Deployment of Real-Time Containers in TSN-based Distributed Embedded Systems

Muhammad Waqas*, Per Strandberg†, Johan Furunäs Åkesson†, Saad Mubeen*, Mohammad Ashjaei*
*Mälardalen University, Västerås, Sweden
{muhammad.waqas, saad.mubeen, mohammad.ashjaei}@mdu.se
†Westermo Network Technologies AB, Västerås, Sweden
{per.strandberg, johan.furunasakesson}@westermo.com

***Abstract*—Container-based deployment is increasingly adopted in distributed industrial systems to support modular and flexible software integration; however, ensuring timing correctness in such systems depends on both container execution and predictable network communication. While real-time container platforms provide processor isolation and Time-Sensitive Networking (TSN) enables predictable and deterministic communication, the end-to-end timing behavior of distributed real-time applications remains strongly influenced by deployment decisions such as container placement and inter-container communication. This paper addresses this challenge by proposing a timing-aware deployment approach for distributed real-time containers over TSN-enabled infrastructure. The deployment problem is formulated as a mixed-integer linear program that jointly determines container placement and communication under processor capacity, placement admissibility, and network constraints. The candidate deployments are subsequently analyzed using CONAN-TSN to verify timing feasibility and deadline satisfaction.**

***Index Terms*—Real-time, Containers, Orchestration, TSN.**

I. Introduction

Industrial systems are transitioning to modular and distributed architectures that facilitate dynamic reconfiguration and flexible operation [1]. Consequently, such systems must enable rapid adaptation to changing production requirements and support the integration of components across distributed architectures. To address these requirements, industrial system architectures are shifting from rigid, hardware-centric solutions toward software-centric solutions, where system functionality is realized through software components deployed across distributed nodes [2]. While this transition enables scalability, reusability, and flexibility, it also introduces new challenges related to the structuring, deployment, and management of these systems.

To address these challenges, software containers have emerged as a key enabling technology, providing a lightweight and portable mechanism for deploying and managing software components independently of the underlying hardware [3]. Compared to traditional virtual machines, the reduced overhead of containerization makes this technology particularly suitable for industrial deployments, supporting use cases such as Programmable Logic Controller (PLC) virtualization and using these virtual PLCs for distributed real-time control [4], [5]. As industrial systems continue to evolve toward distributed architectures, the number of containerized applications communicating across networked nodes is expected to grow correspondingly.

In such containerized and distributed settings, satisfying end-to-end timing requirements depends not only on the execution of tasks within individual containers, but also on the communication between distributed nodes that host the containers. In this work, end-to-end refers to the timing behavior of both task execution in containers and message transmission in the network where deadlines must be satisfied at each level. At the same time, industrial networks typically carry mixed traffic, where time-critical communication must coexist with non-critical data flows on a shared network infrastructure [6]. This combination of stringent timing requirements and traffic heterogeneity necessitates communication technologies that provide timing predictable behavior while supporting traffic coexistence. Time-Sensitive Networking (TSN) [7] addresses these requirements by extending Ethernet with mechanisms such as traffic shaping and scheduling, enabling real-time and best-effort traffic to coexist on the same network while providing bounded latency for time-critical traffic.

Despite these advances, the combination of software containers and Time-Sensitive Networking (TSN) is insufficient to ensure the timing correctness of distributed real-time applications. While containerization provides a flexible execution environment and TSN enables predictable and deterministic communication, the end-to-end timing behavior of such applications remains highly dependent on how software components are deployed across computational nodes and how their communication flows are routed through the network. These decisions directly influence processor utilization and communication interference, and ultimately determine whether the specified timing requirements can be satisfied. Consequently, the core challenge in this regard is to determine container deployments that preserve timing requirements while jointly accounting for computational and communication constraints. This challenge is not explicitly addressed by mainstream deployment platforms, such as Kubernetes [8], which are primarily designed for general-purpose and best-effort workloads.

To address this gap, this paper proposes a timing-aware deployment approach for distributed real-time containers over TSN-enabled infrastructure. The paper provides the following main contributions.

- A timing-aware deployment workflow for distributed

real-time containerized applications over TSN-enabled industrial networks.
- An optimization-based formulation of container placement and inter-container communication, where deployments are constrained by processor capacity and network bandwidth, and subsequently analyzed for network schedulability.

II. Background and Related work

This section provides background on TSN networks and related work on real-time container deployment.

A. Background

TSN is a set of standards that provide mechanisms for predictable and deterministic communication over Ethernet [9]. TSN supports different traffic types, namely Scheduled Traffic (ST), Audio Video Bridging (AVB), and Best-effort (BE). ST is scheduled offline and transmitted with Time-aware Shaper (TAS) that controls the opening and closing of queue gates according to a predefined schedule. Thus, ST experiences a limited interference from other types of traffic. AVB traffic is shaped using the Credit-Based Shaper (CBS), which assigns credit to each AVB queue. Without the shaper, lower priority queues may experience long delays because the transmission is mainly decided by priority. CBS addresses this by controlling the bandwidth usage of each AVB queue through a credit mechanism. The credit increases when an AVB message is waiting for transmission or when the credit is negative, and decreases when the messages from that queue are transmitted. The rate at which the credit is replenished is called idleSlope, whereas the rate at which it is consumed during transmission is called sendSlope. An AVB queue is eligible for transmission when its credit is zero or positive [10]. In this work, we focus on AVB traffic and restrict it to the two stream reservation classes, Class A and Class B, because the deployment model targets class-based bandwidth-aware placement and routing. In this setting, each message is assigned to an AVB class, and feasibility depends on whether the selected routes have sufficient class-specific CBS bandwidth reservation.

B. Related work

There are several works that have focused on container-based virtualization for real-time systems. Queiroz et al. [11] provide a systematic review of container-based virtualization for industrial real-time systems covering latency, container platforms, orchestration, and open challenges. Existing work on real-time containers has mainly focused on enabling predictable execution on individual nodes. A common direction is to combine container-based deployment with real-time Linux, such as PREEMPT_RT, to reduce kernel latency for real-time workloads [12], [13]. Hierarchical Constant Bandwidth Server (HCBS) further supports temporal isolation by enforcing processor reservations at the cgroup level, allowing real-time containers a guaranteed execution time [14].

Struhar et al. [15] focus on computational orchestration, using HCBS-managed reservations to support processor allocation and runtime adaptation. Similarly, Fiori et al. [16] and Samimi et al. [17] extend container orchestration support for real-time workloads by enabling reservation-based execution of real-time containers. These works mainly focus on predictable container execution, CPU isolation, and computational resource management on nodes, while network constraints are not treated as part of the container deployment decision.

A complementary line of work by Garbugli et al. [18] addresses deterministic communication for containerized applications by providing a deterministic Kubernetes overlay network based on TSN and kernel bypassing techniques. However, they focus on the communication data path rather than deciding container placement or flow routing. Furthermore, Walker et al. [19] integrate container deployment with TSN configuration and allow custom schedulers to be incorporated. However, their focus is architectural integration, their scheduler is illustrative rather than a general placement and routing method, and their TSN configuration is centered only around time-triggered traffic.

Another body of work has studied the routing and schedule synthesis for TSN traffic, focused on scheduled traffic. Schweissguth et al. [20] and Hellmanns et al. [21] propose Integer Linear Programming-based formulations for routing and scheduling of TSN streams. Berisa et al. [22] study AVB-aware scheduling, where scheduled-traffic schedules are constructed while preserving the schedulability of AVB traffic. These studies mainly configure the network with the assumption of fixed end points. Chahed and Kassler [23] study a closely related problem by jointly optimizing TSN task placement, routing, and TAS-based schedule synthesis in virtualized edge-compute platforms. Their work focuses on scheduled traffic and network schedule generation, with node-level resources handled abstractly through candidate servers rather than explicit processor-capacity constraints.

In contrast, the work presented in this paper focuses on the deployment decision problem for distributed real-time containers. We formulate joint container placement and communication routing as a resource-constrained optimization problem that accounts for node-level processor capacity, communication bandwidth, and various timing requirements. The generated deployments are then evaluated through a separate timing feasibility analysis stage.

III. System Model

The system, denoted by $\mathcal{S}$, consists of two or more computing nodes and a TSN network. The system can be formally represented as follows.

$$\mathcal{S} := \langle \mathcal{N}, \mathcal{L}, \Pi, \mathcal{M} \rangle$$

where $\mathcal{N}$ is the set of computing nodes, and $\mathcal{L}$ is the set of links in the TSN network. The term Π represents the set of software containers hosted on the nodes, and $\mathcal{M}$ is the set of messages exchanged between containers.

A. Computing Node Model

The set of nodes, $\mathcal{N}$, represents a set of uniprocessor computing platforms that can host containers and is defined as follows.

$$\mathcal{N} := \{ \langle n_a, \mathrm{Cap}_a \rangle \mid a = 1, \ldots, |\mathcal{N}| \}$$

where each tuple $\langle n_a, \mathrm{Cap}_a \rangle$ represents a computing node n_a and its associated processing capacity Cap_a. The capacity Cap_a represents the amount of processor resource that can be allocated to container reservations on node n_a. During deployment, the sum of the processor demands of the containers assigned to a node must not exceed this capacity. Nodes may have different processing capacities, which allows the model to represent heterogeneous deployment platforms.

B. Container Model

The set of containers that execute tasks in the system is defined as follows.

$$\Pi := \{ \langle \pi_i, Q_i, P_i, A_i \rangle \mid i = 1, \ldots, |\Pi| \}$$

For a container $\pi_i \in \Pi$, Q_i denotes its reserved execution budget, P_i denotes the budget replenishment period, and A_i denotes the admissible node set of π_i. The admissible node set specifies the nodes on which the container is allowed to execute and is defined as follows.

$$A_i := \{ n_a \mid \langle n_a, \mathrm{Cap}_a \rangle \in \mathcal{N} \wedge \pi_i \text{ can be deployed on } n_a \}$$

The admissible node set captures the deployment restrictions. For example, a container that collects data from a physical sensor may need to be deployed on a node connected to that sensor, and therefore cannot be placed on every node in the system.

Containers may host either real-time or best-effort tasks. For real-time containers, the pair (Q_i, P_i) represents the reservation interface provided by the HCBS scheduling model [14]. In hierarchical scheduling, deriving these parameters is known as server design problem [15] and is outside the scope of this work. We assume that these values are given for every container. The internal workload of each real-time container is assumed to be designed and schedulable under its assigned reservation interface, such as in [15]. Therefore, the task model is not required for this paper.

C. Network Model

The network is modeled through a set of directed links $\mathcal{L}$. Each link $l \in \mathcal{L}$ represents a unidirectional communication link in the network topology. We assume that the network features TSN standards, hence we define the links as TSN links in the model. Following the TSN link model, a physical full-duplex connection is represented by two directed links, one in each direction. Thus, transmission and reception over the same physical connection are treated independently. A link may connect a node to a TSN switch, two TSN switches, or a TSN switch to a node. In this model, two nodes may have multiple paths to connect them in the network. A set of possible paths between nodes n_a and n_b is denoted by the set $\mathcal{R}_{a,b} = \{\rho_k\}$, where ρ_k is a path of links between nodes n_a and n_b.

In addition to the network topology, the network contains messages according to TSN standards. In this model, we assume that the containers transmit and receive only AVB messages, hence ST traffic class is outside the scope of this work. Let $\mathcal{X}$ denote the set of AVB classes considered in the system. For each link $l \in \mathcal{L}$ and AVB class $X \in \mathcal{X}$, the CBS queue is associated with an idleSlope $\alpha^+_{X,l}$ and a sendSlope $\alpha^-_{X,l}$. The idleSlope $\alpha^+_{X,l}$ denotes the credit replenishment rate of class X on link l. It specifies the bandwidth budget assigned to AVB class X on link l. Since $\alpha^+_{X,l}$ is defined per link and per AVB class, different AVB classes can be assigned different bandwidth on the same link.

The messages exchanged between containers are modeled by $\mathcal{M}$. Each message represents a communication between source and destination container. The set is defined as:

$$\mathcal{M} := \{ \langle m_j, C_j, \mathrm{Src}_j, \mathrm{Dst}_j, T_j, D_j, X_j \rangle \mid j = 1, \ldots, |\mathcal{M}| \}.$$

For a message m_j, C_j denotes the worst-case transmission time of the message. Moreover, Src_j and Dst_j denote the source and destination containers of message m_j, respectively. The source and destination nodes corresponding to m_j are not fixed in the message model, they are determined by the placement of its source and destination containers. Furthermore, D_j denotes the relative deadline, T_j denotes the message period, and $X_j \in \mathcal{X}$ denotes the AVB class assigned to message m_j. The message parameters are provided as a system input. Note that, when two containers are deployed within the same node, the messages are internal that are not modeled in this paper.

IV. Container deployment method

This section formulates the problem of deploying real-time containers on a TSN-based distributed embedded systems, followed by a design-time method to address it.

A. Problem formulation

The problem addressed in this work is the offline deployment of containerized real-time applications over a TSN network. The formulation considers a static deployment inputs, where the set of containers, admissible nodes, and message properties are known at design time. Dynamic changes to containers and message set changes are outside the scope of this work.

When communicating containers are deployed on different nodes, the deployment must determine both container placement and the routing of inter-container communication over the TSN network. We focus on real-time containers with explicit timing requirements and reserved processing resources. Best-effort (BE) containers are not considered, which can be deployed subsequently using the remaining processor capacity on each node, without any timing guarantees.

The placement part of the problem determines the mapping of containers to computational nodes. This mapping is constrained by the admissible placement set defined for each container, as well as by the available processing capacity of each node. In particular, a container can only be deployed on nodes within its admissible set, and the aggregate processor demand assigned to any node must not exceed its capacity. Moreover the deployment must ensure that the timing requirements of all containers are satisfied.

The routing part of the problem considers messages exchanged between containers that are deployed on different nodes. For a message m_j, let Src_j and Dst_j denote its source and destination containers, respectively. If these containers are assigned to nodes n_a and n_b respectively, the message must be routed along one of the candidate paths $\rho_k \in \mathcal{R}_{a,b}$ connecting these nodes. In contrast, if both containers are deployed on the same node, no network routing is required for the corresponding message.

The deployment must also satisfy the network resource constraints associated with routed messages. For each network link and AVB traffic class, the aggregate bandwidth demand of all messages traversing that link must not exceed the bandwidth allocated to that class. However, satisfying bandwidth constraints alone does not guarantee that messages meet their deadlines, as message latency is also influenced by TSN mechanisms, traffic interference, and the impact of multi-hop transmission. Therefore, bandwidth constraints are used to ensure network resource feasibility, while separate timing analysis is required to verify that the messages meet their corresponding deadlines.

The offline deployment problem is thus formulated as a joint container placement and message routing problem. Its objective is to produce a deployment that assigns all real-time containers to admissible nodes and selects routing paths between communicating containers such that the timing requirements of both containers and messages are satisfied.

B. Deployment method

In order to address this problem, we propose a multi-stage deployment method. The first stage takes the system model as input. Second, the resource-allocation part of the problem is formulated as a mixed-integer linear program (MILP). The MILP finds candidate deployments that satisfy the processor capacity, placement admissibility, routing, and AVB bandwidth constraints. Third, these candidate deployments are subsequently evaluated using the CONAN-TSN toolchain [24] to verify whether message deadlines are met. The toolchain supports several functions, including traffic mapping and network analysis. In this work, however, we focus exclusively on the timing analysis for AVB traffic to assess whether the selected routing paths guarantee that the messages' deadlines are met. Finally, a deployment is accepted only if it satisfies both the MILP constraints and the subsequent timing feasibility check. If no candidate deployment satisfies the timing feasibility check, the deployment is considered infeasible. The overall workflow is shown in Fig. 1.

At the container level, (Q_i, P_i) is assumed to provide a valid reservation for the internal workload of a container. This means that the reservation parameters are assumed to be correctly specified such that all tasks executing within the container meet their timing requirements. The formulation therefore does not check task-level schedulability, but only ensures that the total reserved processor capacity does not exceed the capacity of each node. This capacity check is necessary because the node shall not commit more execution time than it can serve.

The placement and routing problem is combinatorial because each real-time container must be assigned to a node and each inter-node message must be assigned to one candidate path. Even if routing is ignored, the problem reduces to assigning containers with processor demands to nodes with limited capacity, which resembles a bin packing problem and is NP-hard [25]. Therefore, the MILP is used to solve the resource allocation part of the problem, while detailed timing analysis is handled outside the optimization model. The optimization objective is to obtain a feasible candidate solution and avoid concentrating computation or communication load on a small set of nodes and links. For compactness, in the formulation below, n_a, π_i, and m_j refer to the node, container, and message components of their corresponding tuples in $\mathcal{N}$, Π, and $\mathcal{M}$, respectively.

For each real-time container $\pi_i \in \Pi$, let d_i denote its processor demand. This demand is derived from the reservation interface (Q_i, P_i), i.e., $d_i := \frac{Q_i}{P_i}$. The model uses this demand during placement decisions.

For each message m_j, let r_j denote its bandwidth demand. r_j is calculated from the message size divided by its period. It can also be derived from the worst-case transmission time of the message and the link speed.

The MILP ensures that, for each directed link l and AVB class X, the total bandwidth demand of messages of class X routed over link l does not exceed the assigned idleSlope $\alpha^{+}_{X,l}$.

We denote by Ω the set of node pairs for which at least one candidate path exists, defined as follows.

$$\Omega := \{(n_a, n_b) \mid \langle n_a, \mathrm{Cap}_a \rangle \in \mathcal{N},\ \langle n_b, \mathrm{Cap}_b \rangle \in \mathcal{N},\ n_a \neq n_b,\ \mathcal{R}_{a,b} \neq \emptyset\}. \tag{1}$$

Similarly, $\bar{\Omega}$ denotes the set of node pairs for which no candidate path exists.

$$\bar{\Omega} := \{(n_a, n_b) \mid \langle n_a, \mathrm{Cap}_a \rangle \in \mathcal{N},\ \langle n_b, \mathrm{Cap}_b \rangle \in \mathcal{N},\ n_a \neq n_b,\ \mathcal{R}_{a,b} = \emptyset\}. \tag{2}$$

The model uses two binary decision variables. The placement variable $x_{i,a}$ is equal to one if container π_i is assigned to node n_a, and zero otherwise. For nodes $n_a \notin A_i$, the placement variable is fixed to zero. Similarly, the routing variable $z_{j,a,b,k}$ is equal to one if message m_j is routed from node n_a to node n_b over candidate path $\rho_k \in \mathcal{R}_{a,b}$, and zero otherwise. Both variables are binary:

$$x_{i,a},\ z_{j,a,b,k} \in \{0, 1\}. \tag{3}$$

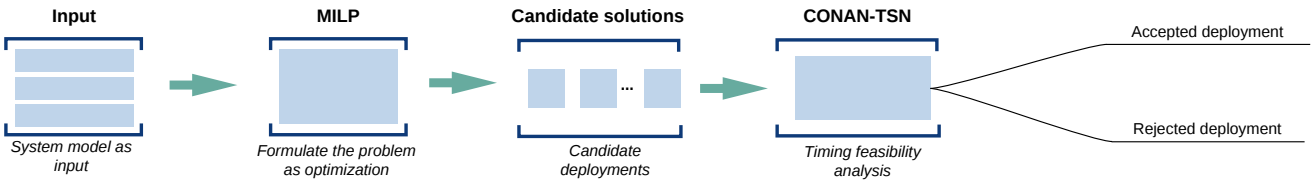

Fig. 1. Offline placement workflow.

Placement feasibility: Each container must be assigned to exactly one admissible node:

$$\sum_{n_a \in A_i} x_{i,a} = 1, \qquad \forall \pi_i \in \Pi. \tag{4}$$

The total processor demand of the containers placed on a node must not exceed its capacity:

$$\sum_{\pi_i \in \Pi} d_i x_{i,a} \leq \mathrm{Cap}_a, \qquad \forall \langle n_a, \mathrm{Cap}_a \rangle \in \mathcal{N}. \tag{5}$$

Placement-routing consistency: Routing is required only for messages whose source and destination containers are placed on different nodes. For every node pair $(n_a, n_b) \in \Omega$, the following constraints ensure that a candidate path is selected only when the source container of message m_j is placed on node n_a and its destination container is placed on node n_b:

$$\sum_{\rho_k \in \mathcal{R}_{a,b}} z_{j,a,b,k} \leq x_{\mathrm{Src}_j,a}, \qquad \forall m_j \in \mathcal{M},\ \forall (n_a, n_b) \in \Omega. \tag{6}$$

$$\sum_{\rho_k \in \mathcal{R}_{a,b}} z_{j,a,b,k} \leq x_{\mathrm{Dst}_j,b}, \qquad \forall m_j \in \mathcal{M},\ \forall (n_a, n_b) \in \Omega. \tag{7}$$

$$\sum_{\rho_k \in \mathcal{R}_{a,b}} z_{j,a,b,k} \geq x_{\mathrm{Src}_j,a} + x_{\mathrm{Dst}_j,b} - 1, \qquad \forall m_j \in \mathcal{M},\ \forall (n_a, n_b) \in \Omega. \tag{8}$$

Together, Constraints (6)-(8) force the candidate path selection sum to be one if the source and destination containers of m_j are placed on n_a and n_b, and zero otherwise. If no candidate path exists between two nodes, the model forbids placing the source and destination containers of the message on that node pair:

$$x_{\mathrm{Src}_j,a} + x_{\mathrm{Dst}_j,b} \leq 1, \qquad \forall m_j \in \mathcal{M},\ \forall (n_a, n_b) \in \bar{\Omega}. \tag{9}$$

Communication capacity feasibility: For each link and AVB class, the aggregate bandwidth of all messages routed over that link must not exceed the available bandwidth of that class:

$$\sum_{\substack{m_j \in \mathcal{M} \\ X_j = X}} \sum_{(n_a,n_b) \in \Omega} \sum_{\substack{\rho_k \in \mathcal{R}_{a,b} \\ l \in \rho_k}} r_j z_{j,a,b,k} \leq \alpha^{+}_{X,l}, \quad \forall l \in \mathcal{L},\ \forall X \in \mathcal{X}. \tag{10}$$

The offline optimization problem is therefore formulated as:

$$\min\ \lambda_u \delta_u + \lambda_b \delta_b + \lambda_h H, \tag{11}$$

subject to the placement, placement-routing consistency, and communication constraints above.

In Eq. (11), δ_u denotes the normalized processor-load imbalance across nodes. It is computed from the absolute deviation between the processor utilization of each node and the average processor utilization of the system. Similarly, δ_b denotes the normalized bandwidth-load imbalance across network links. It is computed from the absolute deviation between the bandwidth utilization of each link and the average bandwidth utilization of the system. These terms help balance the processor load across nodes and the bandwidth load across network links. For each candidate path $\rho_k \in \mathcal{R}_{a,b}$, h_{ρ_k} denotes its hop count, derived from the number of links in that path. It is computed by summing the hop count h_{ρ_k} of each selected candidate path, followed by normalization. Limiting the hop count is important because placing communicating containers closer together can reduce communication delay. The weights λ_u, λ_b, and λ_h control the relative importance of processor-load balancing, bandwidth-load balancing, and short-path routing, respectively.

The absolute-deviation terms used in δ_u and δ_b are implemented with auxiliary variables, while H is linear by construction. Therefore, the optimization model remains linear.

V. Implementation

The implementation of the offline placement method is currently in progress. The MILP model is implemented using Gurobi [26] with API `gurobipy` version 13.0.1 in Python. The prototype takes the system model as input, including the nodes, messages between containers, admissible node sets, and candidate paths, and constructs the placement and routing variables together with the constraints described in Section IV-B.

After solving the model, the prototype extracts the selected container placements and routes from the decision variables. These candidate deployments are then intended to be passed to CONAN-TSN for network-timing feasibility analysis. However, this integration is still under development. At this stage, the implementation is used to validate that the proposed formulation can be instantiated and solved for concrete system descriptions.

VI. Conclusion and Future Work

This paper presented an offline placement and routing selection workflow for distributed real-time containers deployed over TSN. The proposed approach formulates the deployment

problem as an MILP with accounting for processor availability, network timing requirements, and placement admissibility. The generated candidate deployments are then intended to be evaluated using CONAN-TSN to ensure schedulability.

The offline placement of containers is part of a broader timing-aware container orchestration framework. We plan to complete the implementation of the proposed workflow. As future work, we aim to extend the proposed workflow with an online deployment manager that can reorganize existing deployments when new containers are introduced to the system, ensuring that container deadlines and communication between containers are respected.

References

[1] J. Morgan, M. Halton, Y. Qiao, and J. G. Breslin, "Industry 4.0 smart reconfigurable manufacturing machines," *Journal of Manufacturing Systems*, vol. 59, pp. 481–506, 2021.

[2] R. Neumann, M. Walker, M. Neubauer, and A. Verl, "Towards uniform and consistent data modelling of resources in distributed industrial control systems," *Procedia CIRP*, vol. 138, pp. 304–309, 2026.

[3] V. Struhár, M. Behnam, M. Ashjaei, and A. Papadopoulos, "Real-time containers: A survey," in *Workshop on Fog Computing and the Internet of Things*, 2020.

[4] T. Goldschmidt, S. Hauck-Stattelmann, S. Malakuti, and S. Grüner, "Container-based architecture for flexible industrial control applications," *Journal of Systems Architecture*, vol. 84, pp. 28–36, 2018.

[5] M. Sollfrank, F. Loch, S. Denteneer, and B. Vogel-Heuser, "Evaluating docker for lightweight virtualization of distributed and time-sensitive applications in industrial automation," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 5, pp. 3566–3576, 2021.

[6] T. Zhang, G. Wang, C. Xue, J. Wang, M. Nixon, and S. Han, "Time-sensitive networking (tsn) for industrial automation: Current advances and future directions," *ACM Comput. Surv.*, vol. 57, no. 2, 2024.

[7] "IEEE TSN task group." http://www.ieee802.org/1/pages/tsn.html.

[8] Kubernetes, "Kubernetes," https://kubernetes.io/, accessed: 2026-04-20.

[9] M. Ashjaei, L. L. Bello, M. Daneshtalab, G. Patti, S. Saponara, and S. Mubeen, "Time-sensitive networking in automotive embedded systems: State of the art and research opportunities," *Journal of Systems Architecture, 2021*, vol. 121, pp. 1–47, 2021.

[10] D. Bujosa, J. Proenza, A. V. Papadopoulos, T. Nolte, and M. Ashjaei, "An improved worst-case response time analysis for avb traffic in time-sensitive networks," in *IEEE Real-Time Systems Symposium (RTSS)*, 2024, pp. 135–147.

[11] R. Queiroz, T. Cruz, J. Mendes, P. Sousa, and P. Simões, "Container-based virtualization for real-time industrial systems—a systematic review," *ACM Comput. Surv.*, vol. 56, no. 3, 2023.

[12] N. Ben Kebaier, B. Geib, E. Lyczkowski, R. Venanzi, and M. Barth, "Real-time performance evaluation of containerized virtual plcs: A comparative study of docker and podman for industrial automation," in *IEEE CAMAD*, 2025.

[13] Linux Foundation, "PREEMPT_RT Versions," https://wiki.linuxfoundation.org/realtime/preempt_rt_versions, accessed: 2026-02-18.

[14] L. Abeni, A. Balsini, and T. Cucinotta, "Container-based real-time scheduling in the linux kernel," *SIGBED Rev.*, vol. 16, no. 3, p. 33–38, 2019.

[15] V. Struhár, S. Craciunas, M. Ashjaei, M. Behnam, and A. Papadopoulos, "Hierarchical resource orchestration framework for real-time containers," *ACM Transactions on Embedded Computing Systems*, vol. 23, no. 1, pp. 1–24, January 2024.

[16] S. Fiori, L. Abeni, and T. Cucinotta, "Rt-kubernetes: containerized real-time cloud computing," in *ACM/SIGAPP Symposium on Applied Computing*, ser. SAC '22. Association for Computing Machinery, 2022, p. 36–39.

[17] N. Samimi, L. Abeni, D. Casini, M. Marinoni, T. Basten, M. Nasri, M. Geilen, and A. Biondi, "Enabling Containerisation of Distributed Applications with Real-Time Constraints," in *37th Euromicro Conference on Real-Time Systems (ECRTS)*, 2025.

[18] A. Garbugli, L. Rosa, A. Bujari, and L. Foschini, "Kubernetsn: a deterministic overlay network for time-sensitive containerized environments," in *IEEE International Conference on Communications (ICC)*, 2023.

[19] M. Walker, N. Imhoff, R. Neumann, M. Neubauer, A. Lechler, and A. Verl, "Methodology for the combined orchestration of real-time containers and time-sensitive network," *Procedia CIRP*, vol. 138, pp. 292–297, 2026.

[20] E. Schweissguth, P. Danielis, D. Timmermann, H. Parzyjegla, and G. Mühl, "ILP-based joint routing and scheduling for time-triggered networks," in *Proceedings of the 25th International Conference on Real-Time Networks and Systems*. New York, NY, USA: Association for Computing Machinery, 2017.

[21] D. Hellmanns, L. Haug, M. Hildebrand, F. Dürr, S. Kehrer, and R. Hummen, "How to Optimize Joint Routing and Scheduling Models for TSN Using Integer Linear Programming," in *Proceedings of the 29th International Conference on Real-Time Networks and Systems*, 2021.

[22] A. Berisa, L. Zhao, S. S. Craciunas, M. Ashjaei, S. Mubeen, M. Daneshtalab, and M. Sjödin, "AVB-aware Routing and Scheduling for Critical Traffic in Time-sensitive Networks with Preemption," in *Proceedings of the 30th International Conference on Real-Time Networks and Systems*. Association for Computing Machinery, 2022.

[23] H. Chahed and A. Kassler, "Optimizing tsn routing, scheduling, and task placement in virtualized edge-compute platforms," *2024 27th Conference on Innovation in Clouds, Internet and Networks (ICIN)*, 2024.

[24] D. B. Mateu, M. Ashjaei, and S. Mubeen, "CONAN-TSN: An Integrated Toolchain for CONfiguration and ANalysis of TSN Networks," in *30th IEEE International Conference on Emerging Technologies and Factory Automation*, 2025.

[25] M. Delorme, M. Iori, and S. Martello, "Bin packing and cutting stock problems: Mathematical models and exact algorithms," *European Journal of Operational Research*, vol. 255, no. 1, pp. 1–20, 2016.

[26] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2026. [Online]. Available: https://www.gurobi.com