





Static Timing Analysis of OPC UA PubSub

Patrick Denzler 
Institute of Computer Engineering
TU Wien
Vienna, Austria
patrick.denzler@tuwien.ac.at

Thomas Frühwirth 
Research Department
Austrian Center for Digital Production
Vienna, Austria
thomas.fruehwirth@acdp.at

Andreas Kirchberger
Institute of Computer Engineering
TU Wien
Vienna, Austria
a.kirchberger@kbit.pro

Martin Schoeberl 
Department of Applied Mathematics and Computer Science
Technical University of Denmark
Lyngby, Denmark
masca@dtu.dk

Wolfgang Kastner 
Institute of Computer Engineering
TU Wien
Vienna, Austria
wolfgang.kastner@tuwien.ac.at

Abstract—Industrial automation is changing towards higher integration and seamless communication. A stepping stone is end-to-end real-time machine-to-machine communication, now becoming feasible with technologies such as time-sensitive networking (TSN) and OPC Unified Architecture (OPC UA) publish-subscribe. While TSN takes care of communication, the OPC UA stack’s execution time behavior remains unknown. This paper highlights experiences made while adjusting the OPC UA subscriber of the open62541 stack for worst-case execution time (WCET) analysis. Two directly connected time-predictable T-CREST platforms hosting the publisher and subscriber delivered end-to-end timing measures validating the WCET estimates. The paper concludes by outlining further research with several time-predictable publishers and subscribers.

Index Terms—worst-case execution time, industrial software, real-time communication, OPC UA

I. INTRODUCTION

Modern factories are complex entities built upon software and hardware components from the domains of information technology (IT) and operational technology (OT). IT and OT in industrial automation form a pyramid alike architecture referred to as the automation pyramid (cf. Figure 1) [1].

The lower two OT levels of the pyramid consist of the field layer with its sensors and actuators and the control layer with programmable logic controllers (PLCs) often coupled by industrial communication systems, such as EtherCat or Profibus [2]. For specific control loops, OT must fulfill real-time requirements to ensure timely communication and processing to meet fixed deadlines. The Supervisory Control and Data Acquisition (SCADA) systems and human-machine interfaces on the third layer observe and handle the lower layers below [3]. The upper two IT levels establish the connection to the enterprise systems represented with Manufacturing

This work has been partially supported and funded by the Austrian Research Promotion Agency (FFG) via the “Austrian Competence Center for Digital Production” (CDP) under the contract number 881843. Moreover, the research leading to these results has received funding from the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 764785, FORA—Fog Computing for Robotics and Industrial Automation.

978-1-6654-2478-3/21/\$31.00 ©2021 IEEE

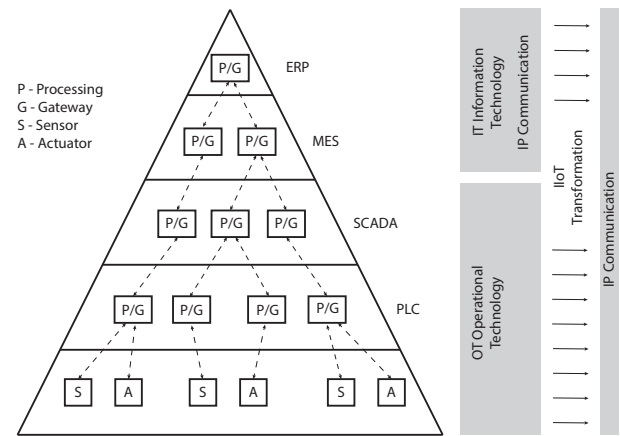


Fig. 1. Automation pyramid transformation towards Industrial Internet [3]

Execution Systems (MESs) and Enterprise-Resource-Planning (ERP) applications built upon commercial off-the-shelf components such as servers, desktop PCs communicating via the Internet Protocol (IP). Due to the technological differences, a gap occurs between OT and IT.

Currently, industrial automation is changing towards higher integration and seamless communication. The Industrial Internet envisions an IP-based architecture that vertically and horizontally integrates all devices in a factory from the shop floor to the cloud. Technological advances such as time-sensitive networking (TSN) address OT real-time requirements and fog computing provides computing resources necessary to close the OT/IT gap [4]. For machine-to-machine communication, OPC Unified Architecture (OPC UA), a middleware with extensive information modeling capabilities, enables accessing industrial equipment and systems. The combination of TSN and the OPC UA extension for publish-subscribe, further pushes towards an end-to-end real-time machine-to-machine communication [5].

A remaining challenge on the way to end-to-end real-time communication is the timing behavior of the OPC UA stack. While TSN ensures the timely transport of data packages, OPC UA PubSub does not provide any timing guarantees for processing the packages. Other domains (e.g., avionics or automotive engineering) with strict timing requirements address such an issue with program timing analysis [6]. A well-known timing measure to determine a program’s execution time characteristics is the worst-case execution time (WCET) of a program [7]. Available tools for determining the WCET are measurement-based or perform static program analysis.

This paper addresses the challenges of applying static WCET analysis on the OPC UA PubSub part as the first step towards a distributed real-time end-to-end data transfer environment suitable for the Industrial Internet. The open-source OPC UA stack open62541 provided the foundation for the WCET analysis, and the time-predictable platform T-CREST [8] the WCET tools and the Patmos processor [9].

On the one hand, the contributions are an OPC UA stack adapted for WCET analysis with necessary changes at the source code for specifying WCET values and indications to determine RT-capabilities. The adjusted source code is available in a Git repository [10]. On the other hand, a measurement setup with two time-predictable platforms is introduced to evaluate the WCET values and timing results for a complete end-to-end data transfer. The obtained WCET values form a baseline for comparison with measurements collected on off-the-shelf processors not purposely built for WCET analysis. Moreover, the findings enable research on the inclusion of TSN as a means of communication between platforms.

This paper contains seven sections: the following two sections present background and related work relevant to the topic. Section IV introduces the WCET adjustment process and gained insights applying it to the publisher and subscriber. The evaluation Section V addressed the setup and obtained results. Section VI discusses the findings, and Section VII concludes the article.

II. BACKGROUND

The primary aim of timing analysis is to characterize the timing behavior of programs or systems and provide guarantees on upper timing bounds [7]. Widely used time bounds are WCET- and best-case execution times (BCETs). WCET and BCET refer to the longest and shortest execution time of a program or task, respectively (cf. Figure 2). Worst case guarantees for programs or systems are larger than the WCET value. There are two main methods, static analysis and measurement-based, to obtain timing bounds [7].

A. WCET Analysis and Tools

In industry, a frequently used method for program timing analysis is by measurements [11]. Numerous test-runs measure a program’s execution time while varying the input parameters (cf. Figure 2). This kind of method provides an average timing behavior or an approximate WCET value. However, there is a limitation of this type of analysis. Each run only

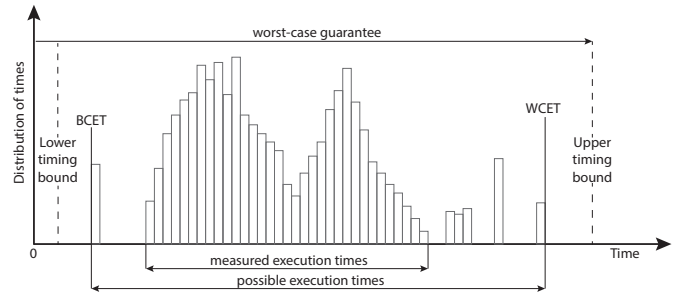


Fig. 2. Basic notions of timing analysis [7].

follows one program path, i.e., if there are too many execution paths, this method is not suitable because the measurements underestimate the WCET. This method requires adding safety margins to ensure that the WCET value is not too low. These margins carry the risk of over- or under-provisioning of computing resources or cause schedulability issues. Measurement equipment includes oscilloscopes, logic analyzers, and in-circuit emulators on the hardware to measure execution times.

The static WCET analysis technique is more suitable for programs with multiple execution paths and stricter timing boundaries. This type of method does not execute the program but statically analyses the timing properties [12]. Static WCET tools tend to give larger WCET estimates (upper bounds) than the actual execution time without the need for additional margins. Simplified, a WCET analysis consists of three parts: a flow analysis to distinguish the possible program execution paths, a low-level analysis to approximate times for atomic parts of the code (e.g., instructions, basic code blocks), and the calculation part to combine the two previous phases into a WCET approximation.

Flow analysis is about defining maximum loop bounds [13] because the number of loop iterations affects the WCET estimates. Advanced tools include methods to determine loop bounds automatically; however, the manual annotation of loop bounds is still required in most cases. Another characteristic of the flow analysis is the possibility of recognizing infeasible paths, i.e., paths that are feasible in the control-flow graph (CFG), but inaccessible when studying the input data values and the semantics of the program [13].

The low-level analysis considers that modern hardware features pipelines, caches, and out-of-order execution influence the timing behavior of the program [8]. A technique to deal with these issues is using models, e.g., simulators, to enable the analysis without the actual hardware. Nevertheless, creating accurate models of a processor and hardware is complicated and sometimes introduces additional complexity into the analysis. Safe and straightforward processor models, on the other hand, lead to higher WCET bounds. The combination of flow and low-level analysis allows the calculation of the WCET. For more detailed information, readers are encouraged to consider reading the detailed survey about available methods and technological advances in determining timing guarantees by Wilhelm et al. [7].

With the increasing complexity of current software and hardware, there is a broad spectrum of research activities. The topics span from integer linear programming [14], model checking [15], and tree-based calculation, [13]. Other researchers investigate code conversion to WCET-analyzable single-path code [16] or specialized programming languages [17]. On the tooling side, there are a few commercial products available such as *aiT* [18] from AbsInt or *RapiTime* [19] from Rapita Systems, and academic open-source prototypes, such as T-CREST [9] or SWEET [13].

B. OPC Unified Architecture

OPC UA is the successor of the open platform communications (OPC) protocol and regarded as one pathfinder to homogenize communication in the industrial domain. A typical view of the OPC UA architecture is one of two pillars [20]. The first represents the Meta Model that enables information modeling. In contrast, the second pillar describes the Transport Mechanisms responsible for encoding data and exchanging messages between devices. OPC UA's primary communication paradigm is Client-Server but it also offers support for Publish-Subscribe (OPC UA PubSub) communication patterns. The Client-Server mechanism can invoke complex services like browsing the information model and calling methods. The newer OPC UA PubSub part minimizes the communication overhead and is primarily intended for exchanging process data.

There are several OPC UA software stacks available in a variety of programming languages. A widely used and well-maintained open-source stack is the open62541 project. The open62541 supports most of the OPC UA features, implements the OPC UA PubSub specification, and allows easy porting to other hardware platforms. The stack is implemented in C, which is well-supported by the T-CREST project and other WCET analysis tools.

III. RELATED WORK

The range of publications concerned with timing analysis is considerably extensive; however, concrete studies on analyzing industrial software are quite rare. As the OPC UA PubSub standard is relatively new, there are no studies available that approach WCET analysis. Therefore, the related work presents research closely linked.

The authors in [21] propose a concept of real-time capable and long-running tasks in OPC UA. They use OPC UA programs and combine it with real-time communication over TSN to achieve a distributed and synchronized message exchange for industrial applications. Similarly in [22], the authors investigate the usability of TSN for the transport of OPC UA PubSub messages in practice. They propose an open62541 adjusted publisher triggered by a hardware interrupt that accesses a shared information model to achieve real-time properties. Other research focuses on TSN enabled field devices with OPC UA running on commercial off-the-shelf (COTS) hardware and software [23], or explore the pos-

sibilities of configuring TSN and OPC UA as an application layer protocol [24].

In [25], the authors interwove TSN, OPC UA, and software-defined networking to provide a solution for future industrial networks, especially for dynamically interconnected devices and applications with time-sensitive requirements. The always-changing interconnection between devices and applications requires a dynamic configuration. The authors in Panda et al. [26] simulated OPC UA field level communication by using OPC UA PubSub, best-effort IP-based communication between OPC UA servers, and a shared information model. In almost all current research related to OPC UA PubSub the open-source OPC UA project open62541 was chosen.

Timing analysis on industrial software reaches from space applications [27] to the avionics industry [6]. Relevant for the industrial communication context, the authors in [28] present a case study to find upper time bounds for time-critical industrial code with static WCET analysis. This case study focused on identifying practical difficulties when applying current WCET analysis methods and how labor-intensive the analysis becomes. Gustafson et al. [29] compiled their results from five different industrial case studies using both static and measurement-based tools with a similar approach. Of particular interest was if current timing analysis methods are suitable for industrial code. The authors achieved very high accuracy but also indicated a high complexity due to the required annotation work. In [30], the authors used automatic flow analysis on industrial real-time system code to reduce manual work. Most case studies analyze binary code, which makes the annotation more difficult. Lisper et al. [31], proposed to address the annotation issue by executing the program flow analysis on the source level and resolved some program flow constraints on the binary level.

Other authors focused on general WCET challenges. For example, Sehlberg et al. [32] found that industries in this field oppose complexities with strict guidelines that exclude language constructs that make programs not analyzable for WCET. Especially structures such as pointers, recursive data structures, dynamic memory allocation, assignments with side effects, recursive functions, and variable-length loops are known to be an obstacle for WCET analysis [33]. Modern WCET tools can handle some of those constructs; however, reprogramming is required in some cases. The accuracy of the required WCET bounds determines the amount of work to be completed [34].

IV. WCET ANALYSIS OF OPC UA PUBSUB

The main objective of this paper is to enable real-time communication via OPC UA PubSub. Therefore, this section first examines the timings involved in end-to-end data transmission. Next, it presents a process for preparing existing software for WCET analysis. This process is then applied to the OPC UA publisher and subscriber implementations of the open62541 open-source stack and the tools provided by the T-CREST project are used to determine their WCETs.

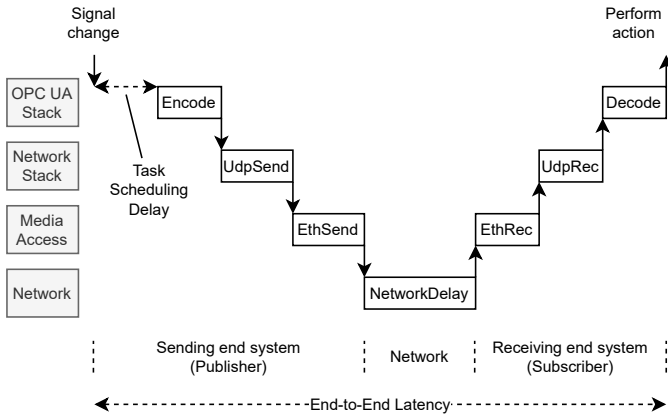


Fig. 3. OPC UA PubSub end-to-end latency diagram

A. End-to-end latency

The exchange of an input signal from one device to another via OPC UA PubSub involves numerous delays at the sending end system (i.e., the publisher), the receiving end system (i.e., the subscriber), and the network connection. These delays are illustrated in Figure 3 and have to be analyzed to obtain a guaranteed upper limit for the end-to-end latency of transmissions.

The analysis starts at the publisher with the change of a signal representing some operational data relevant for another device. In most real-time systems, software tasks are executed at predefined points in time rather than triggered by external events. This concept avoids unpredictable behavior in high-load scenarios, e.g., if the input signal changes very rapidly. However, it introduces the *TaskSchedulingDelay*, which is the time elapsing between a change of the signal and the subsequent execution of the publisher task. The *TaskSchedulingDelay* is limited by the task's period, i.e., the time between two consecutive activations. It can be reduced close to zero if the signal represents some internal data value or if changes to the input signal can be synchronized to the publisher task.

Next, during *Encode*, the OPC UA publisher encodes the signal in a message, following the structure defined in Part 6 and Part 14 [35] of the specification. The message is then handed from the OPC UA stack to the network stack, which adds additional information like the UDP, IP, and Ethernet datagram headers (*UdpSend*). All operations performed during *Encode* and *UdpSend* are implemented in software. Therefore, upper limits for their execution times can be determined by applying WCET analysis.

The Ethernet frame is then processed and sent over the network by the Ethernet controller, typically implemented in hardware. Therefore, the delay introduced in this step (*EthSend*) is predictable, as long as the Ethernet controller is not occupied with the processing of other messages. This behavior has already been analyzed in other studies [36]. The next timing to be considered is the *NetworkDelay*. If the message shall be transmitted with bounded delays, a real-time Ethernet protocol (e.g., TSN) is mandatory.

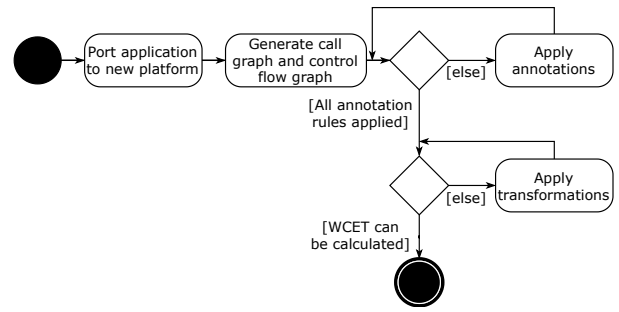


Fig. 4. Process for adjusting existing software for WCET analysis [37]

The delays occurring at the subscriber are analogous to the ones introduced at the publisher. They are caused by the *EthRec*, *UdpRec*, and *Decode* functions, whereby only the latter two are implemented in software and need to be analyzed for WCET.

B. WCET Analysis Process

Although existing WCET analysis tools like Absint *aiT* or the *T-CREST* platform offer significant support, determining the WCET cannot be fully automated. In practice, static WCET analysis of existing code that has not been written for real-time applications often requires additional manual work to calculate the WCET. Furthermore, finding reasonable tight bounds to make the code applicable in real-time applications may require considerable effort.

Figure 4 illustrates the essential steps to prepare existing program code for WCET analysis in a simple process derived from modern WCET analysis tools. The process is intended to serve as an abstract guide and a helpful starting point for static WCET analysis. It consists of four main steps, which are presented in detail in [37]: (A) porting the existing code to a time-predictable platform, (B) examining the code structure via the call graph and the control-flow graph (CFG), (C) applying code annotations, and (D) code transformations.

Most importantly, the process steps cover annotation rules to define upper bounds and enable WCET analysis of while loops, do-while loops, for loops, and direct recursions. Furthermore, code transformation rules suggest how to adjust indirect recursions, jump tables, callback functions, and other non-analyzable code for WCET analysis. Applying this process to an OPC UA publisher and subscriber allows determining the WCET for sending and receiving PubSub messages.

C. Adjusting the OPC UA Publisher

The OPC UA publisher receives information from the application, encodes the information in an OPC UA PubSub network message, and transmits/publishes the message via a network interface. Thereby, the OPC UA Specification Part 14: PubSub [35] defines the message format. The specification limits the number of data fields in a PubSub message and each data field's length to 2,147,483,647 (max Int32). This value is not suitable for static WCET analysis, and a trade-off between

TABLE I
PROGRAMMING CONSTRUCTS AND NUMBER OF OCCURRENCES FOR THE
OPC UA PUBLISHER AND SUBSCRIBER

Programming construct	Number of occurrences	
	Publisher	Subscriber
While loop	1	0
Do-While loop	0	0
For loop	1	11
Indirect recursion	1	3
Jumptable	1	1
Other, non-WCET-analyzable code	6	7

the software stack’s flexibility and the desire for a tight WCET bound has to be made.

Therefore, appropriate changes to the open62541 stack were necessary, reducing the maximum number of supported data fields per message. The modified version of the software supports a maximum of two data fields per message with a limited selection of data types. In addition, the application needs to set the open62541 UA_PUBSUB_RT_FIXED_SIZE flag, which defines that the message structure does not change. Furthermore, only data types of fixed size like Boolean, Integer, and Float may be used, but support for variable-length data types like String and ByteString is removed. These limitations are considered acceptable because additional data values can easily be transmitted in separate messages and real-time-critical data, e.g., sensor values, typically are of fixed size.

The WCET analysis process presented in Section IV-B was applied to the publisher of the open62541 [38] OPC UA stack. Table I summarizes how often each annotation and transformation rule was used. The analysis was conducted with the tools provided by the T-CREST project and yielded a WCET of 18,632 processor cycles. This value corresponds to about 232.9 μ s for a processor operating at 80 MHz. A detailed discussion of the evaluation results is presented in [37] and the code is available under [10].

D. Adjusting the OPC UA Subscriber

The OPC UA subscriber receives PubSub messages, decodes their contents, and hands the data values over to the application. The application can then act upon the received data values, e.g., by performing calculations and setting outputs. The same considerations regarding the number of data fields in a message and the supported data types mentioned for the publisher also apply for the subscriber.

However, the subscriber included in the open62541 stack uses dynamic memory allocation for each received frame and each data field. The standard library implementations that handle memory allocation in C (*malloc*, *free*, and related functions) fall in the category of non-WCET-analyzable code. Therefore, these functions had to be re-implemented. The new, WCET-analyzable implementation of *malloc* provides only a fixed number of 32 memory blocks with 512 bytes each. Calling *malloc* returns a pointer to the next free memory block and marks it as “in use”. If no free memory block is available,

malloc causes an out-of-memory error (ENOMEM), which the caller needs to handle. Furthermore, calling *free* releases the memory block corresponding to the address that is passed as a parameter. The new functions share the same method signature with the standard C library and require no additional changes to the remaining code.

Table I summarizes how often each annotation and transformation rule had to be applied for the open62541 subscriber to obtain a WCET analyzable implementation. The analysis yielded a WCET of 443,543 processor cycles, corresponding to about 5,544.29 μ s for a processor operating at 80 MHz.

V. EVALUATION

This section covers the evaluation performed primarily to verify the results obtained from static WCET analysis. Furthermore, the evaluation setup allows determining hardware delays and estimating end-to-end latency.

A. Setup

Figure 5 illustrates the evaluation setup. PublishCallback and SubscribeCallback handle publishing and receiving OPC UA PubSub messages via the open62541 stack. Note that the PublishCallback, in addition to Encode and UdpSend, performs some pre-processing and post-processing of its internal data structures. Likewise, the SubscribeCallback requires some extra operations in addition to UdpRec and Decode. The setup includes two Altera DE2-115 development boards featuring Cyclon IV FPGAs. A Patmos time-predictable processor, which is part of the T-CREST project, is instantiated on each FPGA and operating at a frequency of 80 MHz. The OPC UA publisher and subscriber are executed on these platforms. Furthermore, the two FPGA boards are directly connected with a 100 Mbit point-to-point Ethernet connection of 2 m in length. The publisher and subscriber set and reset general purpose input/output (GPIO) pins at relevant positions in their program flow. A Saleae Logic Pro 8 logic analyzer logs these events on the GPIOs for the subsequent timing analysis. The evaluation setup exchanges a single Int32 counter value that is incremented every time before publishing a new message.

B. WCET Results

The histograms depicted in Figure 6 show the distributions of the execution times measured for PublishCallback, SubscribeCallback, encoding (Encode), sending (UdpSend), receiving (UdpRec), and decoding (Decode) 1000 PubSub messages. As the publisher handles encoding and sending messages, the added execution times of Encode and UdpSend must be lower than the theoretical WCET obtained in Section IV-C. The longest execution time for Encode + UdpSend recorded by the logic analyzer is 136.852 μ s. Therefore, the WCET bound is approximately 70 % higher than the execution time obtained by the measurements.

Similarly, the subscriber receives and decodes messages via the UdpRec and Decode functions, respectively. Again, the combined execution time of these two functions is below the theoretical upper bound obtained via the WCET analysis

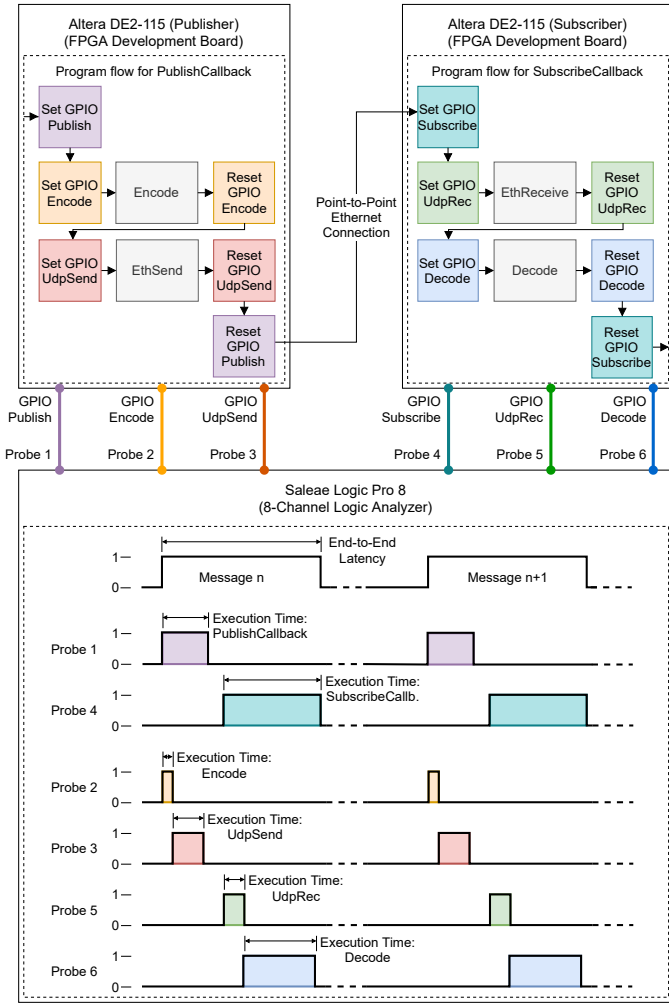


Fig. 5. Evaluation setup

in Section IV-D. The longest execution time measured is 513.08 μ s. Therefore, the WCET bound is approximately 980 % higher than the execution time obtained by the measurements. The over-estimation in the case of the subscriber is worse than for the publisher because of its higher complexity, particularly regarding the challenges arising from dynamic memory allocation.

C. End-to-End Latency Analysis

The measurements presented so far only verify the results of the two WCET analyses. However, as shown in Section IV-A, the OPC UA publisher and subscriber cause only a part of the end-to-end latency. Therefore, Table II sums up all the involved software and hardware delays. The first value TaskSchedulingDelay is not included in this analysis because the counter value used as a payload is only incremented right before triggering the Encode task (cf. Figure 3). The next entries in the table represent the theoretical WCET bounds and the longest execution times measured for Encode, UdpSend, UdpRec, and Decode, which have already been discussed. The remaining entries EthSend, NetworkDelay, and EthRec are

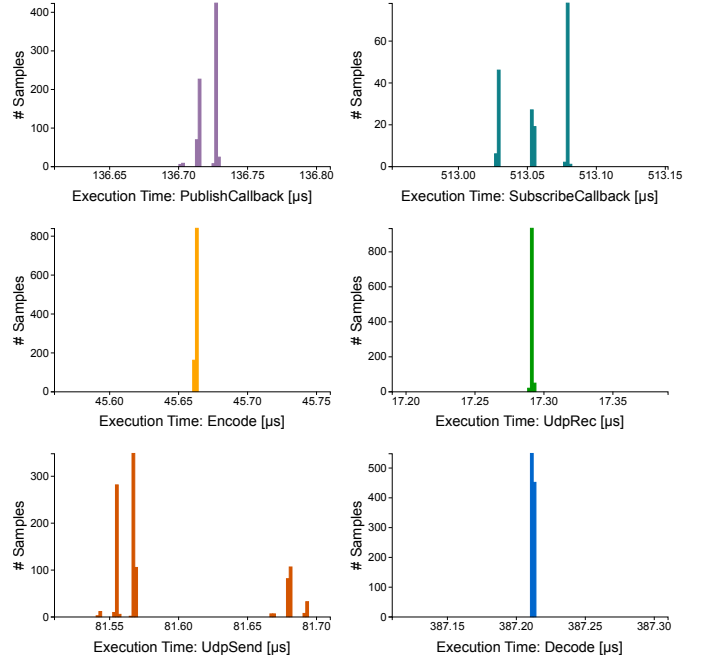


Fig. 6. Histograms of measured execution times for PublishCallback, Encode, UdpSend, SubscribeCallback, UdpReceive, Decode

TABLE II
COMPARISON OF HIGHEST TIMING MEASUREMENT RESULTS AND THEORETICAL UPPER BOUNDS

SW/HW component	Upper bound	Measurement result
TaskSchedulingDelay	not relevant	not relevant
PublishCallback	232.9 μ s	136.85 μ s
▷ Encode	▷ -	▷ 45.67 μ s
▷ UdpSend	▷ -	▷ 81.69 μ s
▷ Pre- and postprocessing	▷ -	▷ 9.49 μ s
EthSend + NetworkDelay + EthRec	HW-dependent	15.4 μ s
SubscribeCallback	5,544.29 μ s	513.08 μ s
▷ UdpRec	▷ -	▷ 17.29 μ s
▷ Decode	▷ -	▷ 387.22 μ s
▷ Pre- and postprocessing	▷ -	▷ 108.57 μ s
End-To-End Latency	5,777.19 μ s + NetworkDelay	642.32 μ s

specific to the evaluation platform and are briefly discussed in the following.

Sending an Ethernet frame using the Altera DE2-115 evaluation platform involves two distinct hardware components: Ethernet MAC and Ethernet PHY [39]. The Ethernet MAC functionality is implemented in the FPGA, while the Ethernet PHY uses a dedicated chip (Marvell 88E1111). The total delay caused by these components for transmitting a frame is subsumed as EthSend. At the subscriber, the time required for receiving an Ethernet frame is subsumed as EthRec. Furthermore, the NetworkDelay of the point-to-point Ethernet connection is in the order of nanoseconds and, therefore, neglectable. The longest total duration of EthSend + Net-

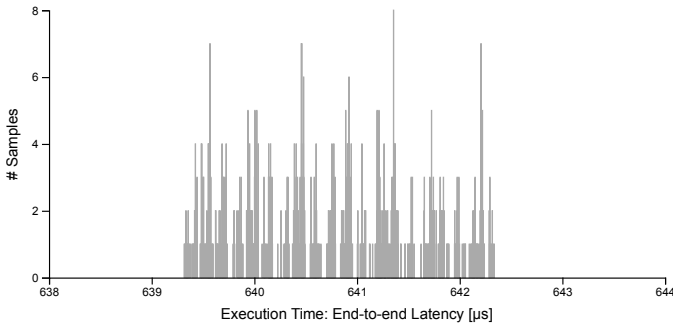


Fig. 7. Histogram of measured end-to-end latency

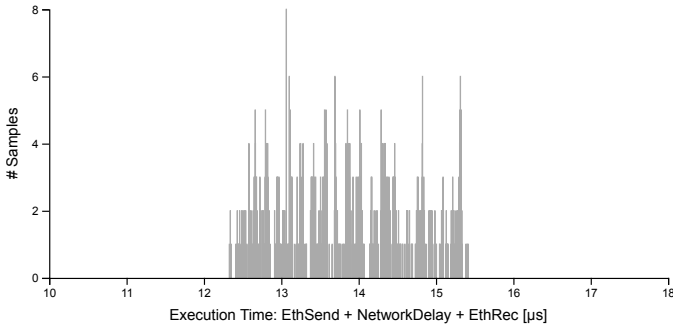


Fig. 8. Histogram of measured NetworkDelay including hardware delays

workDelay + EthReceive was measured at 15.4 μs . As all of these components are implemented in hardware, the jitter observed is minimal, and the theoretical limit for this delay is set equal to the measurement result.

With an assumed NetworkDelay of ≤ 20 μs , the guaranteed upper bound for publishing, transmitting, and receiving a signal via OPC UA PubSub results to 5,797.19 μs . The longest observed end-to-end latency is 642.32 μs and, therefore, within the expected bound. The measurement results for the end-to-end latency and the NetworkDelay are depicted in Figures 7 and 8, respectively.

VI. DISCUSSION

This research aims to adjust the open62541 OPC UA PubSub stack to obtain WCET values, identify relevant challenges and provide a baseline for further research. The subscriber part is significantly more complex than the publisher [37], as it uses dynamic memory allocation to receive unknown size messages. Such code constructs are not WCET analyzable [29] and require reprogramming [28], [34]. Therefore, as indicated in Section IV-D, several assumptions were made regarding the maximum number of data fields in a message and the supported data. The obtained WCET values are valid for various applications as long as they do not violate these assumptions. However, conducting this analysis again with known message sizes and data types could result in more accurate WCETs estimates.

A noteworthy fact is that neither the publisher nor the subscriber has dependencies on global data structures, especially as [30] reports such dependencies as quite common in

industrial code. This circumstance occurs due to the absence of global values changeable by external program parts. Moreover, the implemented loops do not contain any break statements, nor are variables involved in the loop condition manipulated within the loop. It is unclear why the the open62541 OPC UA PubSub contributors choose not to use such dependencies.

The obtained WCET values of the publisher and subscriber are larger than the measured values in the evaluation, which is an expected outcome as the WCET analysis needs to provide worst-case guarantees. This characteristic distinguishes WCET analysis from work done by Pfrommer et al. [22] where a hardware interrupt ensures the timely execution of the publisher. However, addressing the identified code issues would allow closer worst-case guarantees.

In general, the conducted research confirms the feasibility of realizing a distributed real-time end-to-end data transfer environment for the Industrial Internet. However, the results are not directly comparable with other research, as the setup does not include TSN, and the Patmos is a non-standard processor specifically designed to simplify the determination of WCET bounds. Nevertheless, the results are in similar ranges, as in Eymüller et al. [21], where a round trip for a floating-point value (8 to 160 byte) takes on average 268 μs – 369 μs , which equals 134 – 182 μs end-to-end latency. Their latency is approximately 25 % of the highest measured value (642 μs) presented in this article. Possible reasons are that the setup in Eymüller et al. uses specific OPC UA programs, and the evaluation was done on standard office computers. Therefore, it is reasonable to use the WCET values presented in this paper as a baseline for comparison.

Another limitation is that depending on the application scenario and potentially required certification, the claim that the hardware delays (EthSend, NetworkDelay, EthRec) in the evaluation setup are constant may require additional experiments. If other software also accesses the network interfaces, this may be particularly challenging and require hardware support directly in the end systems (e.g., a TSN interface).

VII. CONCLUSION

The paper presents findings and experiences made while adjusting the open62541 OPC UA PubSub stack part for WCET analysis. The use of a simple process and the time predictable open-source T-CREST platform yielded representative WCET estimates. WCET timing values were evaluated by an actual point-to-point setup on two development boards hosting the publisher and subscriber. In combination, the results lay the ground for a distributed real-time end-to-end data transfer environment. The next step in the research process is adding additional time predictable platforms hosting several publishers and subscribers to a TSN network. Adding multiple participants could allow the observation of an entire system's timing behavior, thus creating a reference for industrial implementations. Additionally, the adjustment of the open62541 OPC UA PubSub stack needs to be continued to allow further studies where time-critical communication is relevant, e.g., in control, optimization, data processing, and decision making.

REFERENCES

- [1] T. J. Williams, "The Purdue enterprise reference architecture," *Computers in Industry*, vol. 24, no. 2-3, pp. 141–158, 1994.
- [2] M. Wollschlaeger, T. Sauter, and J. Jasperneite, "The Future of Industrial Communication: Automation Networks in the Era of the Internet of Things and Industry 4.0," *IEEE Industrial Electronics Magazine*, vol. 11, no. 1, pp. 17–27, 3 2017.
- [3] S. Schriegel, T. Kobzan, and J. Jasperneite, "Investigation on a distributed SDN control plane architecture for heterogeneous time sensitive networks," in *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*, 6 2018, pp. 1–10.
- [4] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12. New York, NY, USA: Association for Computing Machinery, 2012, pp. 13–16.
- [5] D. Bruckner, M. Stanić, R. Blair, S. Schriegel, S. Kehrer, M. Seewald, and T. Sauter, "An Introduction to OPC UA TSN for Industrial Communication Systems," *Proceedings of the IEEE*, vol. 107, no. 6, pp. 1121–1131, 2019.
- [6] P. Montag, S. Görzig, and P. Levi, "Challenges of Timing Verification Tools in the Automotive Domain," in *Second International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (isola 2006)*, 2006, pp. 227–232.
- [7] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The Worst-Case Execution-Time Problem—Overview of Methods and Survey of Tools," *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 3, 5 2008.
- [8] M. Schoeberl, S. Abbaspour, B. Akesson, N. Audsley, R. Capasso, J. Garside, K. Goossens, S. Goossens, S. Hansen, R. Heckmann, S. Hepp, B. Huber, A. Jordan, E. Kasapaki, J. Knoop, Y. Li, D. Prokesch, W. Puffitsch, P. Puschner, A. Rocha, C. Silva, J. Sparsø, and A. Tocchi, "T-CREST: Time-predictable multi-core architecture for embedded systems," *Journal of Systems Architecture*, vol. 61, no. 9, pp. 449–471, 2015.
- [9] M. Schoeberl, W. Puffitsch, S. Hepp, B. Huber, and D. Prokesch, "Patmos: a time-predictable microprocessor," *Real-Time Systems*, vol. 54, no. 2, pp. 389–423, 2018.
- [10] P. Denzler, T. Frühwirth, A. Kirchberger, and M. Schoeberl, "Readme [Source code]," <https://github.com/t-crest/rt-ua>, 2020.
- [11] M. Lv, N. Guan, Y. Zhang, Q. Deng, G. Yu, and J. Zhang, "A Survey of WCET Analysis of Real-Time Operating Systems," in *2009 International Conference on Embedded Software and Systems*, 2009, pp. 65–72.
- [12] P. Puschner and C. Koza, "Calculating the maximum execution time of real-time programs," *Real-Time Syst.*, vol. 1, no. 2, pp. 159–176, 1989.
- [13] J. Gustafsson, A. Ermedahl, C. Sandberg, and B. Lisper, "Automatic Derivation of Loop Bounds and Infeasible Paths for WCET Analysis Using Abstract Execution," in *2006 27th IEEE International Real-Time Systems Symposium (RTSS'06)*, 2006, pp. 57–66.
- [14] B. Lisper, "Fully Automatic, Parametric Worst-Case Execution Time Analysis," *WCET*, vol. 3, pp. 77–80, 2003.
- [15] M. Lv, Z. Gu, N. Guan, Q. Deng, and G. Yu, "Performance Comparison of Techniques on Static Path Analysis of WCET," in *2008 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, vol. 1, 2008, pp. 104–111.
- [16] D. Prokesch, P. Puschner, and S. Hepp, "A Generator for Time-Predictable Code," in *Proceedings - 2015 IEEE 18th International Symposium on Real-Time Distributed Computing, ISORC*, 2015, pp. 27–34.
- [17] B. Lickly, I. Liu, S. Kim, H. D. Patel, S. A. Edwards, and E. A. Lee, "Predictable Programming on a Precision Timed Architecture," in *Proceedings of the 2008 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*. New York, USA: Association for Computing Machinery, 2008, pp. 137–146.
- [18] AbsInt, "ait," Available at <https://www.absint.com/ait/>, 2021.
- [19] Rapita Systems, "Rapidtime," Available at <https://www.rapitasystems.com/products/rapidtime>, 2021.
- [20] W. Mahnke, S.-H. Leitner, and M. Damm, *OPC unified architecture*. Springer Science & Business Media, 2009.
- [21] C. Eymüller, J. Hanke, A. Hoffmann, M. Kugelmann, and W. Reif, "Real-time capable opc-ua programs over tsn for distributed industrial control," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2020, pp. 278–285.
- [22] 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, 2018, pp. 1087–1090.
- [23] A. Gogolev, R. Braun, and P. Bauer, "Tsn traffic shaping for opc ua field devices," in *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, vol. 1, 2019, pp. 951–956.
- [24] F. Prinz, M. Schoeffler, A. Eckhardt, A. Lechler, and A. Verl, "Configuration of application layer protocols within real-time i4.0 components," in *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, vol. 1, 2019, pp. 971–976.
- [25] T. Kobzan, I. Blöcher, M. Hendel, S. Althoff, A. Gerhard, S. Schriegel, and J. Jasperneite, "Configuration solution for tsn-based industrial networks utilizing sdn and opc ua," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2020, pp. 1629–1636.
- [26] S. K. Panda, M. Majumder, L. Wisniewski, and J. Jasperneite, "Real-time industrial communication by using opc ua field level communication," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2020, pp. 1143–1146.
- [27] N. Holsti, T. Langbacka, and S. Saarinen, "Using a worst-case execution time tool for real-time verification of the DEBIE software," *Proceedings of DASIA 2000 Conference (Data Systems in Aero-space 2000, ESA SP-457)*, vol. 457, pp. 307–312, 2000.
- [28] S. Byhlin, A. Ermedahl, J. Gustafsson, and B. Lisper, "Applying static WCET analysis to automotive communication software," in *17th Euromicro Conference on Real-Time Systems (ECRTS'05)*, 2005, pp. 249–258.
- [29] J. Gustafsson and A. Ermedahl, "Experiences from Applying WCET Analysis in Industrial Settings," in *10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)*, 2007, pp. 382–392.
- [30] D. Barkah, A. Ermedahl, J. Gustafsson, B. Lisper, and C. Sandberg, "Evaluation of Automatic Flow Analysis for WCET Calculation on Industrial Real-Time System Code," in *2008 Euromicro Conference on Real-Time Systems*, 2008, pp. 331–340.
- [31] B. Lisper, A. Ermedahl, D. Schreiner, J. Knoop, and P. Gliwa, "Practical experiences of applying source-level WCET flow analysis to industrial code," *International Journal on Software Tools for Technology Transfer*, vol. 15, no. 1, pp. 53–63, 2013. [Online]. Available: <https://doi.org/10.1007/s10009-012-0255-9>
- [32] D. Sehlberg, A. Ermedahl, J. Gustafsson, B. Lisper, and S. Wiegatz, "Static WCET Analysis of Real-Time Task-Oriented Code in Vehicle Control Systems," in *Second International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (isola 2006)*, 2006, pp. 212–219.
- [33] P. Axer, R. Ernst, H. Falk, A. Girault, D. Grund, N. Guan, B. Jonsson, P. Marwedel, J. Reineke, C. Rochange, M. Sebastian, R. V. Hanxleden, R. Wilhelm, and W. Yi, "Building Timing Predictable Embedded Systems," *ACM Trans. Embed. Comput. Syst.*, vol. 13, no. 4, 03 2014.
- [34] M. Platzer and P. Puschner, "A Real-Time Application with Fully Predictable Task Timing," in *Proceedings - 2020 IEEE 23rd Int. Symposium on Real-Time Distributed Computing, ISORC*, 2020, pp. 43–46.
- [35] OPC Foundation, "OPC Unified Architecture Specification Part 14: PubSub, Release 1.04," 2018.
- [36] T. Frühwirth, W. Steiner, and B. Stangl, "TTEthernet SW-based End System for AUTOSAR," in *Proceedings of the 10th IEEE International Symposium on Industrial Embedded Systems (SIES)*, Siegen, Germany, Jun. 2015, pp. 1–8.
- [37] P. Denzler, T. Frühwirth, A. Kirchberger, M. Schoeberl, and W. Kastner, "Experiences from Adjusting Industrial Software for Worst-Case Execution Time Analysis," in *Accepted in - 2021 IEEE 24th International Symposium on Real-Time Distributed Computing, ISORC*, 2021.
- [38] F. Palm, S. Grüner, J. Pfrommer, M. Graube, and L. Urbas, "open62541-der offene OPC UA-Stack," *5. Jahreskolloquium "Kommunikation in der Automation" (Komma 2014)*, 2014.
- [39] L. Pezzarossa, J. K. Toft, J. Lønbæk, and R. Barnes, "Implementation of an ethernet-based communication channel for the patmos processor," *Technical University of Denmark*, 2015.