Assessment of OPC UA PubSub at Scale using TSN Infrastructure and Network Calculus

Filip Rezabek, Max Helm, Nicolas Buchner, Monika Smolarska, Benedikt Jaeger, and Georg Carle
Department of Computer Engineering, TUM School of CIT, Technical University of Munich, Germany
Email: {rezabek|helm|buchner|smolarsk|jaeger|carle}@net.in.tum.de

***Abstract*—OPC UA PubSub is a recent addition to the OPC UA industry standard used for machine-type communication in the Industrial Internet of Things. It can be combined with Time Sensitive Networking to offer deterministic latency and bounded jitter. In this work, we model the network interactions between Credit-Based Shaper (CBS) and PubSub using Network Calculus. We derive worst-case delay bounds for various numbers of hops, payload sizes, and competing flows. The derived bounds are confirmed using deployments on commercial-of-the-shelf hardware and open-source solutions relying on a low-latency Linux kernel. Based on the results, we observe that the derived bounds are satisfied in all scenarios of seven hops with a round-trip-time (RTT) varying between 1 to 2.5 ms for respective payload sizes of 59 to 1481 B. Similarly, for jitter we observe fluctuations of $\pm 250\,\mu s$. Complementing the CBS results, we evaluate time-aware shaping, where we observe lower RTTs for average payload sizes of around 800 µs. Our contributions show that OPC UA PubSub applications can co-exist with TSN.**

***Index Terms*—COTS, TSN, OPC UA, Network Calculus**

I. Introduction

Time Sensitive Networking (TSN) and the Open Platform Communications (OPC) Unified Architecture (UA) in its Publish–Subscribe (PubSub) variant aim to offer deterministic latency guarantees to machine-type communication for industrial automation. OPC UA is developed by the OPC Foundation [1] and brings a platform-independent design, which offers an efficient and secure framework for interoperability between different systems and devices. Its ability to integrate with different hardware and software, along with support for client-server and PubSub communication models, makes OPC UA a key enabler for the Industrial Internet of Things (IIoT).

Relying on Ethernet enables higher link rates and a combination of high and low traffic priority on the same medium [2]. However, by default, Ethernet does not provide any time guarantees required for real-time scenarios. Therefore, the TSN family of IEEE standards[1] brings reliability and determinism to Ethernet networks. It allows precise timing across a network, ensuring deterministic latency and jitter for exchanged packets. By integrating TSN, OPC UA PubSub can fulfill industrial application requirements, ensuring synchronized and timely communication between devices.

Despite the key role of real-time communication in OPC UA [2], there needs to be more research that explores the capabilities of open-source OPC UA on Commercial off-the-Shelf (COTS) hardware (HW). The current studies primarily rely on specialized HW and proprietary software (SW) or do not evaluate the deployments at scale.

To determine the performance of OPC UA irrespective of the HW setup, we utilize and extend a Network Calculus (NC) model to determine upper delay bounds. We model the behavior of the Credit-Based Shaper (CBS) queuing discipline (qdisc) [3] and the open62541 project[2]. The open62541 project is selected as it is one of the most widely established open-source implementations of OPC UA stack, including the PubSub variant. We also assess the Time-Aware Shaper (TAS) and its Linux Time Aware Priority Shaper (TAPRIO) qdisc implementation. For TAPRIO, we do not provide a theoretical model. To evaluate the real-time performance of OPC UA over TSN and the fulfillment of the theoretical upper delay bounds for CBS, we conduct several experiment campaigns to measure Round Trip Time (RTT) and jitter, which are crucial metrics as outlined in the TSN experimentation methodology [4]. We focus on the usage of Stream Reservation (SR) classes, with requirements of 2 ms delay and 125 µs jitter over seven hops. We extend the open-source[3] Environment for Generic In-vehicular Networking Experiments (EnGINE) [5], [6] framework with open62541 project PubSub implementation to conduct our experiments. Building on top of EnGINE, we rely on open-source solutions using Linux deployed on COTS HW. Compared to previous works [7]–[11], we aim to determine the OPC UA performance at scale with a higher number of peers (up to seven hops one way), varying message sizes, and competing traffic. This provides insights into various use cases, such as tactile interaction, safety monitoring and control alarms, automated guide vehicles, smart grid protection, and motion control.

In this paper, we present the following contributions:

C1 Model OPC UA PubSub over CBS using NC
C2 Evaluate the model in a HW deployment, validating delay bounds
C3 Assess the performance of OPC UA PubSub with various qdiscs at scale

All links are valid as of 21 April 2024.
[1]https://www.ieee802.org/1/pages/tsn.html
[2]https://github.com/open62541/open62541
[3]https://github.com/rezabfil-sec/engine-framework

II. Background

In the following, we introduce TSN, OPC UA PubSub, and NC. For TSN, we focus on the Precision Time Protocol (PTP) [12], IEEE 802.1Qav [3], and IEEE 802.1Qbv [13] and their corresponding Linux implementations.

A. *Time Sensitive Networking*

To support TSN experiments with COTS HW and open source solutions, we introduced the *EnGINE* framework [6]. It is extended by a methodology [4] for TSN experiments and enabled its integration with the Objective Modular Network Testbed in C++ (OMNeT++) simulator [14]. Previous experiments focus on achieving the latency, jitter, and packet loss requirements outlined by [15] and [16] for Intra-Vehicular Networks (IVNs), which we also consider in our evaluations. We utilize Linux qdisc implementations for TAPRIO and Earliest Time First (ETF), as well as CBS. PTP provides accurate time synchronization in the network.

1) Precision Time Protocol: In a TSN system, precise time synchronization can be achieved through the use of the PTP, defined by the IEEE 802.1AS standard [17]. System clocks are individually synchronized via PTP instances, organized in a master-slave hierarchy. The exchange of timing messages between the master and slave occurs over the network (either at the Link Layer or Transport Layer), synchronizing the slave clock to the master clock. The Grandmaster Clock (GM) establishes the reference time for the entire system clock, situated at the top of the hierarchy.

For time synchronization on Linux, we rely on the *linuxptp* project [18]. It includes the *ptp4l*, *phc2sys*, and *pmc* tools[4]. The *ptp4l* tool implements the PTP standard IEEE 1588 [12]. If the used Network Interface Card (NIC) supports IEEE 802.1AS, it achieves nanosecond precision using hardware timestamping. We use COTS Intel® I210 NIC, which supports the IEEE 802.1AS standard. The *ptp4l* daemon must run on all interfaces to synchronize and find the PTP GM. The synchronization of clocks on one system is handled by *phc2sys*. It can run in automatic mode, using the information of *ptp4l* to synchronize clocks.

2) Credit-Based Shaper: CBS, as defined in the IEEE 802.1Q-2022 [19] family of standards, allows for bandwidth allocation to SR classes and regulates and secures the specific traffic class load. It is implemented as a child qdisc and is combined with root Multiqueue Priority Qdisc (MQPRIO), which performs traffic classification. The bandwidth allocation is achieved by a scheduler defining which frames shall be dequeued next using a credit system. Figure 1 shows the credit lifecycle policed by four CBS parameters - *highCredit*, *lowCredit*, *idleSlope*, and *sendSlope*. The first two parameters limit the maximum and minimum credit. *idleSlope* marks the credit replenishment rate, and *sendSlope* the credit spending rate. After a frame arrives in a queue the credit starts to build up ①. After *highCredit* is reached ②, the frame is sent after

[4]https://linux.die.net/man/8/ptp4l, https://linux.die.net/man/8/phc2sys, https://linux.die.net/man/8/pmc

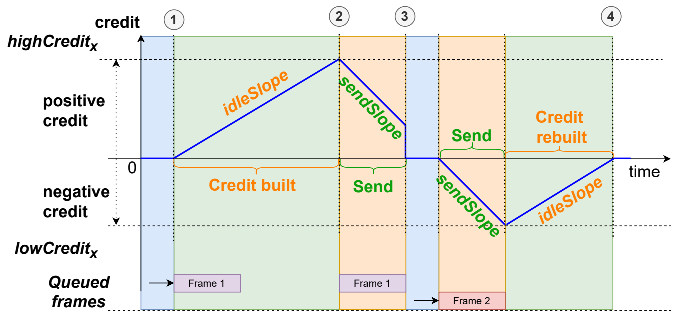


Fig. 1: Behavior of CBS, based on [3], [4]

non-policed frames are transmitted, following the *sendSlope* parameter. In case no additional frames are in the queue, the credit drops to 0, ③. The next frame gets sent out directly in case credit is $\geq$ 0. The credit must be rebuilt if it is below 0, ④. With an addition of priority enforcement mechanisms, and proper configuration, CBS offers soft guarantees for delay, jitter, and packet loss.

3) Time-Aware Shaper: The TAS or TAPRIO qdisc belongs to the family of synchronous TSN standards requiring precise clock synchronization. Like the MQPRIO qdisc, packets are initially mapped to traffic classes and transmit queues. This allows various SR class traffic to have a dedicated transmission window within a configurable cycle time length. During this window, the gate of the given class is open, and packets are sent only when there is enough time for the transmission until the gate closes. This separates various SR class traffic in the time domain with pre-defined transmission windows. The ETF qdisc accompanies the TAPRIO qdisc and is used to dequeue packets at a specific `TxTime`.

B. *OPC UA PubSub*

OPC UA offers a secure and efficient framework for interoperability between various devices and systems. It supports different communication patterns, originally focusing on client-server. In 2018, the OPC foundation introduced the PubSub extension [1]. This model is crucial for factory plans and machine-type communication. We rely on the open62541 project[5] which supports the PubSub protocol [20]. OPC UA PubSub encompasses three communication parties. A publisher sends messages to subscribers through a middleware. While the PubSub protocol does not specify the middleware itself, it relies on underlying protocols for its operation. We map the OPC UA PubSub messages binary Unified Architecture Datagram Protocol (UADP). UADP can operate on top of Ethernet or UDP. Due to the support of TSN, we focus only on brokerless OPC UA over Ethernet. To transfer the priorities of the traffic, we use the VLAN headers and store the priority in Priority Code Point (PCP) field. Overall, we utilize UADP over Ethernet. Figure 2 depicts the frame structure with the option to increase the payload in multiples of 9 B.

[5]https://github.com/open62541/open62541

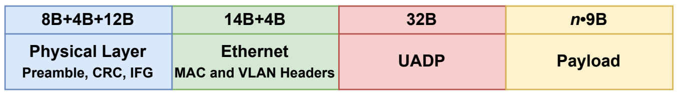


Fig. 2: Frame structure used in the experiments

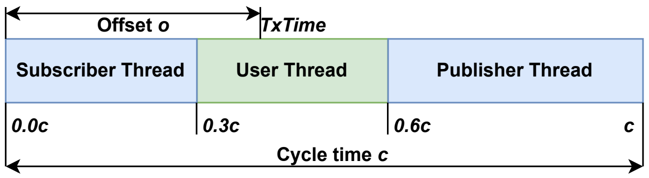


Fig. 3: open62541 implementation behavior

The open62541 project implements the IEEE 802.1Qbv standard in software, as it was not present in the Linux kernel during its creation. The PubSub setup requires two hosts: the publisher **Pub** and loopback **Loop**. The OPC UA server periodically retrieves and increments data variables using an *application thread*. Additionally, two supplementary threads are responsible for publishing and subscribing. Figure 3 presents the behavior of the open62541 implementation. A cycle time c defines the duration of a cycle (emulating the TAPRIO cycle time). At $0.4c$ before the start of the next cycle time, the publisher thread is activated and initiates publication. The transmission time is set to the start of the next cycle time plus a configurable offset o. The subscriber thread is executed at the start of the cycle ($0.0c$). At $0.3c$, the user thread stores the values the subscriber receives in the OPC UA address space of the loopback host or increments the variables in the publisher host. At $0.6c$, the publisher thread is again activated, and the execution follows as described above.

C. Network Calculus

NC is a mathematical framework for calculating worst-case performance bounds in communication networks [21]–[23]. We can derive worst-case upper bounds on delay and backlog (queue sizes) using arrival- and service curves. An arrival curve is an envelope on the cumulative arrival of traffic. The service curve is a characterization of data processing by a node in the network. The upper bounds on delay and backlog at a single network node are the maximum horizontal and vertical distances between the two curves, as shown in Figure 4. In multi-node networks, an additional network analysis method is required to combine all arrival- and service curves into a single performance bound. The most versatile (albeit pessimistic) network analysis method is the Total Flow Analysis (TFA) [22].

Different network technologies, e.g., CBS, are modeled using different service curves in NC. A physical node with a CBS on each interface is divided into a server for each interface with one or more service curves that match the CBS configuration parameters of this interface [24]. Service curves can be further divided into minimum- and maximum service curves, modeling the minimum and maximum service offered by a node [23].

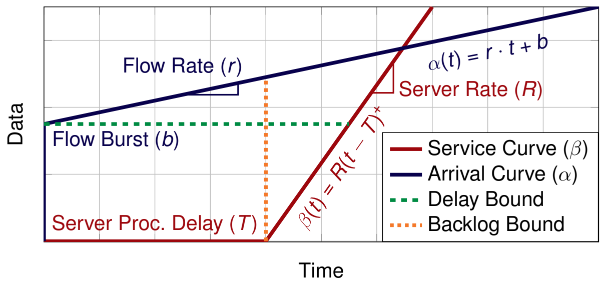


Fig. 4: Basic NC components and bounds

III. Related Work

This section provides an overview of related work relevant to OPC UA evaluations in combinations with TSN. We summarize the works and outline our unique contributions in Table I. Most of the works focus on TAS, and two touch on CBS. Authors in [9], [10] employ OPC UA PubSub to establish communication between Peer-to-Peer (P2P) connected devices, using the Intel® I210 NIC and the open-source *open62541* stack. In [10], the TAS window cycle time is set to 100 µs, while in [9], it is configured to 200 µs. We stick to 100 µs as it corresponds to our HW requirements. Eckhardt et al. utilize specialized embedded hardware for TSN instead of COTS HW without specifying the SW stack [25]. TAS is selected without providing any information regarding the implementation and uses the same window cycle time as our work. For the collection of RTT results, they rely on SW timestamps. Farzaneh et al. conduct experiments using a larger cycle time of 500 µs. The setup relies on switches with a TAS implementation in Field Programmable Gate Arrays (FPGAs). Unlike our work, the traffic is unidirectional, with the first switch acting as the source and the second as the sink without the option to measure RTT. Using two proprietary TSN switches, authors in [26] investigate bridged TAS and TSN proprietary hardware modules on end devices. Like [27], the TSN traffic is only sent in one direction and relies on OPC UA with server/client communication. Only the client is open-source SW. For their results, they achieve bounded latency of 900 µs, which is significantly higher than previous and our works. Arestova et al. focus on the TAPRIO framework and experiments on a two-node network, analyzing one-way data flow from sender to receiver [11]. The sender utilizes a specialized TSN NIC from Kontron, while the receiver employs an Intel® I210 NIC for hardware timestamp capture. Our work also utilizes the OPC UA PubSub implementation *open62541*, but we use 100 µs for priority traffic window, which differs to 1 ms used in [11]. Grüner et al. [28] evaluate OPC UA PubSub over TSN on COTS hardware, relying on the *open62541* stack, similar to [10]. Their assessment is limited to a P2P topology with Intel® I350 NIC and employs a real-time kernel. While the NIC lacks advanced TSN features, the focus remains on TAPRIO

TABLE I: Our contributions extended from [7]

Works	[10] [9]	[25]	[27]	[26]	[8] [30]	[11]	[28]	[29]	Our work
PubSub‡	✓	✓	✗	✗	✗	✓	✓	✓	✓
P2P	✓	✓	✗	✗	✗	✓	✓	✓	✓
Bridge	✗	✗	✓	✓	✓	✗	✗	✓	✓
ETF	✓	✗	✗	✗	✗	✗	✗	✗	✓
CBS	✗	✗	✗	✗	✓	✗	✗	✗	✓
TAPRIO	✗	○	✓	✓	✓	✓	✓	✓	✓
COTS HW	✓	✗	✗	✗	✗	✗	✓	✗	✓
Open-source	✓	?	✗	client	✗	✓	✓	✓	✓
Modeling	✗	✗	✗	✗	✗	✗	✗	✗	✓
Scale†	✗	✗	✗	✗	✗	✗	✗	✗	✓

‡ OPC UA PubSub, †Scale in topology size, payload sizes, and number of flows
✓ full, ○ partial, ✗ no, ? unknown satisfaction

in the OPC UA application, enhanced with eXpress Data Path for loopback performance. In contrast, [29] extends the *open62541* OPC UA PubSub stack with 802.1Q VLAN tag for IEEE 802.1Qbv scheduling. Their study on time-predictable T-CREST platforms explores end-to-end latency and worst-case execution time analyses with varied payloads. Unlike [29], our work focuses on COTS hardware for timing analyses, encompassing all Linux TSN queuing disciplines beyond IEEE 802.1Qbv, broadening the comparative scope. Gogolev et al. evaluate proprietary software-based TAS switches using specialized hardware, employing OPC UA server and client for requests [8]. However, it only focuses on average RTT. As the only study, in subsequent work, the authors combine TAS with CBS to limit bandwidth for Best Effort (BE) traffic, revealing that TAS' impact is more significant than CBS [30].

As outlined in Table I, we provide a detailed overview of relevant Linux qdiscs using the OPC UA PubSub software and COTS hardware in scale. Besides, we verify the theoretical bounds presented by the NC model we introduce specifically for the PubSub application and corresponding traffic flows.

IV. Design

This section provides an overview of the theoretical bounds calculation for PubSub over CBS, the experiment design to verify those, TAPRIO considerations, and the evaluation setup.

A. Theoretical Bounds

We model the complete behavior of OPC UA over CBS using network calculus. We base our CBS implementation on *Boyer and Azua* [24]. We derive, to the best of our knowledge, a novel—albeit simple and possibly pessimistic—service curve description for an OPC UA PubSub service.

OPC UA PubSub Service Curve The OPC UA PubSub service has an internal processing delay before publishing information. We assume that the publishing process after the delay is instantaneous and does not happen with a rate—other effects, such as the service rate of the network interface card, can be considered using an additional rate-latency service curve. Therefore, the PubSub service can be modeled using a burst-delay service curve as shown in Equation (1).

$$\delta(t) = \begin{cases} \infty, & \text{if } t \geq T \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

The only parameter needed to define this equation is the processing delay T. It is the maximum delay this system induces on data without any queueing or serialization effects. Based on Figure 3, the maximum processing delay occurs when a packet arrives just after the last point in time where it can be included in the publishing process. Let t_0 be the time at which the packet arrives. This corresponds to the time when the subscriber thread changes to the user thread ($t_0 = 0.3 \cdot c$). The first point in time where the packet is considered eligible for publishing is the beginning of the next subscriber thread in the next cycle time ($t_1 = t_0 + 0.7 \cdot c$). The packet will be published at the offset after the next publisher thread ($t_2 = t_0 + 0.7 \cdot c + 1.0 \cdot c + o$). Therefore, a packet's maximum time before publication is $T = 1.7 \cdot c + o$.

Theorem 1. *The worst-case delay induced by an OPC UA PubSub service with cycle time c and offset o on any arriving packet is* $1.7 \cdot c + o$.

Proof. Assume the packet arrives at $t_0' = t_0 - \epsilon, \forall \epsilon : 0 < \epsilon \leq 0.3 \cdot c$. Then the packet is eligible for publishing at $t_1' = t_0 + 0.3 \cdot c$. It would then be published at $t_2' = t_0 + 0.7 \cdot c + o$. This gives a processing delay of $T' = 0.7 \cdot c + o + \epsilon$. In the worst-case for $\epsilon = 0.3 \cdot c$ this leads to $T' = 1.0 \cdot c + o$.

Assume the packet arrives at $t_0'' = t_0 + \epsilon', \forall \epsilon' : 0 < \epsilon' \leq 0.7 \cdot c$. Then the packet is eligible for publishing at $t_1'' = t_0 + 0.7 \cdot c$. It would then be published at $t_2'' = t_0 + 0.7 \cdot c + 1.0 \cdot c + o$. This gives a processing delay of $T'' = 1.7 \cdot c + o + \epsilon'$. In the worst-case for $\epsilon' = 0$ this leads to $T'' = 1.7 \cdot c + o$.

Since $max(\epsilon) + max(\epsilon') = 1.0 \cdot c$ is equal to one full cycle time, the worst-case processing delay is $T = T'' = 1.7 \cdot c + o$. □

Theorem 2. *The best-case delay induced by an OPC UA PubSub service on any arriving packet is* $0.7 \cdot c + o$.

Proof. Assume the packet arrives at $t_0' = 0.3 \cdot c$. Then the packet is eligible for publishing at $t_1' = t_0' + 0.3 \cdot c$. It would then be published at $t_3' = t_0' + 0.3 \cdot c + 0.4 \cdot c + o$. This gives a processing delay of $T' = 0.7 \cdot c + o$. It is best case because $\forall t_0'' < t_0'$ the processing delay would increase by $t_0' - t_0''$. Similarly, $\forall t_0'' > t_0'$ the processing delay would increase by at least c. □

We derive minimum- and maximum burst-delay service curves based on Theorem 1 and Theorem 2 as shown in Equation (2) and Equation (3).

$$\delta^{min}(t) = \begin{cases} \infty, & \text{if } t \geq 1.7c + o \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

$$\delta^{max}(t) = \begin{cases} \infty, & \text{if } t \geq 0.7c + o \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

TABLE II: Payload and frame sizes in Bytes and their relation to repeated node counts (*RNC*). PHY - Physical layer

RNC	2	4	8	16	32	64	128	160
Payload	59	77	113	185	329	617	1193	1481
Total (w/o PHY)	77	95	131	203	347	635	1211	1499
Total (w/ PHY)	101	119	155	227	371	659	1235	1523

CBS We utilize arrival curves for periodic traffic and service curves for the CBS derived by *Boyer and Azua* [24].

NC Bound Implementation The implementation of the network analysis is a TFA [22] realized in *Nancy* [31]. The TFA CBS network analysis is available online[6].

B. Experiment Design

Our experiments focus on CBS and TAPRIO qdiscs at scale. For CBS, we verify the theoretical bounds by increasing the number of hops, modifying the bitrate for the CBS configurations (under- or over-provisioning), and increasing payload size. Also, we assess the impact of competing flows considering one flow, two flows (higher priority for OPC UA PubSub and lower priority for 100 Mbit/s synthetic traffic), and three flows (same as two flows with an additional 100 Mbit/s best effort flow). We do not consider competing flows for TAPRIO, as we expect less interaction between the flows due to the time separation.

Table II shows the increase of the frame size by modifying the `REPEATED_NODECOUNTS` (RNC or N in short) in the open62541 project. Considering Figure 2, we can compute a total frame size for $RNC = 2$. The n is computed as $n = 1 + RNC = 3$, adding the headers results in a total size of 101 B on the physical layer. Physical layer sizes are crucial for correct CBS configuration. We choose a maximal RNC as 160 to roughly correspond to the MTU of 1518 B. We measure several metrics including RTT, one-way delays, jitter, throughput, and packet loss. For all experiments, we generate 200 000 packets and trim the first and last 20 000 packets to avoid noise caused by the ramping up- and closing period.

C. Experiment Setup

For our evaluations, we pick 11 identical nodes equipped with Intel® Xeon® E3-1265L V2 CPU and 16 GB RAM. All setups are interconnected with Intel® I210 NIC supporting 1GbE Ethernet and complying with IEEE 802.1AS, IEEE 802.1Qav, and IEEE 802.1Qbv. We rely on 11 nodes instead of eight nodes required for the seven hops due to the PTP and TAPRIO coexistence challenge [32]. This challenge does not allow for PTP timing messages to be exchanged as non-policed traffic and must be used as priority traffic. To avoid that, we use additional wiring and connections among the nodes. All selected components are easily accessible and belong to the COTS HW segment.

[6]https://github.com/Moni5656/npba/tree/main/ComputationMethods/NancyComputations/CBS

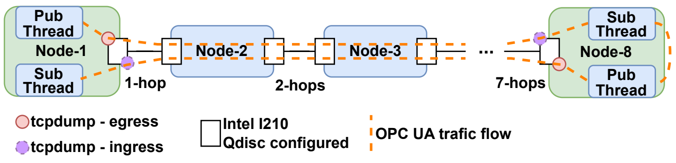


Fig. 5: Simplified line topology of the setup with varying number of hops and tcpdump collection points

We focus on a line topology, as shown in Figure 5, where we can add nodes to increase the total number of hops. This results in seven hops in one direction or 14 hops for RTT measurements. To collect the data, we rely on `tcpdump` using the HW timestamping in ingress and SW timestamping in the egress direction. The RTT is measured using the packet capture data on the same node. The data is correlated using the sequence numbers stored inside the payload. Corresponding qdisc configurations are applied on all used NIC ports. The configuration is the same in both directions for CBS, but for TAPRIO, it is mirrored to account for processing offset on each node. We use Linux traffic control to apply the configuration.

All experiments and configurations are deployed using the EnGINE framework [6]. We plan to update the repository with the experiment artifacts, integrated source code, and experiment campaigns for the published version. For the scope of this work, we had to modify the processing pipeline to accommodate for RTT measurements and integrate the open62541 project. We rely on low-latency Linux Ubuntu Focal for all our experiments. The low latency kernel enables CPU affinity and isolation. This is useful as we can isolate and pin the OPC UA PubSub and the Linux networking stack to specific cores. Instead of relying on the Linux bridge on the intermediary nodes, EnGINE uses Open vSwitch (OvS) to forward the traffic. To visualize the results the measured latencies are plotted as boxplots. For each box we show the minimum, maximum, mean, and median (as orange line), along with the quartiles Q_1 and Q_3 as box and 1.5 times the inter-quartile range as whiskers.

V. Evaluation

This section covers the evaluation of our theoretical model for CBS and the general assessment of TAPRIO. We highlight some of the default parameters selected for our experiments.

A. Default Parameters

For our work, we rely on several qdiscs and OPC UA PubSub applications, which bring their own set of optimal parameters that are system-dependent. For TAPRIO and ETF qdiscs, we need to define the values δ and `txtime-delay` and shift to accommodate for processing delay on each hop. Similarly, for the OPC UA PubSub, we talk about the cycle time c and offset o. Fortunately, we identified ideal values from previous works in our systems that can be reused [4], [7]. Similar values must be found for other setups using the

TABLE III: Selected key parameters for CBS | TAPRIO

cycle time c	offset o	δ	`txtime-delay`	Shift per hop
250 µs | 200 µs	150 µs | 100 µs	150 µs	180 µs	20 µs

TABLE IV: Upper bounds in ms calculated by NC for various hop counts and message sizes. Default CBS values.

Hops	Message size							
	59 B	77 B	113 B	185 B	329 B	617 B	1193 B	1481 B
1	0.954	0.972	0.995	1.018	1.037	1.046	1.045	1.043
2	1.334	1.370	1.415	1.462	1.499	1.518	1.516	1.511
3	1.714	1.767	1.836	1.906	1.961	1.989	1.987	1.979
4	2.094	2.165	2.256	2.350	2.423	2.461	2.458	2.447
5	2.474	2.563	2.677	2.794	2.885	2.933	2.929	2.915
6	2.854	2.960	3.097	3.238	3.347	3.404	3.400	3.383
7	3.234	3.358	3.518	3.682	3.809	3.876	3.871	3.851

approaches outlined in [4], [7]. A summary of the selected values is in Table III. For TAPRIO experiments we modify the cycle time c to 200 µs and offset of 100 µs. This way, it matches the TAPRIO window cycle of 100 µs with 40 µs allocated for the highest priority traffic. For other TSN related values, we follow the gPTP profile for PTP operating in P2P mode and on L2.

For a proper configuration of CBS, we require information on packet spacing and physical layer frame sizes generated by the application. Considering Table II, we get the frame sizes and the cycle time c of 250 µs (corresponding to the packet spacing), resulting in the generation of 4000 packet/s. For the under- and overprovisioning, we modify the calculated bitrate by the negative (underprovisioning) or positive (overprovisioning) percentage, which is then reflected in the CBS parameters calculation. Such behavior might better accommodate the imperfections of the application cycle time c. When evaluating multiple flows for CBS, we generate additional 100 Mbit/s flows. We consider 1180 B payload sizes (resulting in 1250 B on the wire) and packet spacing of 100 µs using `iperf3`. For 2 flows, we police both flows - higher for PubSub traffic and lower priority for `iperf3` traffic. For 3 flows, we add BE traffic flow.

B. Assessment of CBS Bounds

For the evaluation, we first define the theoretical bounds for various message sizes and number of hops following the NC approach presented in the Section IV-A. Table IV summarizes the values for the expected payload sizes and packet spacing of 250 µs corresponding to the cycle time c. With each hop, we model an increase of roughly 400 µs accommodating also for the delay introduced by the loopback OPC UA PubSub application.

Figure 6 shows the calculated end-to-end delay bounds of a baseline, as well as over- and underprovisioning. We can observe that overprovisioning leads to marginal decreases in delay bound while underprovisioning leads to a more significant increase in delay bounds. Underprovisioning can additionally lead to infinite delay bounds for larger packet sizes. More underprovisioning consistently leads to a higher share of infinite delay bounds.

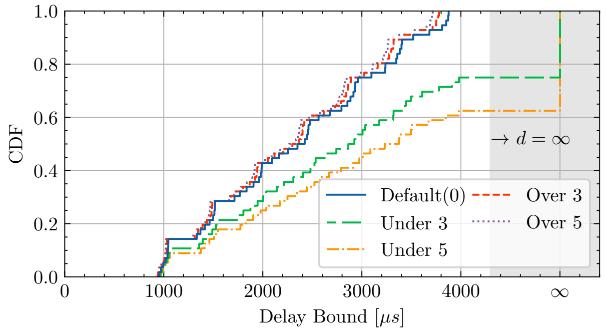
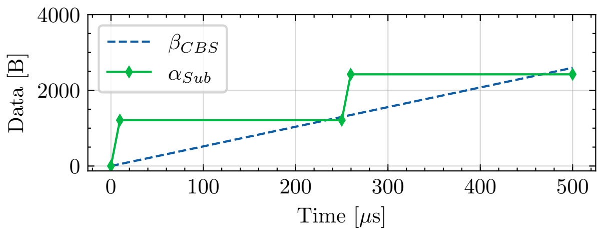
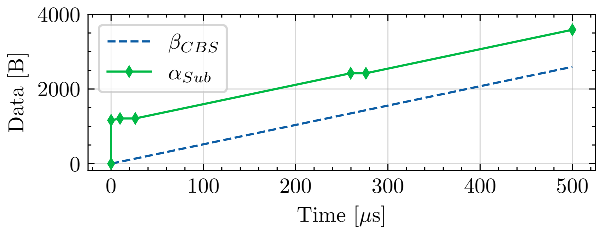


Fig. 6: RTT NC bounds, including the PubSub processing, including CBS configuration with over- and under-provisioning of 3 and 5 %.



(a) First CBS server

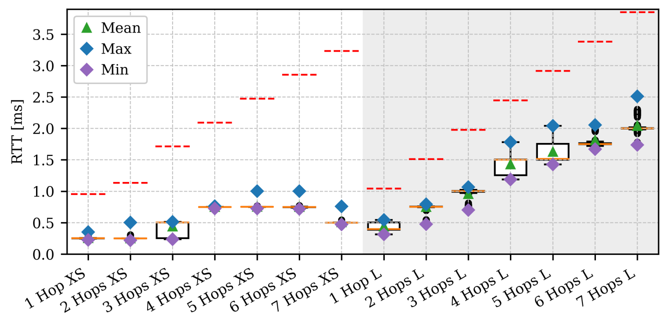


(b) Second CBS server

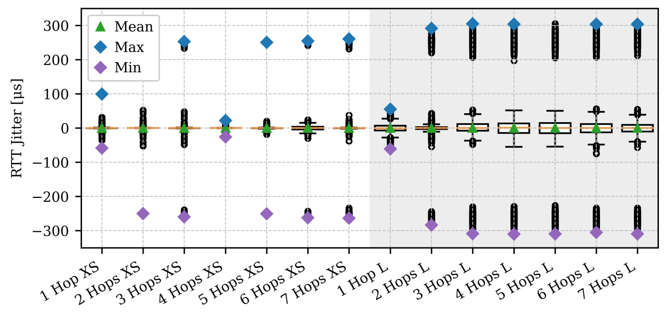
Fig. 7: NC arrival curve of OPC UA subscriber traffic (α_{Sub}) and CBS service curve (β_{CBS}) at different network positions

Figure 7 shows arrival- and service curves, i.e., the NC model of PubSub traffic and CBS processing, at the first and second hop of the network. We can observe the periodic nature of the OPC UA traffic in Figure 7a. This effect is mainly negated by the smoothing effect of CBS on the arrival curve as shown in Figure 7b.

First, we assess RTT for a single flow for 1-7 hops and the smallest (**XS**) and the largest (**L**) message sizes along with the calculated upper bounds. As shown in Figure 8a, we can observe an increase in delay with each additional hop for **L**. The increased latency is expected, considering the RTT is over 14 hops. This leads to 50 to 150 µs delay per hop, depending on the message size. For CBS, especially with larger payload sizes, the algorithm sacrifices latency over determinism. However, for **XS** we observe rather less delay caused by each hop and sometimes even better performance for more hops. The improved performance corresponds to the



(a) RTT Delay, 1 flow, 1-7 hops, red lines are NC bounds



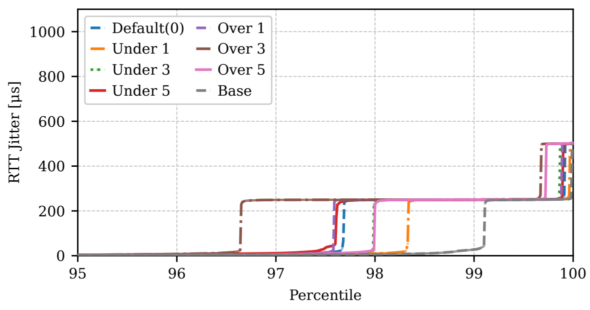
(b) RTT Jitter, 1 flow,1-7 hops

Fig. 8: Comparison of CBS Delay and Jitter for various hops and payload sizes. **XS:** 59 B, **L:** 1481 B

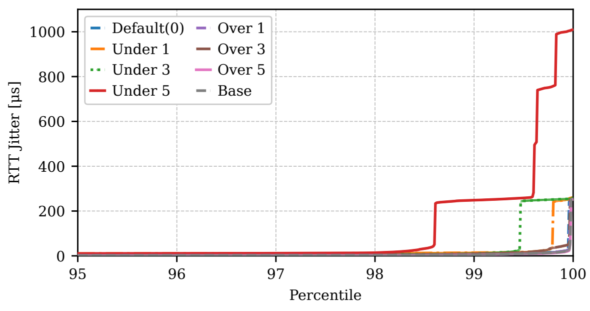
artifact caused by the PubSub application and matches the model presented in Theorem 1 and Theorem 2. Even though we have fewer hops, the packet arrives too late to reach the *subscribe thread* and must wait for an additional cycle, whereas for the seven hops the packet arrives right on time to be processed. Figure 8b follows the spread seen in boxplots, where depending on the number of hops, at least some packets miss the cycle on the PubSub application, which corresponds to the spread of roughly ±250 µs matching the cycle time c.

To possibly improve the performance, we experiment with under- (**under**) and overprovisioning (**over**) values and compare them to **default** CBS and baseline (**base**) values. From Figure 9a, we cannot collect conclusive results, as we observe in some situations higher or lower values of under- or overprovisioning yielding better jitter for the 59 B payloads. However, with a larger payload size of 329 B, shown in Figure 9b, we clearly see that underprovisioning performs worse than overprovisioning. To narrow down the impact of overprovisioning values, we continue evaluating larger payload sizes Figure 10. In Figure 10a we observe comparable performance for payload **S** (617 B), which was selected based on its stability as shown in Figure 6. For **L** (1481 B), a smaller spread is present for latency of 1 % overprovisioning. As a result of this latency measurement and jitter values, as shown in Figure 10b, made us decide to continue with 1 % overprovisioning value and the default CBS.

The last family of experiments for CBS assess multi-flow scenarios covering 1 to 3 flows comparing default and 1 %

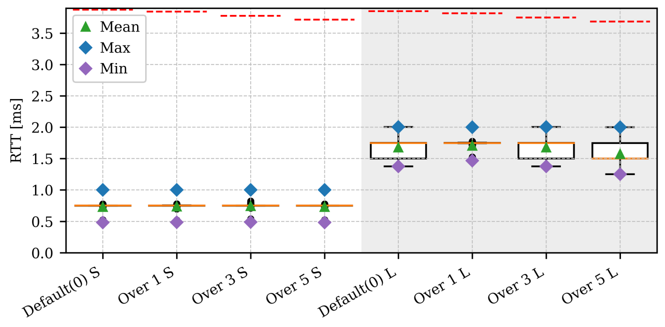


(a) 95th percentile for payload size of 59 B, 7 hops

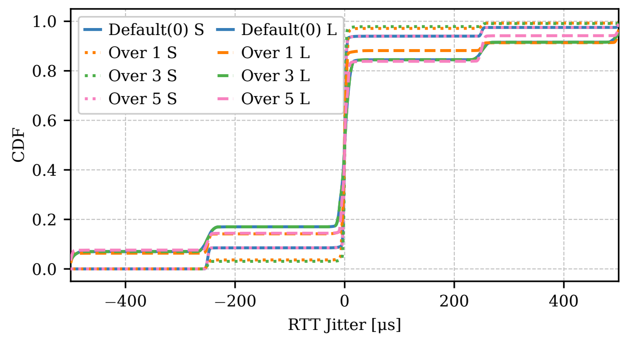


(b) 95th percentile for payload size of 329 B, 7 hops

Fig. 9: 7 hops jitter CBS with a default value, baseline (no CBS), and under- and over-provisioning of 1%, 3%, and 5%.

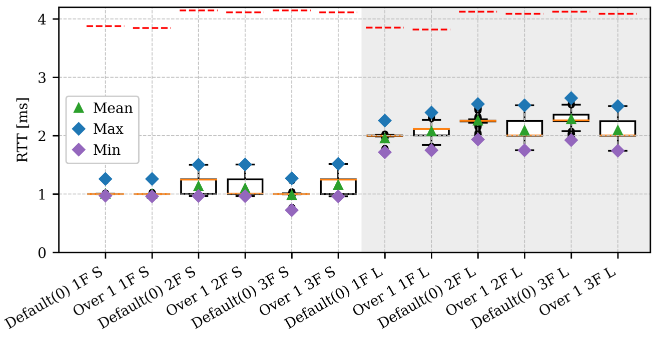


(a) RTT delay with delay bounds

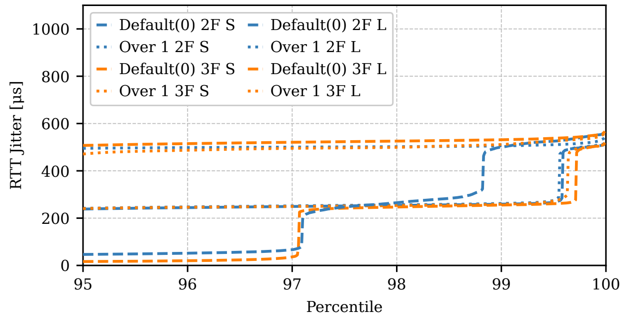


(b) CDF of jitter for various payloads and mode

Fig. 10: Comparison of CBS Delay and Jitter for 7 hops, payload sizes and overprovisioning of 1%, 3%, and 5%. **S:** 617 B, **L:** 1481 B.

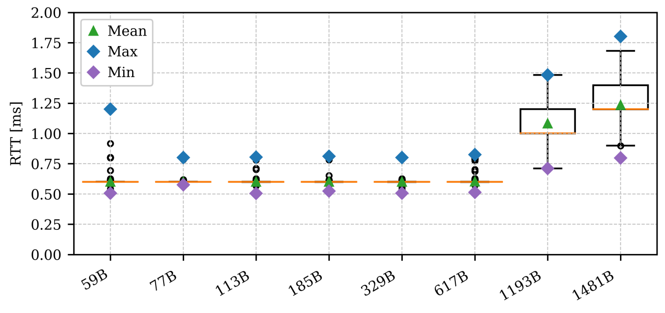


(a) RTT delay with delay bounds

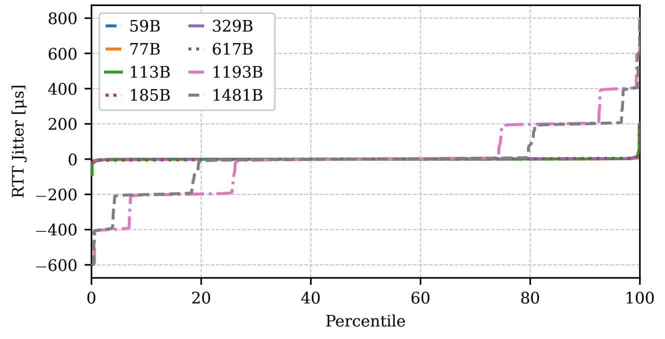


(b) 95th percentile of jitter for various flows and payloads

Fig. 11: Comparison of multiflow CBS Delay and Jitter for 7 hops and payload sizes. **S:** 617 B, **L:** 1481 B. Compares CBS default value and over-provisioning of 1%



(a) RTT for seven hops, various payload sizes of N



(b) Percentile jitter for seven hops, various payload sizes of N

Fig. 12: Comparison of seven hops TAPRIO Delay and Jitter for payload sizes N.

overprovisioning. Comparing the delays in Figure 11a, we observe that the single-flow experiments outperform the multi-flow setups for both payload sizes. For all experiments, we satisfy the upper bounds (in red). Overprovisioning generally does not bring much improvement for the latencies, and in some cases, even worsens the outcomes for delays. However, when comparing the jitters in Figure 11b, we observe that overprovisioning performs better than the default CBS value. Overall, the network modeling using NC is satisfied irrespective of the scenario. When continuing with deeper evaluations, we observe that overprovisioning has slight improvement for jitter and, in some scenarios, latencies.

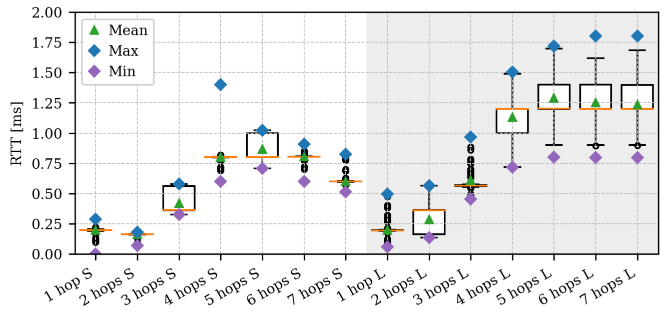
C. *Evaluation of TAPRIO*

For TAPRIO, we focus mainly on the impact of large payload sizes and the number of hops. Starting with the impact of varying payload sizes and fixed seven hops, we observe a stable performance for values below 1193 B. Figure 12a presents boxplot RTT with values around 600 µs for the lower payload sizes. These results are confirmed by the jitter experiments in Figure 12b. Nevertheless, the performance is not as good for the larger payloads. The performance is within 2 ms for latency and 500 µs jitter for 98 % of packets. Depending on the use case, such values might be sufficient or require further analysis. Even though the window size for the PubSub application is wide enough to accommodate larger payloads, it is possible that Linux resolutions cannot cope with it.

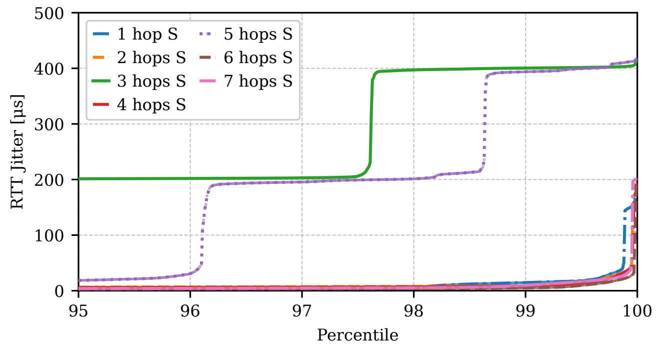
After identifying relevant payloads, we continue with fixing the values of payloads to the largest stable payload (**S**) and 1481 B for **L**. Starting with RTT shown in Figure 13b, we can observe that less hops may yield worse results, as we saw for CBS. Overall, smaller payloads perform better and are more stable than larger ones. This might hint at the fact that unlike CBS, TAPRIO is more sensitive to selecting specific parameters based on the payload size, and finding one configuration for various scenarios is more challenging. This is confirmed also with jitter results in Figure 13a, in which a higher payload yields worse results. The fluctuation confirms that not only the cycle time of the PubSub is missed, but also the TAPRIO windows.

VI. Conclusion

We evaluated OPC UA PubSub using NC and evaluation on COTS HW. Overall, we see that COTS HW and open source solutions can bring stable performance to OPC UA PubSub applications. The NC bounds were met for all of the experiments, including for large numbers of hops and increased payload sizes. Our NC calculations included various CBS behavior, such as under- and overprovisioning, and competing flows, in addition to the PubSub application traffic. Focusing mainly on the CBS qdisc, we observed stable performance for the default values with a slight improvement for overprovisioning by 1 %. These findings hold for single-



(a) RTT delay



(b) 95th percentile of jitter for various numbers of hops

Fig. 13: Comparison of TAPRIO Delay and Jitter for 1-7 hops and payload sizes. **S:** 617 B.

and multi-flow scenarios, where we observed a slight but negligible increase in RTT, considering the CPU isolation and affinity of individual applications in the system. TAPRIO performs better, especially for smaller payload sizes.

In future work, we plan to investigate network modeling for TAPRIO and consider the security of OPC UA PubSub and its implications on the TSN performance.

References

[1] I. E. Commission, "Opc unified architecture," International Electrotechnical Commission, Tech. Rep., 2020.

[2] D. Bruckner *et al.*, "An Introduction to OPC UA TSN for Industrial Communication Systems," *Proceedings of the IEEE*, vol. 107, 2019.

[3] "Ieee standard for local and metropolitan area networks - virtual bridged local area networks amendment 12: Forwarding and queuing enhancements for time-sensitive streams," *IEEE Std 802.1Qav-2009 (Amendment to IEEE Std 802.1Q-2005)*, pp. C1–72, 2010.

[4] M. Bosk *et al.*, "Methodology and infrastructure for tsn-based reproducible network experiments," *IEEE Access*, vol. 10, 2022.

[5] F. Rezabek *et al.*, "EnGINE: Developing a Flexible Research Infrastructure for Reliable and Scalable Intra-Vehicular TSN Networks," in *2021 17th International Conference on Network and Service Management (CNSM)*, Izmir, Turkey, 2021.

[6] F. Rezabek *et al.*, "Engine: Flexible research infrastructure for reliable and scalable time sensitive networks," *Journal of Network and Systems Management*, vol. 30, no. 4, 2022.

[7] E. Kirdan *et al.*, *Real-time performance of opc ua*, 2023. arXiv: 2310.17052 `[cs.NI]`.

[8] A. Gogolev, F. Mendoza, and R. Braun, "Tsn-enabled opc ua in field devices," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2018.

[9] C. Eymüller *et al.*, "Real-time capable opc-ua programs over tsn for distributed industrial control," in *2020 25th IEEE ETFA*, vol. 1, 2020.

[10] 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.

[11] A. Arestova, M. Martin, K.-S. J. Hielscher, and R. German, "A service-oriented real-time communication scheme for autosar adaptive using opc ua and time-sensitive networking," *Sensors*, vol. 21, 2021.

[12] "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems," *IEEE Std 1588-2019*,

[13] "Ieee standard for local and metropolitan area networks – bridges and bridged networks - amendment 25: Enhancements for scheduled traffic," *IEEE Std 802.1Qbv-2015 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, and IEEE Std 802.1Q-2014/Cor 1-2015)*, pp. 1–57, 2016.

[14] M. Bosk *et al.*, "Simulation and Practice: A Hybrid Experimentation Platform for TSN," in *22nd International Federation for Information Processing (IFIP) Networking Conference*, Spain, Jun. 2023.

[15] "ISO/IEC/IEEE International Standard - Information technology – Telecommunications and Information Exchange between Systems – Local and Metropolitan Area Networks – Specific Requirements – Part 1BA: Audio video bridging (AVB) Systems," *ISO/IEC/IEEE 8802-1BA First edition 2016-10-15*, pp. 1–52, 2016.

[16] "IEEE Standard for a Transport Protocol for Time-Sensitive Applications in Bridged Local Area Networks," *IEEE Std 1722-2016 (Revision of IEEE Std 1722-2011)*, pp. 1–233, 2016.

[17] "Ieee standard for local and metropolitan area networks–timing and synchronization for time-sensitive applications," *IEEE Std 802.1AS-2020 (Revision of IEEE Std 802.1AS-2011)*, pp. 1–421, 2020.

[18] R. Cochran, *linuxptp*, Last accessed on 2022-11-26. [Online]. Available: https://sourceforge.net/projects/linuxptp/.

[19] "Ieee standard for local and metropolitan area networks–bridges and bridged networks," *IEEE Std 802.1Q-2022 (Revision of IEEE Std 802.1Q-2018)*, pp. 1–2163, 2022.

[20] N. Mühlbauer, E. Kirdan, M.-O. Pahl, and K. Waedt, "Feature-based comparison of open source opc-ua implementations," 2021.

[21] R. L. Cruz, "A calculus for network delay. I. Network elements in isolation," *IEEE Transactions on information theory*, 1991.

[22] R. L. Cruz, "A calculus for network delay. II. Network analysis," *IEEE Transactions on information theory*, 1991.

[23] J.-Y. Le Boudec and P. Thiran, *Network calculus: a theory of deterministic queuing systems for the internet*. Springer, 2001.

[24] J. A. R. De Azua and M. Boyer, "Complete modelling of AVB in network calculus framework," in *Proceedings of the 22nd International Conference on Real-Time Networks and Systems*, 2014.

[25] A. Eckhardt and S. Müller, "Analysis of the round trip time of opc ua and tsn based peer-to-peer communication," in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2019, pp. 161–167.

[26] Y. Li, J. Jiang, C. Lee, and S. H. Hong, "Practical implementation of an opc ua tsn communication architecture for a manufacturing system," *IEEE Access*, vol. 8, pp. 200 100–200 111, 2020.

[27] M. H. Farzaneh and A. Knoll, "Time-sensitive networking (tsn): An experimental setup," in *2017 IEEE VNC*, 2017, pp. 23–26.

[28] S. Grüner, A. E. Gogolev, and J. Heuschkel, "Towards performance benchmarking of cyclic opc ua pubsub over tsn," in *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2022.

[29] P. Denzler *et al.*, "Timing analysis of tsn-enabled opc ua pubsub," in *2022 IEEE 18th International Conference on Factory Communication Systems*, 2022.

[30] 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.

[31] R. Zippo and G. Stea, "Nancy: An efficient parallel network calculus library," *SoftwareX*, vol. 19, 2022.

[32] F. Rezabek, M. Bosk, G. Carle, and J. Ott, "TSN Experiments Using COTS Hardware and Open-Source Solutions: Lessons Learned," in *IEEE PerCom Workshops*, Atlanta, USA, Mar. 2023.