sensor device and a sensor. While a sensor is a data source that provides sensed data, a sensor device manages serialization, security, and transportation. It is easy to see that in an OPC UA PubSub system, a sensor device can play the role of a publisher, and a sensor can play the role of a dataset writer. In this sense, since an OPC UA PubSub-C system is dedicated to sensor devices, the term "sensor device" is interchangeable with "publisher" and the term "sensor" is interchangeable with "dataset writer" in this kind of system.

While considering the memory-consumption factor presented in Section II-C, OPC UA PubSub-C tends to select the most lightweight configurations proposed by OPC UA PubSub. Concerning the data serialization, this approach chooses UA

[3]http://opcfoundation.org/UA-Profile/Transport/pubsub-mqtt-uadp
[4]http://opcfoundation.org/UA-Profile/Transport/pubsub-amqp-uadp

binary encoding with UADP message mapping to format data. The reason is that the size of a UADP network message formed by UA binary encoding is smaller than the other format; thus, a sensor device can grant less pre-allocated memory for each message. OPC UA PubSub-C suggests configuring a UADP network message to contain only datasets generated from one dataset writer at a time, since sensors may have different measurement schedules. The field representation should be *DataValue* which contains a sensed value, the sensed time in timestamp format, and the sensor's status. Concerning security policy and transport, OPC UA PubSub-C selects to implement only the profiles in the list proposed by OPC UA PubSub appropriate to the UADP message mapping. Therefore, the accepted security policy profiles are PubSub-Aes128-CTR[5], PubSub-Aes256-CTR[6], and None[7]. The accepted transport profile for the broker-less mode is PubSub UDP UADP[8]. In the broker-based mode, to recall, OPC UA PubSub reuses and supports both MQTT and AMQP protocols. However, OPC UA PubSub-C uses only the MQTT protocol for communication, since it is designed for lightweight and limited resources systems [10]. Thus, the accepted transport profile for the broker-based mode is PubSub MQTT UADP.

In OPC UA specification 14, *PublisherId*, *WriterGroupId*, and *DataSetWriterId* are three fundamental parameters. In detail, they are always integrated into a UADP network message. A subscriber can further process the message's payload based on these parameters. *PublisherId* is the identifier of a publisher. In OPC UA PubSub-C, a publisher is a sensor device; then, this parameter corresponds to the identifier of the sensor device, and it is called *DeviceId*. Next, *DataSetWriterId* is the identifier of a dataset writer. This parameter corresponds to the identifier of a sensor in OPC UA PubSub-C. In this case, it is known by the name *SensorId*. Finally, *WriterGroupId* is the identifier of an abstracted group of dataset writers sharing one or several similar features. OPC UA PubSub-C simplifies the dataset writer group into a group of sensors that have the same security policy profile and transport profile. For short, this parameter is called *GroupId*.

The following example is to clarify the meaning of the above parameters. A weather station is equipped with two sensors: one pluviometer to measure rain quantity and one thermometer to measure air temperature. This weather station is a component of an OPC UA PubSub-C system that implements the PubSub UDP UADP transport profile and the None security policy. The unique identifier of this weather station in this system is a *DeviceId*. The identifier of its pluviometer is a *SensorId* and the identifier of its thermometer is another *SensorId*. Since the data from the pluviometer and the data from the thermometer are secured and sent with the same method; then, they should be in the same group. The identifier of this group is a *GroupId*.

[5]http://opcfoundation.org/UA/SecurityPolicy#PubSub-Aes128-CTR
[6]http://opcfoundation.org/UA/SecurityPolicy#PubSub-Aes256-CTR
[7]http://opcfoundation.org/UA/SecurityPolicy#None
[8]http://opcfoundation.org/UA-Profile/Transport/pubsub-udp-uadp

B. OPC UA PubSub-C System

Three fundamental roles in an OPC UA PubSub-C system are publisher, subscriber, and configurator. As an OPC UA PubSub system, a publisher is a data provider, and a subscriber is a data consumer. However, a publisher has no address space management feature, and a message contains only datasets from one dataset writer. A configurator supports publishers by featuring address space management. This address space contains nodes of all publishers in an OPC UA PubSub-C system. Also, a configurator handles the other two fundamental features, which are registration management and dataset metadata management. It can optionally have the security key management feature. Moreover, in broker-based mode, the configurator can be a broker; in other words, it provides topic management and data forwarding features. Figure 2 illustrates the prototype of an OPC UA PubSub-C system.

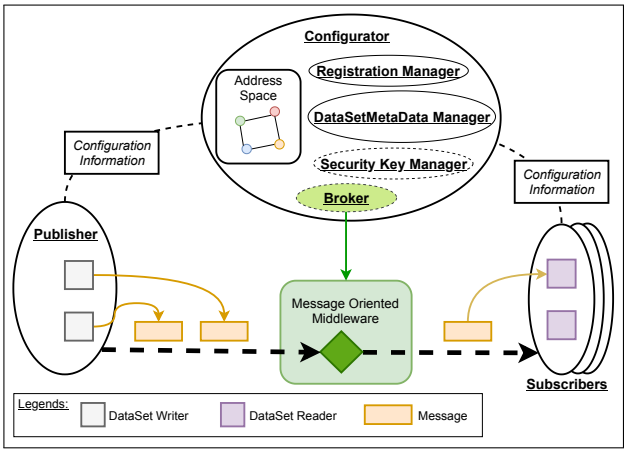

Fig. 2: Prototype of an OPC UA PubSub-C system

The operation of an OPC UA PubSub-C has two phases: configuration and data exchange. In the configuration phase, publishers expose their resources, parameters, and requirements to the configurator of an OPC UA PubSub-C system. The configurator uses the information from publishers to update the address space and sends back configuration information to publishers. Configuration information means the information for a component of the system to configure itself. Also, in this phase, subscribers can request the configurator for topics. The configurator sends back configuration information to subscribers. The configuration information for subscribers differs from the one for publishers. In the data exchange phase, publishers send messages to subscribers. The destination of the messages are multicast addresses in the broker-less mode or is the broker's address in the broker-based mode.

OPC UA PubSub-C proposes three new messages for the configuration phase, as follows.

- REGISTER: is a message sent from a publisher to a configurator. This message has two missions. First, it contains the default identifier of a sensor of the publisher. Also, it may contain other information, such as the security and communication modes that the publisher

supports. They allow the configurator to register the sensor. Second, the message contains dataset metadata corresponding to the sensor. The dataset metadata is necessary for the data exchange phase.

- QUERY: is a message sent from a subscriber to a configurator. This message contains the query content. For example, a query for a list of registered thermometers and their dataset metadata. Moreover, it may contain other information, such as the security and communication modes that the subscriber supports.
- CONFIGURE: is a message sent from a configurator to a publisher or a subscriber. This message contains the configuration information required by the publisher or subscriber so that they can start the data exchange phase. On the one hand, the configuration information fundamental to a publisher are *PublisherId*, *GroupId*, *SensorId*, the destination's address, the destination's port, the security mode, and the communication mode. On the other hand, a subscriber requires available topics, available publishers, dataset metadata corresponding to topics or publishers, the security mode, and the communication mode. When the data exchange of the system uses a security policy that differs from None, such as PubSub-Aes128-CTR, the publisher and subscriber also need encryption and signature keys. In this case, if the configurator features security key management, a CONFIGURE message will optionally contain the two above keys.

C. OPC UA PubSub-C in the Broker-less Mode

The configurator of an OPC UA PubSub-C system in the broker-less mode has three fundamental features: address space management, registration management, and dataset metadata management. Security key management is an optional feature. Figure 3 shows the exchange between publishers, subscribers, and a configurator.

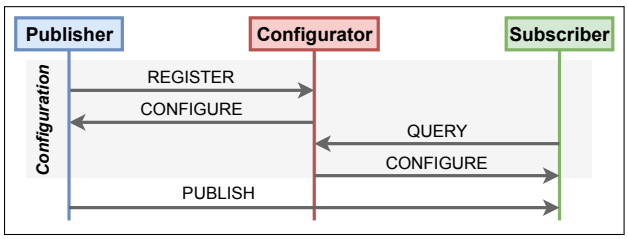

Fig. 3: Exchanges in the broker-less mode

On the one hand, publishers send REGISTER messages to the configurator in the configuration phase. A publisher sends a REGISTER message relevant to each sensor that it possesses. For example, the weather station in Section III-A is equipped with one thermometer and one pluviometer; then, it has two different REGISTER messages. The publisher repeats to send the message after an interval and waits for CONFIGURE messages from the configurator. CONFIGURE messages must contain the information which assigns the communication mode to broker-less. A sensor is in the data exchange phase as soon as having enough configuration information. In this phase, the publisher encodes the sensed data of the sensor into a PUBLISH message and sends the message to destinations. Corresponding to each sensor, the schedule of sending PUBLISH messages is different. For example, the weather station sends PUBLISH messages containing the air temperature every 30 minutes but sends a PUBLISH message containing the rain quantity every 15 minutes.

On the other hand, subscribers send QUERY messages to the configurator and wait for CONFIGURE messages in the configuration phase. CONFIGURE messages must contain the information which assigns the communication mode to broker-less. Each subscriber repeats to send the message until having enough configuration information. It can choose to listen to one or several multicast addresses and wait for PUBLISH messages from publishers.

D. OPC UA PubSub-C in the Broker-based Mode

The configurator of an OPC UA PubSub-C system in the broker-based mode has three fundamental features: address space management, registration management, and dataset metadata management. Security key management is an optional feature. Moreover, in this case, the configurator also functions as a broker. Then, it provides the data forwarding feature and the topic management feature. Figure 4 shows the exchange between publishers, subscribers, and a configurator.

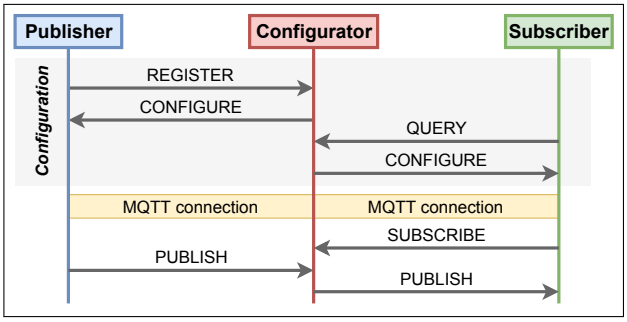

Fig. 4: Exchanges in the broker-based mode

The configuration phase in the broker-based mode has many commons with the one in the broker-less mode presented in Section III-C. The difference is that CONFIGURE messages of the broker-based mode contain the information indicating the communication mode is broker-based.

In the data exchange phase, publishers and subscribers must establish MQTT connections with the configurator. Technically speaking, publishers and subscribers are MQTT clients, the configurator is a broker, and the MQTT connections between them are established and maintained based on four MQTT messages: CONNECT, CONNACK, PINGREQ, and PINGRESP [11]. Next, subscribers send SUBSCRIBE messages to the configurator to subscribe to topics. When a publisher sends a PUBLISH message to the configurator, it classifies the received PUBLISH message and forwards it to subscribers based on the subscribed topics.

IV. Experimentation and Result

The experimental platform comprises three fundamental components: a publisher, a subscriber, and a configurator. The three components connect to a TCP/IP local network. Figure 5 illustrates the infrastructural architecture of the platform. This platform can run in the broker-less or in broker-based mode.

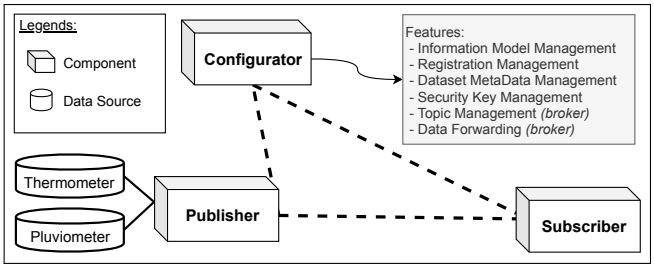

Fig. 5: Infrastructural architecture of the experimental platform

The publisher is equipped with two sources of data, imitating the sensed data from one thermometer and one pluviometer. The thermometer is an air temperature sensor, and the pluviometer is a rain quantity sensor. Each source of data processes a *SensorID*. The application of the publisher is developed on the framework of the Zephyr real-time operating system (RTOS). Zephyr RTOS (https://zephyrproject.org/) is a small-footprint kernel designed for use on a resource-constrained embedded system. An application developed on Zephyr RTOS is called a Zephyr application. While implementing the Zephyr application for the publisher, in addition to the Zephyr core libraries, the Zephyr built-in mbedTLS and MQTT API are also used. The Zephyr built-in mbedTLS provides functions to implement security policy profiles. The Zephyr MQTT provides functions for the MQTT protocol.

The Open62541 (https://open62541.org/) library is used to develop the application of the configurator. This application profits from the advantages of Open62541 in managing address space. It also turns the configurator into an OPC UA server. The configurator in this experiment provides five features. First, it manages the address space that contains all nodes representing publishers' resources corresponding to address space management. Second, it manages the list of configuration information of publishers corresponding to the registration management. Third, it manages the list of dataset metadata corresponding to dataset metadata management. Finally, it runs Eclipse Mosquitto (https://mosquitto.org/), an open-source and lightweight MQTT broker, in a subprocess to turns itself into a broker. As a broker, it provides two more features: topic management and data forwarding features. Finally, it distributes encryption and signature keys to the publisher and subscriber. This experimental platform uses the PubSub-Aes128-CTR security policy; then, the length of the encryption key is 16 bytes, and the length of the signatures key is 32 bytes.

The subscriber's application is developed using Node-Red (https://nodered.org/). Node-Red is a flow-based programming tool that allows building an application by configuring and wiring Node-Red nodes. A Node-Red node is a functional unit of the Node-Red tool that performs one or several predefined tasks. The application developed by Node-Red provides a friendly web browser graphical user interface. The subscriber's application can monitor air temperature and rain quantity value data derived from the publisher and projects them in two graphs. Figure 6 shows a screenshot of the subscriber's application when the experimental platform is running.

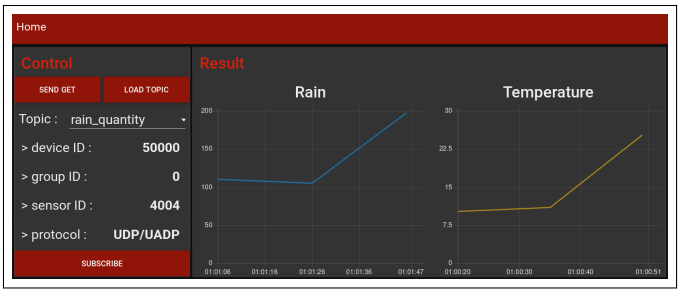

Fig. 6: Screenshot of the subscriber's application

This research uses several tools to evaluate the experimentation. The first tool is a UaExpert (https://unified-automation.com), a full-featured OPC UA Client of Unified Automation GmbH. This tool allows users to access the address space of an OPC UA server. In this project, this tool access the address space located in the configurator. Figure 7 shows the address space and attribute windows on the screen of UaExpert after the configurator receiving a REGISTER message from the publisher. In detail, the configurator adds new objects representing the publisher and its sensors. The configurator creates a new identifier (*NodeId*) for each object. It uses the information provided by the publisher to assigns the name, description, and data type attributes.

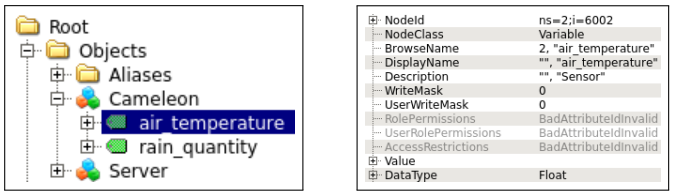

Fig. 7: UaExpert's address space and attribute windows

The second and third tools are the RAM report and ROM report, two tools of the Zephyr project. By using these two tools, users can view the reports of memory consumption of a Zephyr application. In evaluation with different security policy and transport profiles presented in Section III-A, the produced maximal RAM is always smaller than 48 KB, and the produced maximal ROM is always smaller than 96 KB.

V. Related Work

Many research works concern integrating OPC UA for field devices, of which the challenge is still tailoring OPC UA into devices with limited memory size. The first approach is to implement OPC UA nano or micro profiles. These two profiles have fewer OPC UA conformance units than the standard version, such as they include no security mechanism. Imtiaz

et al. claimed the core of their OPC UA server implemented OPC UA nano profile only needs 15 KB of ROM, and a low amount of RAM [12]. Note that the above memory footprint reflects only the functions of OPC UA conformance units. However, the memory footprint of their application, including the information model with the nodes of namespace zero and the nodes of their light sensor process application, is unspecified. Pribiš et al. deployed a demo with embedded devices implemented in the OPC UA micro profile [13]. The information model used in this demo includes the nodes of namespace zero and their custom-specific application nodes. Their devices are equipped with 512 KB of RAM and 2 MB of ROM. Another work of Pfrommer et al. using open62541 library to build an OPC UA server which only needs 100 KB of both RAM and ROM [14]. It declared to use OPC UA PubSub with a minimal set of activated features; however, there is no further detail about the features that this system can provide.

The second approach is to reduce the size of OPC UA implementation by using special techniques to optimize the software or hardware of an embedded device. Veichtbauer et al. propose a software prototype and a technique to load nodes of an information system when they are queried [7]. As result, the memory utilization of their OPC UA server is between 1000 KB and 1900 KB. The implementation of Bauer et al. is an optimal OPC UA hardware engine following the OPC UA nano profile [15]. The hardware engine is integrated into a test chip and requires only 36 KB of memory.

The approach of OPC UA PubSub-C is different from the above ones. It proposes to use a configurator to support other components in an OPC UA PubSub-C system to accomplish their goals. While a configurator occupies the heavy works for memory-constrained sensor devices, such devices can reserve their memory for other works. The idea of OPC UA PubSub-C is inspired by the work of Liu et al. [16]. They use an OPC UA MQTT Configuration Tool to support OPC UA PubSub systems. However, their tool aims only to configure parameters in publishers and subscribers distantly but provides no other OPC UA PubSub features as the configurator in this paper.

VI. Conclusion

To sum up, this paper presents OPC UA PubSub-C, an OPC UA PubSub implementation approach for memory-constrained sensor devices. The difference of OPC UA PubSub-C to other implementations in the market is that it uses a new component called configurator to manage the address space of all sensor devices in the system and to provides other features defined in OPC UA PubSub. OPC UA PubSub-C systems can work both in the broker-less and broker-based modes. In practice, this approach proves that it can fit sensor devices that require less than 48 KB of RAM and 96 KB of ROM.

This research has two points to discuss further. First, the broker-less mode can work in the local network. However, the Intranet or Internet requires routers to have a multicast routing feature. An OPC UA PubSub-C system that uses these routers must configure them to recognize the multicast addresses available in the system. The broker-based mode can avoid the above inconvenience since the broker can connect to the intranet and internet to forward data. However, the broker itself is a weak point: when it is disabled by accident or by a cyberattack, the whole system is down.

Second, the messages REGISTER, QUERY, and CONFIGURE in this paper are only prototypical. In practice, they should be in a widely accepted format, such as JSON. As if it happens, the REGISTER message in JSON format should be simple, since a complex message involves a complex data parsing process; that also means more memory consumption.

Acknowledgment

This research is funded by SNCF, the France's national railway company.

References

[1] Y. Lu, "Industry 4.0: A survey on technologies, applications and open research issues," *Journal of Industrial Information Integration*, p. 10, June 2017.

[2] 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 *2019 IEEE International Conference on Industrial Technology (ICIT)*. Melbourne, VIC, Australia: IEEE, February 2019, pp. 955–962.

[3] OPC Foundation, "OPC Unified Architecture - Part 3: Address Space Model," Industry Standard Specification OPC 10000-3, 2017.

[4] OPC Foundation, "OPC Unified Architecture - Part 5: Information Model," Industry Standard Specification OPC 10000-5, 2017.

[5] OPC Foundation, "OPC Unified Architecture - Part 7: Profiles," Industry Standard Specification OPC 10000-7, November 2017.

[6] OPC Foundation, "OPC Unified Architecture - Part 14: PubSub," Industry Standard Specification OPC 10000-14, February 2018.

[7] A. Veichtlbauer, M. Ortmayer, and T. Heistracher, "OPC UA integration for field devices," in *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*, July 2017, pp. 419–424.

[8] D. Happ, N. Karowski, T. Menzel, V. Handziski, and A. Wolisz, "Meeting IoT platform requirements with open pub/sub solutions," *Annals of Telecommunications*, vol. 72, no. 1-2, pp. 41–52, February 2017.

[9] V. Romanov, I. Galelyuka, and Y. Sarakhan, "Wireless sensor networks in agriculture," in *2015 IEEE Seventh International Conference on Intelligent Computing and Information Systems (ICICIS)*. Cairo, Egypt: IEEE, December 2015, pp. 77–80.

[10] N. Q. Uy and V. H. Nam, "A comparison of AMQP and MQTT protocols for Internet of Things," in *2019 6th NAFOSTED Conference on Information and Computer Science (NICS)*. Hanoi, Vietnam: IEEE, December 2019, pp. 292–297.

[11] OASIS, "MQTT Version 3.1.1," OASIS Open, OASIS Standard, September 2014.

[12] J. Imtiaz and J. Jasperneite, "Scalability of OPC-UA down to the chip level enables Internet of Things," in *2013 11th IEEE International Conference on Industrial Informatics (INDIN)*. Bochum, Germany: IEEE, July 2013, pp. 500–505.

[13] R. Pribiš, L. Beňo, and P. Drahoš, "Implementation of Micro embedded OPC Unified Architecture server-client," *IFAC-PapersOnLine*, vol. 52, no. 27, pp. 114–120, 2019.

[14] 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)*. Turin: IEEE, September 2018, pp. 1087–1090.

[15] H. Bauer, S. Höppner, C. Iatrou, Z. Charania, S. Hartmann, S.-U. Rehman, A. Dixius, G. Ellguth, D. Walter, J. Uhlig, F. Neumärker, M. Berthel, M. Stolba, F. Kelber, L. Urbas, and C. Mayr, "Hardware implementation of an opc ua server for industrial field devices," *ArXiv*, vol. abs/2105.00789, 2021.

[16] Z. Liu and P. Bellot, "OPC UA PubSub Implementation and Configuration," in *2019 6th International Conference on Systems and Informatics (ICSAI)*. Shanghai, China: IEEE, November 2019, pp. 1063–1068.