Timing-Aware Configuration Framework for TSN and OPC UA in Industrial Automation Systems

Kasra Ekrad[1,2], Bjarne Johansson[2], Inés Alvarez Vadillo[2], Saad Mubeen[1], Mohammad Ashjaei[1]
[1]Mälardalen University, Västerås, Sweden
[2]ABB, Västerås, Sweden
{kasra.ekrad, saad.mubeen, mohammad.ashjaei}@mdu.se,
{ines.alvarez-vadillo, bjarne.johansson, kasra.ekrad1}@se.abb.com

***Abstract*— The convergence of OPC UA Publish–Subscribe (PubSub) and Time-Sensitive Networking (TSN) enables vendor-agnostic and predictable communication in real-time distributed industrial automation systems. Deploying such systems, however, requires consistent configuration across application and network layers that currently expose mismatched timing abstractions. TSN depends on precise traffic specifications, while OPC UA PubSub provides extensive configuration options without guidance on selecting timing-related parameters, creating engineering complexity and increasing commissioning effort. To address this challenge, this paper presents TPConf, a tool-supported framework that co-configures OPC UA PubSub and TSN from traffic specifications by classifying application requirements into industrial automation traffic classes, mapping them to TSN classes (ST, AVB, BE), and deriving corresponding PubSub parameters. TPConf further generates ready-to-compile publisher code for open62541, and an industrial use case with diverse simulated flows demonstrates the automated co-configuration workflow. By unifying application and network configuration, TPConf reduces engineering effort and supports predictable and scalable OPC UA over TSN deployments in real-time automation systems.**

***Index Terms*—OPC UA, TSN, PubSub, Network Configuration.**

I. Introduction

Industrial automation ecosystems are undergoing a fundamental transformation driven by an increasing demand for connectivity, flexibility, and interoperability [1]. Instead of isolated control islands, modern systems integrate large numbers of field devices, controllers, supervisory systems, and cloud services into a converged communication infrastructure. This evolution, central to the Industry 4.0 vision and the ongoing IT/OT convergence, requires combining the scalability and throughput of IT networks with the strict real-time constraints of operational technologies [2]. For the real-time systems community, this shift raises a core challenge: how to preserve predictable end-to-end timing in an environment that increasingly resembles large-scale distributed systems.

A converged industrial network must handle heterogeneous devices and traffic flows with diverse temporal and criticality requirements [3]. Ensuring bounded latency, jitter, and reliability across these flows, while still supporting best-effort communication, is essential for maintaining real-time communication guarantees. Time-Sensitive Networking (TSN) has therefore gained significant traction in industry because it enables deterministic and predictable communication over Ethernet through offline scheduling, traffic shaping, and bandwidth reservation mechanisms [4], [5]. TSN traffic classes such as Scheduled Traffic (ST), Audio and Video Bridging (AVB), and Best Effort (BE) provide different levels of timing guarantees that must be carefully matched to the requirements of each application [6], [7].

Industrial users demand vendor-neutral and interoperable application-layer communication. Standards such as the Open Process Automation Standard (OPAS) build on the Open Platform Communications Unified Architecture (OPC UA) [8]. The OPC UA Publish–Subscribe (PubSub) model is particularly relevant because it enables scalable and decoupled data distribution across multiple network technologies, including TSN [9]. Together, OPC UA and TSN have the potential to provide a communication stack suited for real-time industrial automation systems. However, deploying such converged systems requires bridging a key engineering gap. TSN relies on precise traffic specifications such as periodicity, criticality, deadlines, and latency bounds, whereas these timing properties are typically not expressed at the application layer. At the same time, application developers often lack TSN expertise, while network engineers do not work with application-level semantics. This disconnect is a well-recognized barrier to industrial adoption of OPC UA over TSN [9], [10].

Additional configuration complexity arises from the OPC UA PubSub specification [11]. Although the specification exposes many parameters and models, it provides limited guidance on how these parameters should be configured based on timing requirements. Incorrect configuration of PubSub or TSN can lead to violations of end-to-end timing requirements in industrial automation systems. Modern process plants contain thousands of I/O points and hundreds of thousands of data variables distributed across multiple controllers [12], [13]. Manually designing, mapping, and configuring the corresponding communication behavior is labor-intensive, error-prone, and impractical. Industrial stakeholders consistently report that automated and consistent configuration tools are urgently needed to reduce commissioning effort, prevent misconfigurations, and ensure predictable system behavior.

Contribution. This paper addresses the challenge of jointly configuring the OPC UA PubSub application layer and the TSN network layer to achieve predictable communication in

real-time industrial automation systems. We introduce TPConf (TSN PubSub Configuration), a tool-supported framework that maps traffic specifications to TSN and PubSub configurations that meet the specified timing constraints. TPConf also generates ready-to-compile publisher code for the open-source OPC UA implementation open62541 [14]. An industrial use case demonstrates how TPConf automates configuration steps that otherwise demand deep domain expertise. By unifying application and network configuration, TPConf reduces engineering and commissioning effort and supports the urgent industrial need for predictable and scalable OPC UA over TSN deployments.

II. Background and Related Works

As TPConf proposes configurations for both the OPC UA publisher and the TSN traffic class, the following concepts will provide a brief overview of PubSub and its related configuration, along with TSN and its traffic classes.

A. Industrial Traffic Classes

In the context of industrial automation systems, numerous heterogeneous data flows exist that differ in timing characteristics and requirements. In this paper, we only consider period, deadline, jitter requirement, and criticality of the traffic. We define the criticality as the assigned level of assurance with respect to failure that might threaten the timing constraints required by the traffic. Criticality levels defined here are High, Medium, or Low. These traffic characteristics will be used later in the development of the proposed framework.

Unlike traditional fieldbus systems, converged IT/OT networks must simultaneously support hard real-time control traffic and best-effort data flows on a shared communication infrastructure. Treating all these types uniformly will either lead to over-provisioning or violations of these requirements. Thus, a clear distinction between these traffic classes is required. Some standards such as IEEE 802.1Q [5], IEC/IEEE 60802 [15], Industry IoT Consortium (IIC) [16], and 5G Alliance for Connected Industries and Automation (5G-ACIA) [17] provide classification of industrial traffic classes, but none of them conform to a unify set of traffic and same characteristics for the same traffic class.

Although Mateu et al. evaluated the mapping of legacy Ethernet traffic to TSN traffic classes in [18], they considered only a limited set of traffic characteristics and did not differentiate industrial traffic classes. On the other hand, Ekrad et al. in [19] proposed a comprehensive traffic classification. They proposed an alternative approach to classifying industrial automation traffic classes. The work presents comprehensive characteristics of 13 industrial traffic classes.

B. TSN Traffic Classes and Shaping Mechanisms

In most studies on TSN, traffic is typically categorized into three main classes: Scheduled Traffic (ST), Audio Video Bridging (AVB), and Best-Effort (BE) [6], [7], [20]–[22]. The ST class represents traffic that is scheduled offline and transmitted according to a predefined schedule. This class is typically managed by the Time-Aware Shaper (TAS), which controls traffic transmission according to a cyclic schedule defined in the Gate Control List (GCL). The GCL specifies, for each time interval in the cycle, which transmission gates associated with the output queues are open or closed. By opening the gate of the queue carrying ST traffic at predefined time windows, TAS enables deterministic transmission with bounded latency and minimal jitter.

AVB traffic is managed by the Credit-Based Shaper (CBS), which regulates transmission via a credit-based shaper mechanism. In most studies, two AVB classes are considered. These classes are commonly referred to as AVB Class A and AVB Class B, although the standard allows for a configurable number of such classes. Each AVB class is associated with a credit value that increases at a configured rate (known as idleSlope) when the corresponding queue is not transmitting, and decreases at a constant rate (known as sendSlope) during the transmission. An AVB traffic is eligible for transmission only when the credit is non-negative and higher-priority traffic is not interfering the transmission.

On the other hand, BE traffic includes all remaining traffic that do not require any timing guarantees. This traffic is transmitted when no higher-priority traffic is under transmission. BE is not going through any shaping mechanisms and uses the bandwidth when it is free.

It is important to note that multiple traffic priorities may coexist within each TSN class. The classification into ST, AVB, and BE primarily determines the traffic shaping and scheduling mechanisms applied, whereas precedence among traffic flows is resolved through priority-to-queue mapping and gate control at egress ports.

C. OPC UA PubSub and Configurations

OPC UA provides a communication framework that enables an interoperable industrial network [1], [11]. In addition to its traditional client-server model, OPC UA defines a Publish-Subscribe (PubSub) communication model in OPC UA Part 14 [11] that is better suited to scalable and predictable data distribution.

In the client-server model, communication is based on sessions between clients and servers, where data is transferred upon request [9]. This pattern introduces communication and resource overhead for resource-constrained and time-sensitive systems. In contrast, the PubSub model enables decoupled data distribution and is the pattern prescribed by the standard [11] for cyclic real-time communication, such as between a controller and a device. In this pattern, publishers transmit data without explicit knowledge of the subscribers.

In addition to the PubSub communication paradigm, key features of the traditional OPC UA architecture, including its information model capabilities and services such as historical data access and event notification, are functional in PubSub [9]. PubSub introduces multiple abstraction layers between application data and network messages. The components, their related configurations, and the data formats at each

Fig. 1. OPC UA PubSub Message generation steps and components.

stage are further explained, and a high-level representation is provided in Figure 1.

At the application layer, data typically originates from variables selected from the server's address space [11]. Based on the PublishedDataSet configurations, these variables form into DataSetFields (DataSets) by a logical component called the DataSet Collector. PublishedDataSet defines the structure of the data to be transmitted. DataSets are variables that are tagged with metadata, including data type and descriptions.

DataSetWriter component further transforms DataSets into a DataSetMessage [9], [11]. The DataSetMessage also defines the message structure (DataSetMessage Type), identifier, and control properties. DataSetMessage Type defines whether DataSetMessage is transmitted as a KeyFrame, DeltaFrame, Event, KeepAlive, or ChunkMessage. KeyFrames contain a full DataSet being transmitted cyclically, while DeltaFrames transmit only changed values relative to the previous KeyFrame or previous DeltaFrame. Event messages are intended for sporadic or acyclic data. ChunkMessages allow large payloads to be fragmented into multiple NetworkMessages, and finally, KeepAlive messages are being transmitted when there is no update on DeltaFrames or Events.

At the next stage, one or more DataSetMessages are grouped by WriterGroup component according to predefined policies and encapsulated into a NetworkMessage [11]. WriterGroup defines the NetworkMessage's publishing interval and then hands it over to the underlying transport protocol. Network transmission is performed via PubSubConnection, which defines the transport binding. PubSubConnection specifies the transport profile (e.g., UDP/UADP or broker-based JSON), the network address, and the local network interface.

The structure of a NetworkMessage is controlled by configurable content masks, such as the NetworkMessageContentMask and DataSetMessageContentMask [11]. These masks determine which header fields (e.g., PublisherId, WriterGroupId, SequenceNumber, GroupVersion) are included in the message header. Content masks, therefore, directly influence message size, parsing complexity, and predictability.

Regarding the encoding of the traffic, OPC UA PubSub supports two primary message encoding solutions. The encoding solutions include: (i) UA Datagram Protocol (UADP), which is a compact binary format optimized for low overhead and deterministic communication over UDP; and (ii) JSON format, which is a text-based encoding typically used in broker-based or cloud-integrated scenarios.

The PubSub specification exposes a rich set of configuration entities that directly influence the temporal characteristics of data transmission. These configuration entities include DataSetMessage type, publishing interval, encoding solution, transport profile, and DataSet ordering. Although the specification provides detailed definitions of these components and their configuration, selecting an appropriate set of parameters from the large configuration space remains a challenge for system engineers. Furthermore, the PubSub specification does not provide concrete guidance on how configuration parameters should be derived from application-level traffic characteristics. In practice, this lack of guidance often leads to inconsistent configurations, which in turn leads to a significant impact on the real-time behavior of both the application layer (PubSub) and the network layer (TSN network layer).

III. Proposed Co-configuration Framework

The proposed framework supports the co-configuration of OPC UA PubSub publisher parameters and TSN traffic classification. As illustrated in Figure 2, TPConf comprises four main components, including the Specification Identifier, TSN Classifier, PubSub Parameterizer, and Publisher Code Generator, which together configure both the PubSub and TSN.

The configuration process begins with a user-provided input configuration that specifies the traffic application class and its periodicity. This configuration input is processed by the Specification Identifier, which determines the corresponding industrial traffic class and derives detailed specifications associated with that class. These specifications include, but are not limited to, timing requirements such as deadline, jitter tolerance, and criticality level. The identified industrial traffic class and its specifications are then forwarded to both the TSN Classifier and the PubSub Parameterizer.

The TSN Classifier maps the industrial traffic class to an appropriate TSN traffic class (e.g., AVB, ST, or BE). This mapping is critical for TSN network configuration, as it determines the traffic shaping mechanisms at the switch port and impacts the timing guarantees provided by the network. Incorrect classification may lead to violations of real-time constraints and degraded system performance.

In parallel, the PubSub Parameterizer uses the same industrial traffic class to derive suitable OPC UA PubSub configuration parameters. These parameters are stored in an intermediate PubSub configuration file. By deriving both TSN classification and PubSub parameters from the same traffic specifications, TPConf ensures a consistent and coherent co-configuration of the communication stack. Finally, the intermediate PubSub configuration file is passed to the Publisher Code Generator. This module translates the configuration into ready-to-use OPC UA PubSub publisher code written in C and compatible with the open62541 implementation.

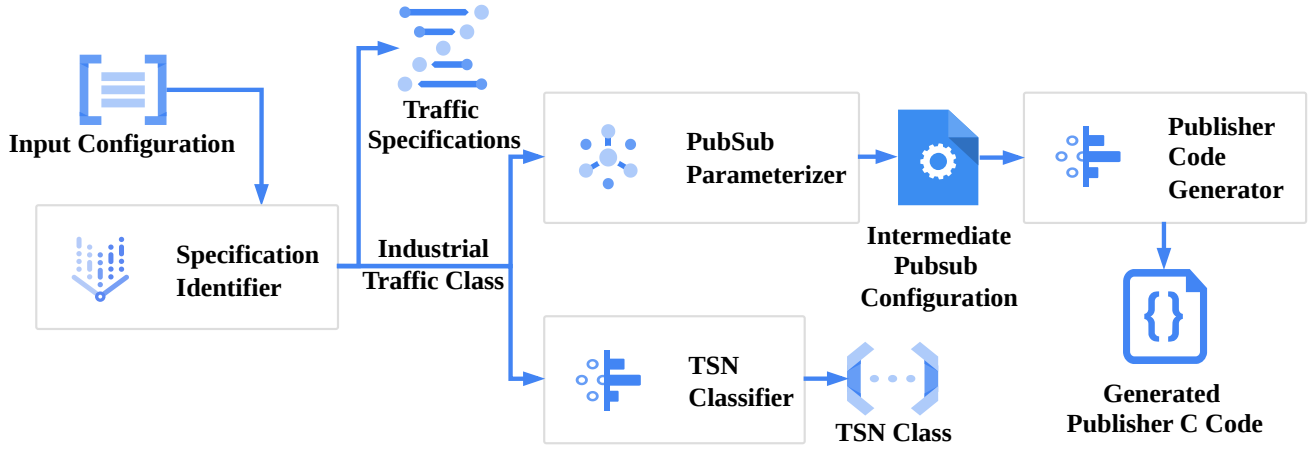

Fig. 2. Architecture of the TPConf framework, illustrating its inputs, outputs, and internal modules. The modules are represented as boxed blocks.

TPConf is designed to support industrial network developers who may only be aware of the traffic application or use case (e.g., control messages, monitoring data, or operator commands) but lack detailed knowledge of traffic specifications, such as jitter constraints or criticality. Manually identifying these characteristics can be impractical and may result in improper TSN traffic classification. To address this challenge, TPConf automatically identifies the industrial traffic class associated with the given application and recommends a complete set of traffic specifications and PubSub Configurations. This way, since both publisher configuration parameters and TSN traffic classes are generated via the same traffic specifications as inputs, the framework ensures a suitable co-configuration.

The outputs of TPConf include:

- the identified industrial traffic class;
- derived traffic specifications;
- mapped TSN traffic class;
- an intermediate PubSub configuration file; and
- generated publisher code compatible with open62541.

The following subsections describe the design and operation of each TPConf component in detail.

A. Specification Identifier

The Specification Identifier module classifies traffic based on two inputs as traffic application class and periodicity. This means that the user should only know what the use case of the data is, and if the data is produced cyclically or event-driven. We used the mapping proposed in [19] for classification. Alongside the 13 industrial traffic classes that were defined in previous work [19], each industrial traffic class corresponds to a traffic application class. The traffic application class represents the traffic use case in industrial automation scenarios. In total, 9 traffic application classes were presented in the mapping. Since some traffic classes share the same application class, we use the traffic's periodicity from the input configuration file to draw a line between these cases. For example, the application class for both Diagnostic-cycle and Diagnostic-acycle is "Diagnostics and Monitoring", and only their periodicity can distinguish between these two traffic classes.

The Specification Identifier module exports four traffic specifications alongside a unique industrial traffic class as specified in Table I. Four specifications are *Periodicity*, *Criticality*, *Deadline*, and *Jitter* requirement. *Periodicity* determines if the traffic is cyclic or event-based, while *Deadline* defines if the traffic has any deadline or latency constraint. For the *Jitter* parameter, 0 corresponds to no jitter constraint, 1 to a low jitter constraint, and 2 to a zero or close to zero jitter constraint. Here, three levels of *Criticality* as High, Medium, and Low are considered for each traffic class. The module returns a unique industrial traffic class (column 2 in Table I) along with the traffic specifications. This identifier is used to retrieve and present a set of traffic specifications to the user in the form of an output file. In this table, columns 1-2 are the inputs from the configuration file, while columns 2-6 are the outputs of the module.

B. TSN Classification

The TSN Classification module classifies traffic into one of the three TSN classes: ST, AVB, or BE. For traffic classification, the module uses a set of parameters including periodicity, deadline, jitter, and criticality values received from the Specification Identifier module. The classification logic is based on the methodology proposed in [19] that maps each industrial traffic class into a TSN traffic class.

The classification logic is presented in Algorithm 1. All parameters are treated as Boolean variables. Based on the user-defined inputs described in Section III-A, the following abstractions are used: periodicity is denoted by P, deadline or bounded latency requirement by D, jitter requirement by J, and high criticality by HRT. Since the input file allows three jitter-related options, values 1 and 2 are mapped to $J = 1$, while 0 is mapped to $J = 0$. Furthermore, from the three criticality levels presented in Table I as High, Medium, and

TABLE I

APPLICATION CLASS TO INDUSTRIAL TRAFFIC CLASS MAPPING. YES AND NO FOR PERIODICITY AND DEADLINE MEANS THAT THE CORRESPONDING TRAFFIC DOES HAVE THE CHARACTERISTIC AND REQUIREMENT. FOR THE PARAMETER JITTER, THE VALUES 0, 1, AND 2 REPRESENT NO JITTER, LOW JITTER, AND ZERO JITTER REQUIREMENTS RESPECTIVELY. PERIOD.=PERIODICITY, AND CRITI.=CRITICALITY

Input		**Output**			
Traffic Application Class	**Period.**	**Industrial Traffic Class**	**Criti.**	**Deadline**	**Jitter**
Open/Closed (Distributed) Control Loop and Motion Control	Yes	Control-iso	High	Yes	2
Open/Closed (Distributed) Control Loop and Motion Control	Yes	Control-sync	High	Yes	1
Factory Automation and Non Synchronized Control	Yes	Control-async	High	Yes	1
Control Events and Alarms	No	Event	High	Yes	0
Voice/Video	Yes	Voice/Video	Low	Yes	0
Network Control and Inter-network Control	Yes	Network	High	No	0
Operator Commands and HMI Interactions	Yes	Command-cyclic	Medium	Yes	0
Operator Commands and HMI Interactions	No	Command-acycle	Medium	Yes	0
Configuration and Management	No	Config	Medium	Yes	0
Diagnostics and Monitoring	Yes	Diagnostic-cycle	Medium	Yes	0
Diagnostics and Monitoring	No	Diagnostic-acycle	Medium	Yes	0
Best Effort	No	Best-effort	Low	No	0

Algorithm 1 Industrial automation traffic mapping to the corresponding TSN Class algorithm.

function TSN_CLASSIFICATION(Traffic Specifications)
  Input: P ▷ Periodicity as Boolean
  Input: D ▷ Deadline constraint as Boolean
  Input: J ▷ Jitter constraint as Boolean
  Input: HRT ▷ High Criticality as Boolean
  Output: $primary, alternative$
    ▷ Recommended primary and alternative TSN class(es)
  $ST \leftarrow P \wedge (J \vee D)$
  $AVB \leftarrow D \wedge \neg(J \wedge HRT)$
  if $ST \wedge AVB$ **then**
    $primary \leftarrow AVB$
    $alternative \leftarrow ST$
  else if ST **then**
    $primary \leftarrow ST$
  else if AVB **then**
    $primary \leftarrow AVB$
  else
    $primary \leftarrow BE$
  end if
  return $(primary, alternative)$
end function

Low, High is mapped to HRT = 1, Medium and Low are mapped to HRT = 0.

In some traffic scenarios described in [19], both TSN classes of AVB and ST are recommended for the given traffic type. In such cases, AVB is defined as the primary recommendation, while ST is considered a feasible alternative under certain scenarios. These scenarios include (1) if tighter latency guarantees are required, (2) if AVB resources are occupied, or (3) if the traffic remains schedulable within the ST class. In this case, ST becomes an acceptable alternative. Here, we presented them as primary and alternative classes.

The output of the TSN classification module is therefore a recommended TSN class, optionally with an alternative class. This recommendation supports design-time TSN network configuration. In converged OPC UA TSN deployments, this output is crucial to ensure that application-layer traffic characteristics are consistently reflected in the network-layer scheduling.

C. PubSub Parameterization

After identifying the traffic class by the Specification Identifier module, the system uses a parameterization function to propose PubSub configurations. Some of the configurations are based on a study presented in [23], which recommends suitable publisher configurations derived from specific traffic specifications. Other configurations proposed by the framework were selected to satisfy the requirement to construct a publisher compatible with the open62541 implementation. PubSub Parameterizer maps each traffic class to a structured output object. This object structures PubSub recommendation as a set of configurations. The set of main parameters that the PubSub Parameterizer module decides is as follows:

- whether the DataSet is cyclic;
- DataSetMessage types, chosen from KeyFrame, DeltaFrame, Event, ChunkMessage;
- whether DeltaFrames are enabled and the corresponding KeyFrameCount interpretation;
- WriterGroup's publishing interval derived from the traffic cycle time;
- a keep-alive parameter, indicating whether a keep-alive configuration is required;
- a DataSet ordering options from Undefined, AscendingWriterID, AscendingWriterIDSingle;
- encoding formats (UADP and/or JSON); and
- transport profiles (UDP/UADP or broker-based JSON).

1) PubSub Parameterization Logic: As presented in Table II, for simplicity, we categorized the traffic classes into four groups that share a similar pubsub configuration. Groups are identified with Control, Cyclic command/diagnostics, Voice/Video, and Event-based tags. The header in this table serves as the identifier for the PubSub configuration

TABLE II
PUBSUB TRAFFIC GROUPS BASED ON PUBSUB CONFIGURATION SIMILARITIES. CYCLIC CMD./DIAG.= CYCLIC COMMAND/DIAGNOSTICS

Control	Cyclic Cmd./Diag.	Voice/Video	Event-based
Control-Iso	Command-Cycle	Voice	Event
Control-Sync	Diagnostic-Cycle	Video	Config
Control-Async	Best-Effort		Command-Acycle
			Diagnostic-Acycle
			Best-Effort

summary given in Table III. The Best-Effort traffic class is considered part of both the Event-based and the Cyclic Command/Diagnostics groups. According to standards [5], [16], [17] and as recommended in [19], the Best-Effort traffic is categorized as sporadic traffic. However, there are traffic types in PubSub that are cyclic and expose specifications similar to the Cyclic Command/Diagnostics group. The best-effort class in the Cyclic Command/Diagnostics group is low-critical cyclic traffic that has no jitter or latency requirements. But with the same configuration in PubSub, similar to the Cyclic Command/Diagnostics group. This categorization does not contradict the Best-effort traffic class in TSN.

Table III shows the configurations assigned by PubSub Parameterizer for each traffic class. The configuration presented here is the configuration set proposed in [23]. The differences in the Voice/Video class from the Control group are the use of ChunkMessages as the DataSetMessage Type and Undefined value for the DataSet ordering parameter. In the Event-based group, only the Event traffic class can utilize AscendingWriterIDSingle as DataSet ordering. The value of KeyFrameCount would be (> 1) if DeltaFrame is the choice, otherwise the value is 1.

2) Intermediate PubSub Configuration: After determining the appropriate PubSub configuration, the PubSub Parameterizer translates the selected parameters into PubSub–compliant parameters and generates a `.ini` configuration file. This file is subsequently processed by the Publisher Code Generator module to produce the final executable implementation. The rationale for decoupling the configuration file from the Code Generator module is that this configuration can serve as input to other backend components. Various components can receive PubSub configurations and use them to configure other devices, such as subscribers and other network management nodes that are dependent on the data structures published by each publisher.

The generated PubSub configuration file comprises three main sections:

- identified Industrial Traffic Class;
- TSN Class together with its associated flags (P, D, J, and HRT); and
- parameterized PubSub configuration parameters.

In addition to the parameters derived from the Parameterization logic, the configuration file includes default PubSub component identifiers (such as connection name, publisher ID, WriterGroup ID, and DataSetWriter ID) as well as a template DataSet field definition with example values. These placeholders later help the user to determine environment-compatible values while keeping the code generated by the publisher generator complete. These parameters were defined based on examples given in open62541.

At this stage, certain configurations may require explicit user decisions. The exact value of the publishing interval parameter should be defined, although the framework recommends a range based on the industrial traffic class. For traffic classes where DeltaFrame and KeyFrame transmission are recommended, the user must decide which to use. If DeltaFrame is selected, the user must additionally specify the KeyFrameCount parameter. For traffic types that require KeepAlive messages, the user must define an appropriate KeepAliveTime value. Furthermore, since multiple encoding and transport are supported, the user must select between UADP and JSON encodings. If JSON is chosen, a subsequent decision between MQTT and AMQP broker variants is required. These revisions to the DataSetMessage Type and message encoding are resolved either interactively in the framework during configuration generation, or the user can later review the generated publisher configuration and manually change the parameter values.

Dataset ordering options are defined in the specification, but it is not implemented in open62541. In a normal situation, pubsub behaves as an undefined option. The only parameter implemented in open62541 that we can use to simulate a condition similar to AscendingWriterID is maxEncapsulated-DataSetMessageCount. This parameter enables us to send each DatasetMessage as a separate NetworkMessage by setting it to 1. However, we do not get the ascending feature with this setting. For the undefined option, this parameter would be set to the maximum. Currently, simulating AscendingWriterIDSingle is not possible.

Finally, before writing the configurations to the PubSub output configuration file, some parameters will be translated to values compatible with open62541. For example, for defining cyclic and even-based messages, other than the cyclic DataSet parameter, the PublishedDataSetType should be set to UA_PUBSUB_DATASET_PUBLISHEDITEMS for cyclic dataset or UA_PUBSUB_DATASET_PUBLISHEDEvent for event messages. Currently, the open62541 does not support Event publishing. Further, the following values are translated for encoding and transport of the messages:

- transport of pubsub-udp-uadp and encoding of UADP-WRITERGROUPMESSAGEDATATYPE;
- transport of pubsub-mqtt-json and encoding of JSON-WRITERGROUPMESSAGEDATATYPE; and
- transport of pubsub-amqp-json and encoding of JSON-WRITERGROUPMESSAGEDATATYPE.

TABLE III
SUMMARIZING PUBSUB PARAMETER CONFIGURATION DERIVED FROM THE PARAMETERIZATION LOGIC.

PubSub Traffic Group	**Control**	**Voice/Video**	**Event-based**	**Cyclic Cmd./Diag.**
DataSetMessage Type	KeyFrame	ChunkMessage	Event	KeyFrame or DeltaFrame
KeyFrameCount	1	1	1	1 or > 1
Cyclic DataSet	Yes	Yes	No	Yes
DeltaFrame	Disabled	Disabled	Disabled	Enable if DeltaFrame used
KeepAlive	Not required	Not required	Required	Required if DeltaFrame used
Encoding & Transport	UADP over UDP	UADP over UDP	UADP over UDP or JSON	UADP over UDP or JSON
DataSet Ordering	AscendingWriterID AscendingWriterIDSingle	Undefined	Undefined	Undefined

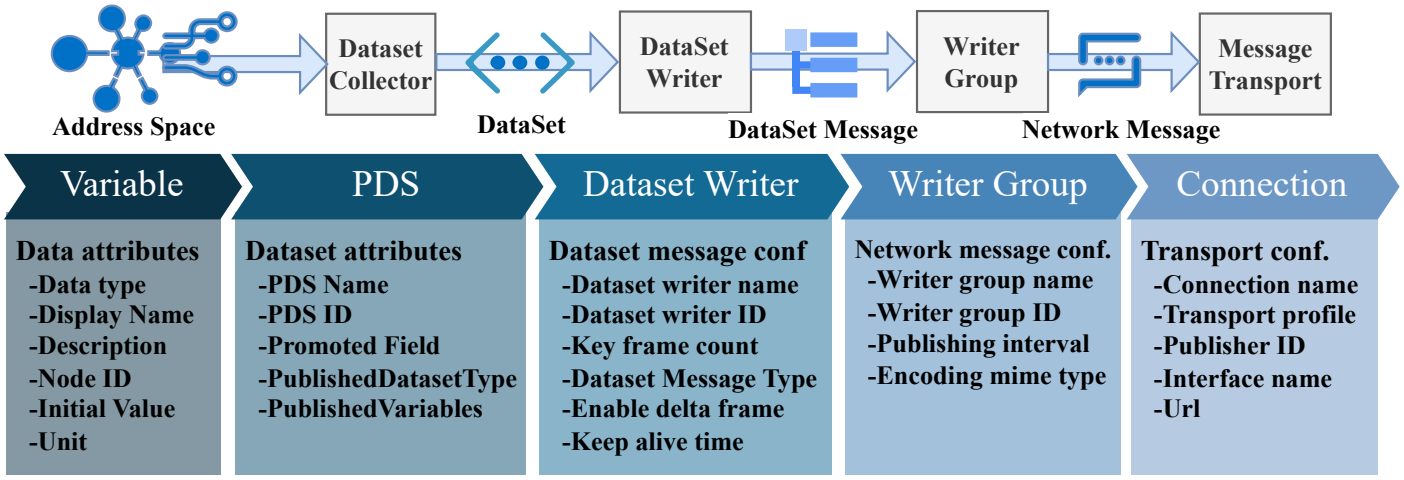

Fig. 3. Publisher Code Generation module parameters categorized by their respective Open62541 implementation.

D. Publsiher Code Generation

To enable reproducible deployments of the OPC UA PubSub publisher, we implemented a lightweight code-generation module that translates the PubSub Configuration file generated by the PubSub Parameterizer module into a C implementation, similar to the publisher examples and templates in open62541 [14]. Again, this module uses a configuration-driven approach to generate the ready-to-build publisher code. The Generator reads the key-value pairs and maps them into in-memory variables. To prevent the build issues caused by incomplete input configurations, it initializes the parameters with default values. For the default configuration, it assumes a UADP encoded cyclic message that transmits as KeyFrames. IDs, component names, and DataSet fields are assumed to be random as they were assigned as a template in the PubSub configuration file.

After parsing the configurations, the Generator constructs the PubSub configuration in the same order as the open62541 templates:

- creation of PubSub variable as provided in the configuration file;
- construction of PublishedDataSet and DataSet bindings;
- creation of PubSubConnection and network-related configurations; and
- instantiation of WriterGroup and DataSetWriter with user-defined message setting and values.

The PubSub specifications define a large set of headers, flags, and settings for PubSub messages. To reduce the complexity of header design, we followed the specification's recommended headers for UADP and JSON messages in this work. The specification distinguishes between fixed and dynamic message layouts. A fixed layout is typically used for periodic, cyclic traffic where the structure and size of the NetworkMessage remain constant over time. In contrast, dynamic layouts are intended for event-driven or traffic which the message structure may vary depending on the content being transmitted. Such as a variable number of DataSetMessages in one NetworkMessage. In this work, fixed message layout corresponds to time-critical periodic traffic that are jitter-sensitive. Such as Control-iso and Control-sync. As determinism is a requirement for these traffic types, a stable header configuration and constant message size should be enforced. A fixed layout minimizes encoding time on the publisher side and parsing ambiguity at the subscriber side [11]. Furthermore, it allows for predictable encoding and decoding times, which is beneficial for time-sensitive networks.

To be able to apply the specification recommendations, we need to configure the NetworkMessageContentMask and

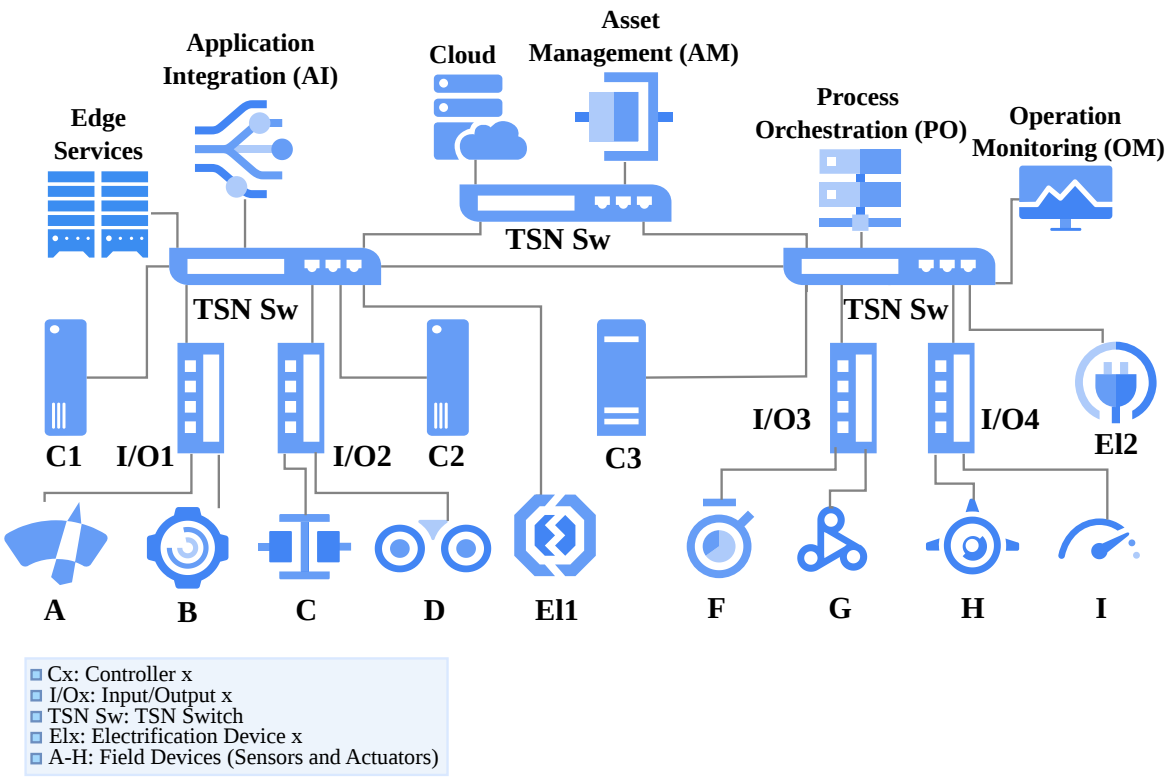

Fig. 4. Converged TSN-based industrial automation use case integrating field devices, I/Os, electrification systems, distributed controllers, and higher-level operational services in an interoperable OPC UA-enabled network.

related content masks at the WriterGroup and DataSetWriter levels. These masks determine which header fields are included in each transmitted NetworkMessage. In the current implementation, the following header fields are enabled for UADP messages: PublisherId; GroupHeader; GroupVersion; WriterGroupId; SequenceNumber. For dynamic layout messages (all other traffic classes except the Control classes), to accommodate variable data structures, Static fields are omitted, and payload-related headers are adjusted. The following fields were enabled for this group: Version and Extended Flags, PublisherId, and PayloadHeader. PayloadHeader is a field that affects encoding speed and message size, and is added for dynamic layout traffic.

In Figure 3, all parameters that the module configures and those that the module uses as defaults can be seen.

IV. Use Case Evaluation

The use case, depicted in Figure 4, showcases a part of the automation industry interconnections. This use case is adopted from the flexible, interoperable, modular, and secure architecture proposed in [24], which is compatible with the open automation paradigm. The use case illustrates a converged OPC UA TSN industrial automation architecture where distributed controllers (C1-C3), I/O modules (I/O1-I/O4), heterogeneous field devices (A-H), and electrification devices (E1-E2) are interconnected via three TSN Switches (TSN S). Higher-level services such as asset management, process orchestration, operation monitoring, application integration, and edge/cloud services share the same scalable network backbone.

The field devices represent process sensors and actuators that interact directly with I/O modules. The I/O devices directly transmit physical field signals to and from the network. The electrification devices are intelligent power distribution units that transmit electrical statistics to the network. The controllers execute real-time control logic based on data received from I/O modules. These controllers communicate with each other and with higher-level control and orchestration modules. TSN-enabled switches provide synchronization and deterministic transport by shaping the traffic to the network. The edge provides localized, intensive processing service, while the cloud supports data analytics and historical data management. Application integration connects industrial data to the enterprise systems and enables interoperability among applications. Operational modules such as system monitoring, asset management, and process orchestration provide supervision and coordination of the processes.

Table IV presents a set of heterogeneous traffic flows derived from the industrial automation use case in Figure 4. The presented flows are samples of traffic flows among devices and do not constitute a complete set. These flows are provided to illustrate the process of the main TPConf modules.

TABLE IV

DIVERSE TRAFFIC FLOWS IN THE INDUSTRIAL USE CASE. PERIOD: 0 OR 1 REFERS TO PERIODIC OR APERIODIC. DEADLINE: 0 OR 1 REFERS TO NO DEADLINE CONSTRAINT OR DEADLINE CONSTRAINT. JITTER: 2, 1, 0 REFERS TO ZERO JITTER, LOW JITTER, AND NO JITTER REQUIREMENT. CRIT: 1 AND 2 REFER TO HIGH AND MEDIUM/LOW CRITICALITY.

Flow	Source	Destination	Use Case	Period.	Deadline	Jitter	Crit.
F1	I/O1 (A,B)	C1	Control Loop	1	1	2	1
F2	C1	I/O1 (A,B)	Actuation	1	1	1	1
F3	I/O2 (C,D)	C2	Measurements	1	1	0	1
F4	I/O4 (G,H)	C3	Diagnostics	1	1	0	0
F5	I/O2 (C,D)	OM	Trend Data	1	1	0	0
F6	I/O3 (A,B)	C3	Alarm	0	1	0	1
F7	Cn	Cn	Event Alarm	0	1	0	1
F8	Cn	Cn	Status Sync	1	1	1	1
F9	E1	C2	Measurments	1	1	0	1
F10	E2	OM	Power Status	0	1	0	0
F11	C2	E1	Setpoint	0	1	0	1
F12	C2	E2	Config Update	0	1	0	0
F13	Cn	OM	HMI Update	1	1	0	0
F14	OM	Cn	Operator CMD	0	1	0	0
F15	OM	AI	Status Export	1	0	0	0
F16	AI	AM	Data Feed	0	0	0	0
F17	Cn	AI	Status Export	1	0	0	0
F18	AI	Cloud	Historian	1	0	0	0
F19	Cloud	AI	Optimization	0	0	0	0
F20	Edge	Cn	Process Data	1	1	1	1
F21	Edge	Cloud	Batch Upload	1	0	0	0
F22	Cn	PO	Orchestration	0	1	0	0
F23	PO	Cn	Recipe Sync	0	0	0	0

TABLE V

DERIVED INDUSTRIAL TRAFFIC CLASS, TSN CLASS, AND PUBSUB CONFIGURATION CATEGORY FOR THE SIMULATED FLOWS IN TABLE IV. PUBSUB CATEGORIES FOLLOW THE GROUPING IN TABLE II. ST IN PARENTHESES REPRESENTS THE ST AS AN ALTERNATIVE RECOMMENDATION TO AVB CLASS.

Flow	Industrial Traffic Class	TSN Class	PubSub category
F1	Control-Iso	ST	Control
F2	Control-Sync	ST	Control
F3	Control-Async	AVB (ST)	Control
F4	Diagnostic-Cycle	AVB (ST)	Cyc. Cmd./Diag.
F5	Diagnostic-Cycle	AVB (ST)	Cyc. Cmd./Diag.
F6	Event	AVB	Event-based
F7	Event	AVB	Event-based
F8	Control-Sync	ST	Control
F9	Control-Async	AVB (ST)	Control
F10	Diagnostic-Acycle	AVB	Cyc. Cmd./Diag.
F11	Event	AVB	Event-based
F12	Config	AVB	Event-based
F13	Command-Cycle	AVB	Cyc. Cmd./Diag.
F14	Command-Acycle	AVB	Event-based
F15	Best-Effort	BE	Cyc. Cmd./Diag.
F16	Best-Effort	BE	Event-based
F17	Best-Effort	BE	Cyc. Cmd./Diag.
F18	Best-Effort	BE	Cyc. Cmd./Diag.
F19	Best-Effort	BE	Event-based
F20	Control-Sync	ST	Control
F21	Best-Effort	BE	Cyc. Cmd./Diag.
F22	Config	AVB	Event-based
F23	Best-Effort	BE	Event-based

The simulated flows cover a broad range of industrial communication patterns. Flows F1–F3, F8, and F20 represent time-critical control traffic with strict deadlines and jitter requirements. Event-driven alarms and commands (e.g., F6, F7, F11, F14, F22) are modeled as aperiodic flows with bounded latency but no jitter constraints. Monitoring and diagnostic data (e.g., F4, F5, F10, F13) are cyclic but less critical. Higher-level integration traffic, including historian uploads, optimization feedback, batch transfers, and cloud synchronization (F15–F19, F21, F23), is modeled with no timing requirements, which represent non-deterministic or best-effort communication. Cn in this table represents the group of all the controllers as C1, C2, and C3. The column Use Case can be easily mapped to the application class in Table I and forms the input parameter to the Specification Identification module alongside the periodicity parameter.

The results of this module are the traffic specification presented in Table IV and the traffic industrial traffic class presented in Table V for each traffic flow. The traffic specification, including Periodicity, Deadline, Jitter, and Criticality, will be handed to the TSN Classification module, and the result is the TSN Class. The PubSub Parameterization utilizes the resulting Industrial Traffic Class to configure the pubsub based on the PubSub Traffic Group presented in Table III. The resulting pubsub configuration will be consumed by the PubSub Generator module and generate the PubSub ready to compile publisher code[1].

The tool-supported framework can be configured to generate a bulk configuration by providing different traffic application classes and their respective periodicity in the input configuration file. In this case, the framework assumes default values for decisions that should be made by the user. It generates incremental values for component IDs and names. In this case, the user should revise the output publisher configuration file if the default values are to be changed. Regarding the encoding, the framework defaults to UADP and does not consider Delta frames. In the provided scenario, we also ran the use case in bulk mode.

V. CONCLUSION

This paper addressed the challenge of co-configuration of OPC UA PubSub and TSN in converged industrial automation systems. While TSN provides deterministic networking mechanisms and PubSub enables vendor-neutral data distribution,

[1]The presented use case, all of its exported files and codes are provided in the GitHub repository of the project with the ID of kasra-ekrad/OPCUA-TSN.

addressing the gap between application-defined traffic and network layer configuration remains an engineering challenge. Furthermore, manual configuration of such systems is error-prone and inefficient while scaling to large industrial systems with thousands of variables with heterogeneous timing requirements.

To address this issue, we introduced TPConf as an integrated tool-supported configuration framework that configures application and network layers, considering the traffic requirements and specifications. TPConf derives industrial traffic classes and provides vide range of traffic specifications. At this point, by utilizing a set of periodicity, deadline constraints, jitter tolerance, and criticality of the traffic under evaluation, the framework maps them to appropriate TSN traffic classes (ST, AVB, BE). Further, it recommends PubSub publisher configurations. This intermediate TSN and PubSub configuration can serve as an input to other engineering tools that are affected by the application and network configuration. Finally, TPConf utilizes PubSub configuration to generate ready-to-compile publisher code compatible with the open62541 stack.

Through an industrial use case with diverse traffic flows, we demonstrated how TPConf automates configuration steps that would otherwise require cross-domain expertise and rigorous manual commissioning in a large-scale industrial use case. Overall, the evaluation shows that in heterogeneous industrial scenarios, deriving both PubSub and TSN configurations from the same traffic specification reduces cross-layer inconsistencies and engineering ambiguity in converged OPC UA–TSN deployments. The current implementation focuses on publisher-side configuration and supports only a single DataSet per NetworkMessage. While the configuration can be extended to enable code generation for multi-DataSet scenarios, subscriber configurations and multi-publisher setups are not yet supported and remain areas for future development.

ACKNOWLEDGMENT

This work is supported by the Swedish Knowledge Foundation (KKS) via the SEINE and MARC projects and by the Swedish Governmental Agency for Innovation Systems (VINNOVA) via the FLEXATION and iSecure projects.

REFERENCES

[1] D. Bruckner, M.-P. Stănică, 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.

[2] M. Ashjaei, A. Bucaioni, and S. Mubeen, "A conceptual framework for software modeling of automation systems," in *ITNG 2022 19th International Conference on Information Technology-New Generations*. Springer International Publishing, 2022.

[3] D. Bruckner, R. A. Blair, M.-P. Stănică, A. Ademaj, W. M. Skeffington, D. Kutscher, S. Schriegel, R. Wilmes, K. Wachswender, L. Leurs, M. G. Seewald, R. Hummen, and S. Ravikumar, "A new solution for industrial communication," 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:201823666

[4] 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)*, 2019.

[5] IEEE, "IEEE Std 802.1Q-2022: IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks," 2022, iEEE Standard.

[6] J. Sasiain, D. Franco, A. Atutxa, J. Astorga, and E. Jacob, "Toward the integration and convergence between 5g and TSN technologies and architectures for industrial communications: A survey," *IEEE Communications Surveys & Tutorials*, vol. 27, no. 1, pp. 259–321, 2025.

[7] Z. Satka, M. Ashjaei, H. Fotouhi, M. Daneshtalab, M. Sjödin, and S. Mubeen, "QoS-MAN: A novel QoS mapping algorithm for TSN-5g flows," in *2022 IEEE 28th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2022.

[8] OPC Foundation, *OPC Unified Architecture Specification*, OPC Foundation, 2025.

[9] 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, no. 7, 2021.

[10] H. Trifonov and D. Heffernan, "OPC UA TSN: a next-generation network for industry 4.0 and IIoT," *International Journal of Pervasive Computing and Communications*, vol. 19, no. 3, pp. 386–411, 2021.

[11] OPC Foundation, "Opc unified architecture part 14: Pubsub," Available: https://www.opcfoundation.org/UA/Part14/, 2024.

[12] H. Koziolek, A. Burger, P. P. Abdulla, J. Rückert, S. Sonar, and P. Rodriguez, "Dynamic updates of virtual PLCs deployed as kubernetes microservices," in *Software Architecture*, S. Biffl, E. Navarro, W. Löwe, M. Sirjani, R. Mirandola, and D. Weyns, Eds. Springer, 2021, vol. 12857, pp. 3–19.

[13] H. Koziolek, A. Burger, and A. Puthan Peedikayil, "Fast state transfer for updates and live migration of industrial controller runtimes in container orchestration systems," *Journal of Systems and Software*, vol. 211, p. 112004, 2024.

[14] open62541 Project, "open62541 - open source opc ua implementation," https://open62541.org, 2025.

[15] IEC/IEEE, "IEC/IEEE 60802 TSN Profile for Industrial Automation," https://1.ieee802.org/tsn/iec-ieee-60802/, 2024.

[16] Industry IoT Consortium, "Time sensitive networks for flexible manufacturing testbed characterization and mapping of converged traffic types," https://www.iiconsortium.org/pdf/IIC_TSN_Testbed_Char_Mapping_of_Converged_Traffic_Types_Whitepaper_20180328.pdf, 2019.

[17] 5G Alliance for Connected Industries and Automation (5G-ACIA), "Integration of 5g with time-sensitive networking for industrial communications," Available: https://5g-acia.org/whitepapers/integration-of-5g-with-time-sensitive-networking-for-industrial-communications, 2021.

[18] D. B. Mateu, M. Ashjaei, A. V. Papadopoulos, J. Proenza, and T. Nolte, "LETRA: Mapping legacy ethernet-based traffic into tsn traffic classes," in *IEEE International Conference on Emerging Technologies and Factory Automation*, 2021.

[19] K. Ekrad, I. A. Vadillo, B. Johansson, S. Mubeen, and M. Ashjaei, "A methodology to map industrial automation traffic to tsn traffic classes," in *2025 IEEE 30th International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2025.

[20] Y. Song, C. Guo, P. Xu, L. Li, and R. Zhang, "Research on routing and scheduling algorithms for the simultaneous transmission of diverse data streaming services on the industrial internet," *SCIENTIFIC REPORTS*, vol. 11, no. 1, p. 18351, 2021.

[21] M. Barzegaran and P. Pop, "Communication scheduling for control performance in TSN-based fog computing platforms," *IEEE Access*, vol. 9, pp. 50 782–50 797, 2021.

[22] M. Jover, M. Barranco, I. Álvarez, and J. Proenza, "Migrating legacy ethernet-based traffic with spatial redundancy to TSN networks," in *2022 IEEE International Conference on Emerging Technologies and Factory Automation*, 2022.

[23] K. Ekrad, B. Johansson, I. A. Vadillo, S. Mubeen, and M. Ashjaei, "Traffic-aware configuration of opc ua pubsub in industrial automation networks," in *2026 IEEE International Conference on Industrial Technology (ICIT)*. IEEE, 2026.

[24] ABB, "The DCS of tomorrow: ABB's process automation system vision White paper," ABB, Tech. Rep., 2022. [Online]. Available: https://new.abb.com/control-systems/control-systems/envisioning-the-future-of-process-automation-systems/automation-system-whitepaper