Author Manuscript of the publication in
Proceedings of the 2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation

Open Source OPC UA PubSub over TSN for Realtime Industrial Communication

Julius Pfrommer*, Andreas Ebner*, Siddharth Ravikumar† and Bhagath Karunakaran†

*Fraunhofer IOSB, Fraunhoferstraße 1, 76131 Karlsruhe, Germany
julius.pfrommer@iosb.fraunhofer.de, ebner.andreas@outlook.com

†Kalycito Infotech, 6/2 Pappampatti Pirivu Trichy Road, 641103 Kannampalayam Coimbatore, India
siddharth.r@kalycito.com, bhagath@kalycito.com

***Abstract*—OPC UA is a client-server communication protocol for industrial use cases without hard realtime requirements. The new PubSub extension of OPC UA adds the possibility of many-to-many communication based on the Publish / Subscribe paradigm. In conjunction with the upcoming Time-Sensitive Networking (TSN) extensions of Ethernet, OPC UA PubSub aims to also cover time-deterministic connectivity. This poses requirements to OPC UA implementations that have traditionally not been regarded. We propose an approach to combine non-realtime OPC UA servers with realtime OPC UA PubSub where both can access a shared information model without the loss of realtime guarantees for the publisher. As a result, the publisher can be run inside a (hardware-triggered) interrupt to ensure short delays and small jitter. An open source implementation of OPC UA PubSub is provided based on the open62541 SDK. This is also the basis for measurements used to evaluate the potential of the technology.**

I. Introduction

OPC UA is widely used for flexible best-effort communication in automation. For many use cases, OPC UA has replaced its predecessor, the Microsoft Windows-centric OPC Classic, as well as many vendor-specific protocols. The hope associated with OPC UA is to achieve uniform standardized and secure communication across vendors, application domains and levels of the automation hierarchy. The current usage practice and the emergence of OPC UA companion specifications for many application domains support this claim.

The new part 14 of the OPC UA specification defines an extension of OPC UA based on the Publish / Subscribe communication paradigm [1]. This opens up new usage scenarios, including many-to-many communication. In addition, the integration of OPC UA PubSub with Time-Senstive Networking (TSN) is designated to additionally enable realtime communication. However, there exists currently a gap between the commonly available OPC UA SDKs and the realtime requirements of field-level automation equipment. This paper attempts to close this gap.

This section continues with some background of the OPC UA standard and Time-Sensitive Networking (TSN). Section II proposes ways to couple non-realtime OPC UA servers with a realtime OPC UA PubSub publisher. Our implementation of this approach based on the open62541 SDK is discussed in Section III. This is the basis for the evaluation in Section IV. Section V concludes with a summary and future outlook.

A. OPC Unified Architecture

OPC UA is a client-server protocol for industrial communication based on TCP/IP. It has been standardized as IEC 62541. An OPC UA server provides access to data and functionality that is structured in an object-oriented information model. Clients interact with the information model via a set of standardized services. Each of the service defines a request and a response message for the interaction. OPC UA strictly adheres to the Request / Response communication pattern. However, since OPC UA is an asynchronous protocol, a subscription mechanism can be used for push-transmission of notifications only when they occur.

Similar to its predecessor OPC Classic, OPC UA has become a major contender for flexible vendor-neutral communication in industrial applications. The OPC Foundation drives the continuous improvement of the standard, the development of companion specifications to standardize information models for the different application domains. Ot also provides the infrastructure for compliance certification in order to ensure the interoperability of implementations.

B. Time-Sensitive Networking (TSN)

Within the IEEE 802.1 standards series, enhancements of Ethernet for realtime communication have been developed first under the name of Audio Video Bridging (AVB) and in recent years as Time-Sensitive Networking (TSN). Some standards that are part of TSN have already been adopted, such as clock synchronization of the participants in IEEE 802.1AS [2] and the reservation of transfer capacity via time slots in IEEE 802.1Qbv. See Table I for an overview on the standards in the TSN standards series. The authors of [3] describe traffic types for common use cases in industrial communication and their relation to the TSN standards. See also [4] for the computation of the potential performance of Ethernet-based realtime protocols.

The advantage of TSN compared to classical fieldbuses is the vendor neutrality without licensing fees, potentially higher

TABLE I
TSN ENHANCEMENTS IN THE IEEE 802.1 SERIES

Standard	Title
IEEE 802.1Qav	Forwarding and Queuing Enhancements for Time-Sensitive Streams
IEEE 802.1AS-Rev	Timing and Synchronization for Time-Sensitive Applications
IEEE 802.1Qbu	Frame preemption
IEEE 802.1Qbv	Enhancements for Scheduled Traffic
IEEE 802.1Qca	Path Control and Reservation
IEEE 802.1Qcc	Stream Reservation Protocol (SRP) Enhancements and Performance Improvements
IEEE 802.1Qci	Per-Stream Filtering and Policing
IEEE 802.1Qcr	Bridges and Bridged Networks Amendment: Asynchronous Traffic Shaping
IEEE 802.1CB	Frame Replication & Elimination for Reliability

throughput [5], the possibility to flexibly add connections with assured quality-of-service properties over multiple hops in a bridged network [6] and the economies of scale of a widespread technology that is intended for both industrial use-cases and consumer devices.

C. OPC UA PubSub and OPC UA PubSub over TSN

The new Part 14 of the OPC UA specification [7] extends OPC UA with Publish / Subscribe. Publish / Subscribe lets many subscribers register for a topic. Published messages are forwarded to all subscribers of the message's topic. Given that potentially many subscribers receive the same message, in some sense OPC UA PubSub returns to the definition of "telegram formats" similar to traditional fieldbuses. But the message layout can be configured at runtime. The content of the published messages is defined by a so-called *PublishedDataSet*, which represents a collection of variables and event sources from the information model of an OPC UA server. The PublishedDataSet can be flexibly configured and its definition can be looked up in the server to understand the semantic meaning of the published payload.

First, the standard defines the integration of existing Publish / Subscribe protocols, specifically MQTT and AMQP. These protocols define a central broker for the message distribution and are commonly used in the public Internet. Second, the standard defines a custom UDP-based distribution protocol, called UADP, based on the multicast mechanisms of the IP standard. Here, the subscriber registers for a multicast group represented by an IP address in a special range. Packets sent to this address are forwarded to all members of the group. This delegates a large part of the publisher complexity to the existing network infrastructure (router, switches, and so on). However, the availability of multicast is generally limited to local area networks. Lastly, the standard also defines transport of PubSub messages directly on the data link layer. Being already at the Ethernet level, OPC UA PubSub can be readily integrated with TSN for realtime transport. For this, the Ethernet EtherType 0xB62C has been registered at the IEEE specifically for the use with OPC UA PubSub over TSN.[1]

It has to be noted that the origin of an OPC UA PubSub message in the server's information model is no longer visible once the message has been serialized into a network frame. This opens the possibility of very lightweight OPC UA PubSub implementations that assume a fixed PublishedDataSet and directly generate the desired network frame without the overhead of a full OPC UA server. This of course comes at the loss of the possibility to lookup the origin of values and hence their semantic interpretation.

II. MIXING NON-REALTIME OPC UA SERVERS WITH REALTIME OPC UA PUBSUB

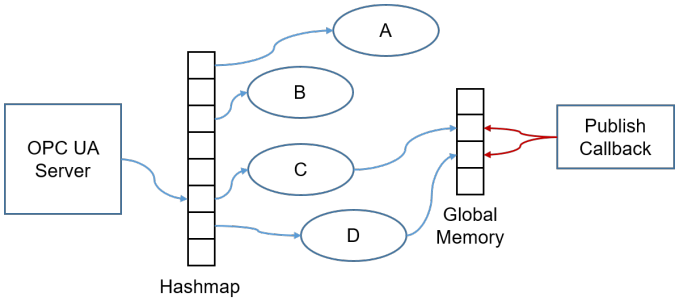
The IEEE 802.1Qbv standard defines transmission queues for different traffic classes, each controlled by a transmission gate. The transmission gates are cyclically triggered. The network devices are clock-synchronized (via IEEE 802.1AS-Rev) and the time slots for the transmission queues (windows) are configured. If a packet does not complete transmission within its designated time slot, then it must wait until the next cycle. Since the realtime guarantees of interest are end-to-end (from the sending application to the receiving application) and not only between network devices, the preparation of the network message needs to happen within a defined delay before the opening of the transmission window.

We have to assume that TCP/IP based OPC UA servers can not achieve hard realtime guarantees. But now they are intended as the source of the values for OPC UA PubSub messages. On a single-core system, the normal OPC UA server's operations could be preempted with a cyclic (hardware-triggered) interrupt to execute the publisher in time. Multi-threaded programs can use locks to protect race-conditions during access to shared memory. But locking is not possible to protect shared memory access for single-threaded programming with interrupts. When the thread holding the lock is interrupted and the interrupt requests the lock, it will never be released and the program execution halts in a deadlock. Instead, all functions called from an interrupt have to be *reentrant*. For example, the standard POSIX `malloc` for dynamic memory allocation is not reentrant and cannot be used inside the publisher [8]. Even and more importantly, the access to the OPC UA information model, with shared access from the normal OPC UA server, needs to be reentrant as well.

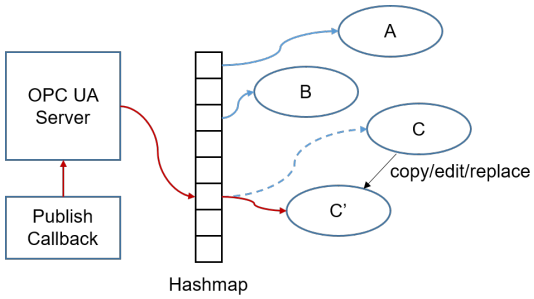
The first possibility for this is the use of well-known shared memory locations for scalar values that are replaced in place with atomic operations.[2] The use of atomic operations ensures that updates cannot be interrupted, which could leave an inconsistent value behind. But only small scalars (such as integers and floating point numerals) are amenable to atomic updates. The approach is illustrated in Figure 1a. It allows for applications with extreme latency requirements, such as

[1] See the registry entry at http://standards-oui.ieee.org/ethertype/eth.txt.

[2] In the open62541 SDK, this is implemented with a DataSource callback for specific variables, forwarding read and write requests to user-defined methods.



(a) Global shared memory with atomic operations.



(b) Copy-on-replace for immutable nodes. The node C is copied to C', edited and finally replaced with an atomic operation.

Fig. 1. Possibilities for accessing the information model of an OPC UA server from an interrupt. Red arrows represent access from the interrupt that needs to be reentrant.

Fig. 2. OPC UA PubSub over TSN demonstrator at the booth of the OPC Foundation at Hannover Fair 2018. The following hardware platforms are integrated into the demonstrator: Intel Atom/I210 by TQ-Systems, Dual chip solution with ARM Cortex M4 + TSN switch from Analog Devices, FPGA-TSN Cyclone V SoC by Intel PSG/Altera, FPGA-TSN Zync SoC by Xilinx. The components are daisy-chained via integrated 2-port switches. The operating systems used are Linux with RT-Preempt patches [9] and FreeRTOS.

motion control, since the values for the publisher are directly accessed and not queried from the information model.

The second possibility is the use of copy-on-replace for updates to the information model: The information model is represented as a graph of nodes and typed references between them. The underlying data structure for the information model in the open62541 SDK is a hash-map from the node identifier to the node representation in memory. Each node is treated as immutable and cannot be modified once it has been inserted into the hash-map. It is only possible to replace the entire node with a modified copy. This replacement uses an atomic compare-and-switch (CAS) operation. So the information is always consistent, even if the server is interrupted mid-update. This approach is shown in Figure 1b. The advantage of the second approach is that any kind of value (also strings) can be used for PubSub. Furthermore, the PubSub configuration, also stored in the information model, can be accessed by the publisher in every cycle without breaking realtime guarantees. In addition to providing reentrant read-access to the information model from an interrupt, copy-on-replace also enables lock-free multi-threading for normal operations of the OPC UA server.

III. Implementation Details

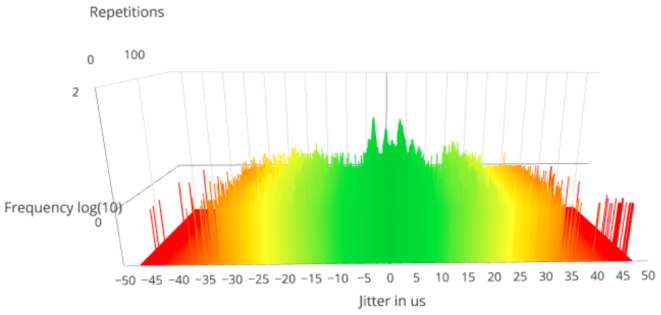
Fraunhofer IOSB has launched an open source implementation and evaluation of OPC UA PubSub over TSN together with Kalycito Infotech, an engineering solutions provider for embedded and realtime applications. The Open Source in Automation Development Labs (OSADL) eG provided the organizational framework for a consortium of companies from industry jointly funding the effort. Besides the open source implementation, the project has resulted in a demonstrator (see Figure 2) and training events for the funding companies.

The implementation is based on the open62541 SDK (https://open62541.org) [10]. It is written in C99 an open source under the MPLv2 license. A hardware-abstraction layer helps to keep the core library free from platform-specific interfaces (also POSIX) so that porting to different hardware architectures and operating systems is straightforward. OPC UA servers based on the open62541 SDK require less than 100kB of both ROM and RAM for a minimal set of activated features.

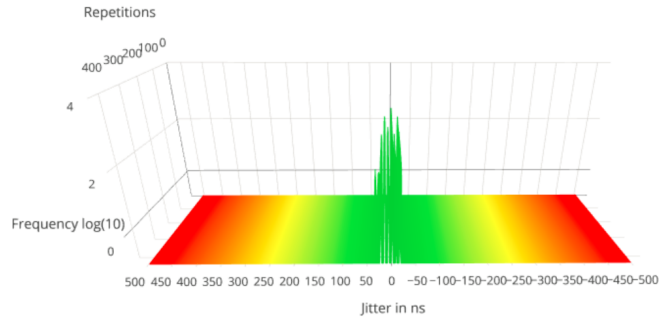
The focus of the OPC UA PubSub implementation is on the binary message format and brokerless transport. The integration of OPC UA PubSub uses three of the internal interfaces of the SDK. First, the en- and decoding routines for the binary protocol are reused to generate the payload of the published NetworkMessage. Second, the OPC UA information model holds the runtime values that are published. Third, the standard OPC UA services are used to change the PubSub configuration at runtime. This is possible as the PubSub configuration is also represented in the information model.

As described in Section II, the OPC UA PubSub publisher is called cyclically from a time-triggered interrupt. The interrupt-based publisher internally performs the following steps:

1) Replace the default `malloc` with a simple buffer-based implementation for the duration of the interrupt
2) Generate and populate the NetworkMessage data struc-



(a) Observed jitter without TBS.



(b) Observed jitter with TBS enabled.

Fig. 3. Jitter of the received OPC UA PubSub packets. Each line on the z-axis (Repetitions) indicates a set of measurements with 400 samples each.

ture from the definition of the PublishedDataSet
3) Encode the NetworkMessage into a binary message buffer
4) Send (enqueue in hardware queue) the binary message buffer in the TSN-capable network interface
5) Reset the buffer-based `malloc` and switch back to the default implementation

IV. Evaluation

The following measurements were performed on two identical PCs with an Intel Core i5-6402P processer running at 2.80GHz and Intel i210 network interface cards connected via PCIe. The operating system is Linux 4.16.8-rt3 with RT-Preempt patches. The PCs are connected peer to peer with an Ethernet cable at a link speed of 1Gbps. The system clocks of the two PCs are synchronized using IEEE 802.1AS-Rev. A patch-set provided by Intel was used for the 802.1Qbv functionality based on the SO_TXTIME socket option and time-based scheduling (TBS).

The OPC UA PubSub traffic is configured at a 100µs cycle time (10kHz). The transmitted OPC UA PubSub NetworkMessage is based on a PublishedDataSet with a single integer value. The PublishedDataSet configuration is read at every cycle and the message is generated based on an up-to-date value read from the information model. The configured offset of 5µs gives enough time for the application to prepare the next packet and transfer it to the underlying layers, so that the packet is enqueued in time for its transmission window.

Figure 3 plots the observed jitter of OPC UA PubSub with time-based scheduling and without. From Figure 3a it is evident that the application publishes the packets without precise timings when TBS is not configured. In contrast, Figure 3b shows the timely transmission of the publish packets with nanosecond jitter when TBS is enabled. This graph shows about 40ns jitter of the received packets.

V. Conclusion

This paper has investigated how the Time-Sensitive Networking (TSN) extension of the Ethernet standards can be used for the transport of OPC UA PubSub messages in practice. We have proposed an approach where the message for the publisher is prepared in a (hardware-triggered) interrupt. Specific modifications are required to allow the interaction between a best-effort standard OPC UA server and a realtime OPC UA PubSub publisher with access to a shared information model. The approach was implemented based on the open62541 OPC UA SDK. Latency measurements based on the Linux RT-Preempt patches show that sub-millisecond publication intervals can be achieved with minimal jitter.

Future plans with regards to the use of TSN for realtime OPC UA PubSub include porting to additional TSN implementations and embedded platform, long-term tests in a QA farm and the creation of OPC UA PubSub over TSN distributions – spanning hardware (TSN IP), operating system, and the OPC UA SDK, with tested end-to-end latency and jitter properties. Additional features for the implementation are planned also for non-realtime use cases of OPC UA PubSub. For example the addition of JSON-based message encoding, broker-based message distribution, using the MQTT and AMQP protocols, and message-encryption.

References

[1] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM computing surveys (CSUR)*, vol. 35, no. 2, pp. 114–131, 2003.

[2] S. Schriegel and J. Jasperneite, "Investigation of industrial environmental influences on clock sources and their effect on the synchronization accuracy of ieee 1588," in *Precision Clock Synchronization for Measurement, Control and Communication, 2007. ISPCS 2007. IEEE International Symposium on*, IEEE, 2007, pp. 50–55.

[3] Industrial Internet Consortium, "Time Sensitive Networks for Flexible Manufacturing Testbed - Description of Converged Traffic Types," Tech. Rep., 2018.

[4] J. Jasperneite, M. Schumacher, and K. Weber, "Limits of increasing the performance of industrial ethernet protocols," in *Emerging Technologies and Factory Automation, 2007. ETFA. IEEE Conference on*, IEEE, 2007, pp. 17–24.

[5] Shaper Group, "OPC UA TSN: A new Solution for Industrial Communication," Tech. Rep., 2018.

[6] M. Gutiérrez, A. Ademaj, W. Steiner, R. Dobrin, and S. Punnekkat, "Self-configuration of ieee 802.1 tsn networks," in *Emerging Technologies and Factory Automation (ETFA), 2017 22nd IEEE International Conference on*, IEEE, 2017, pp. 1–8.

[7] OPC Foundation, *OPC Unified Architecture Specification, Part 14: PubSub*, 2018.

[8] D. R. Butenhof, *Programming with POSIX threads*. Addison-Wesley Professional, 1997.

[9] F. Cerqueira and B. Brandenburg, "A comparison of scheduling latency in linux, preempt-rt, and litmus rt," in *9th Annual Workshop on Operating Systems Platforms for Embedded Real-Time Applications*, SYSGO AG, 2013, pp. 19–29.

[10] F. Palm, S. Grüner, J. Pfrommer, M. Graube, and L. Urbas, "Open source as enabler for opc ua in industrial automation," in *Emerging Technologies & Factory Automation (ETFA), 2015 IEEE 20th Conference on*, IEEE, 2015, pp. 1–6.