Analysis of TSN Time-Aware Shapers Using Schedule Abstraction Graphs

Srinidhi Srinivasan
Eindhoven University of Technology, The Netherlands

Geoffrey Nelissen
Eindhoven University of Technology, The Netherlands

Reinder J. Bril
Eindhoven University of Technology, The Netherlands
Mälardalen University, Västeras, Sweden

Nirvana Meratnia
Eindhoven University of Technology, The Netherlands

Abstract

IEEE Time-Sensitive Networking (TSN) is one of the main solutions considered by the industry to support time-sensitive communication in data-intensive safety-critical and mission-critical applications such as autonomous driving and smart manufacturing. IEEE TSN standardizes several mechanisms to support real-time traffic on Ethernet networks. *Time-Aware Shapers* (TAS) (IEEE 802.1Qbv) is the standardized mechanisms of TSN that is usually considered to provide the most deterministic behavior for packet forwarding. TAS regulates when traffic classes may forward incoming packets to the egress of a TSN switch using gates that are opened and closed according to a time-triggered schedule.

State-of-the-art solutions to configure or analyze TAS do not allow for multiple traffic classes to have their TAS gates opened at the same time according to any arbitrary schedule. In this paper, we present the first response-time analysis for traffic shaped with TAS where no restriction is enforced on the gate schedule. The proposed analysis is *exact*. It is a non-trivial variant of the schedule abstraction graph analysis framework [18]. Experiments confirm the usefulness of the proposed analysis and show that it is promising for doing design-space exploration where non-conventional TAS gates configurations are investigated to, for instance, improve average-case performance without degrading the worst-case.

2012 ACM Subject Classification Computer systems organization → Real-time systems; Networks → Network protocols

Keywords and phrases TSN, Time-Aware Shapers, TAS, SAG, Schedule Abstraction, latency

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2024.16

Supplementary Material
Software (Source Code): `https://github.com/orgs/SAG-org/repositories`

1 Introduction

The amount of information exchanged between interconnected subsystems in distributed safety-critical and mission-critical systems is rapidly increasing. Applications such as autonomous driving, autonomous robots in automated warehouses and advanced heads-up displays in aircraft, are all integrating complex, sometimes AI-based techniques that require manipulation of large amount of data in the form of audio/video streams, radar and lidar cloud-points and other sensory inputs.

Due to the critical nature of these systems, they must respect end-to-end timing constraints on the typical operational sequence of the data acquisition, data fusion, analysis and decision making and actuation steps. Since these operations may not be performed on the same subsystem, the system timing requirements extend to the data exchange via communication networks interconnecting the subsystems.

Time-sensitive Networking (TSN) is a set of standards [2] developed by the IEEE 802.1 TSN task group [1] that aims to provide the necessary functionality for real-time traffic in Ethernet networks. These standards define various mechanisms to help meet timing requirements on communication flows, thus improving the time-determinism of Ethernet. To ensure bounded low latency, TSN introduces the concept of traffic shaping [3, 9]. Traffic shaping refers to the timely distribution of the bandwidth among the packets of various communication flows forwarded through TSN switches. The three prominent traffic shaping mechanisms in TSN are *Time-Aware Shapers* (TAS) (802.1Qbv), *Credit-Based Shapers* (CBS) (802.1Qav), and *Asynchronous Traffic Shapers* (ATS) (802.1Qcr). TAS use gates that open and close according to a time-triggered schedule to shape the time intervals in which different traffic classes may transmit packets through an egress port of the TSN switch. CBS and ATS limit traffic burstiness and interference between different streams and traffic classes using variations of the leaky bucket mechanism. Thanks to its time-triggered approach, TAS is considered to be the shaper providing the highest degree of determinism among all the shapers discussed in the TSN standards. For this reason, TAS is usually assumed to shape the transmission of control traffic and high-priority traffic with the most stringent timing requirements, whilst CBS and ATS are more appropriate for traffic with more dynamic, less deterministic properties (e.g, with sporadic packet arrivals or with large runtime variations on the packet payload). In this paper, we focus on the analysis of TAS, with a more flexible view than traditionally assumed.

To ensure that the latency of time-sensitive communication flows is always within a desirable range, one must determine the worst-case and best-case latency of each packet transmitted through each switch in the network. Many types of worst-case analyses exist to calculate safe upper bounds on the actual worst-case latency of the packets. Network Calculus [26, 8, 31], Compositional Performance Analysis (CPA) [10, 11], eligibility interval [7], trajectory approach [16] or more conventional response time analyses [6, 5] are many different techniques used to analyze CBS. TAS was analyzed in [29, 30] using Network Calculus and CPA. All solutions mentioned so far provide only sufficient analyses, meaning that they may overestimate the worst-case delay experienced by packets.

Most existing work, including some of the examples provided in the annexes of the TSN standards, assume higher priority traffic classes to be shaped by TAS, while medium priority traffic is generally subject to credit-based shaping [13, 14]. State-of-the-art solutions that either generate or analyze TAS schedules also usually assume that TAS-shaped classes have their gates open in non-overlapping time intervals to avoid inter-class interference [29, 25, 27, 17]. The drawback of having such a strict schedule for higher priority classes is that it wastes resources. Lower priority classes are blocked during pre-defined time intervals to let higher priority classes transmit, but higher priority classes may not always need to transmit during the entirety of those time intervals and thus may not utilize the entire bandwidth allocated to them. This means that transmission of some lower-priority packets are intentionally prevented even when the switch egress is available, thereby wasting resources and time. To the best of our knowledge, the only solution providing an analysis for systems with overlapping TAS schedules of different classes (i.e., where lower and higher priority classes may have their gates open at the same time) is [30]. [30] however restricts the gate schedule of each class to have only one opening and one closing event that is periodically repeated. It does not allow for arbitrary gate schedules. Furthermore, it is based on Network Calculus that may be pessimistic and thus provides only a sufficient analysis.

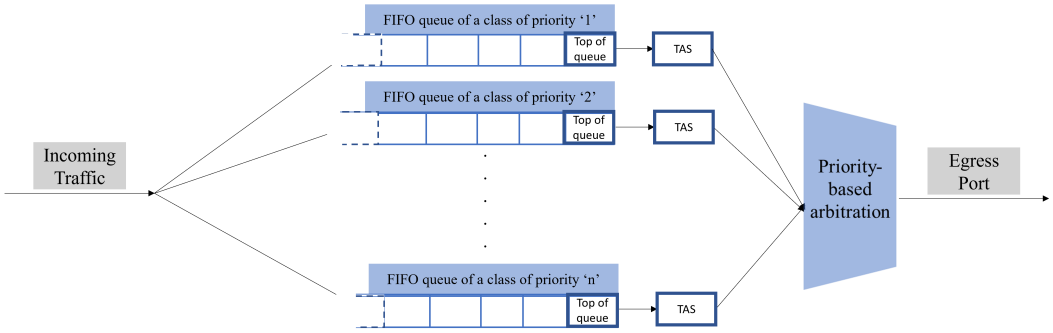


Figure 1 TSN switch with FIFO queues and TAS gates.

In this paper, we propose a new latency analysis for TAS that allows for arbitrary TAS gates schedules, thereby not imposing any restriction on how many gates may be open simultaneously or when gates open or according to which pattern. Therefore, our analysis allows for the introduction of flexible configurations in conventionally rigid use of TAS, and analysis of more generic cases that are compliant with the description of TAS in the TSN standard but were not analyzable so far. We build our analysis using a methodology inspired by the Schedule Abstraction Graph (SAG) framework presented in a series of papers on task scheduling [18, 21, 22, 24, 28, 23, 32, 15]. We adapt the SAG approach to support the analysis of TAS-shaped communication flows in a TSN switch.

2 System Model

As shown in Figure 1, in a TSN switch that shapes its traffic with TAS, the incoming traffic is sorted into prioritized traffic classes. Packets of the same priority are all assigned to the same traffic class. Each traffic class has a dedicated queue that forwards packets in FIFO order. At the exit of each queue, there is a gate that regulates the transmission of packets of that queue. A packet from a queue can only be transmitted when the gate of that queue is open. The opening and closing of the gates are controlled by a time-aware shaper. The time-aware shaper opens and closes the gate according to a periodic time-triggered scheduled recorded in a gate-control list (GCL) associated with each traffic class (see Section 2.2 for more details). If multiple queues have a pending packet and an open gate, then the packet with the highest priority is chosen as the next packet to be transmitted.

2.1 Flow model

In this paper, we propose an analysis to bound the worst-case latency of a set of communication flows $\mathcal{F}$ transmitted through the egress port of a TSN switch and shaped with TAS. Each communication flow $F_j \in \mathcal{F}$ may traverse several switches from its source to its destination. Similar to previous work [30, 5], we assume that each communication flow $F_j \in \mathcal{F}$ injects packets periodically at its source with a period T_j, and that each packet of F_j is subject to an end-to-end deadline $D_j \leq T_j$, i.e., a packet of F_j must have reached its destination before the next packet is injected into the network. Thus, if all packets respect their end-to-end deadline, at most one packet of F_j arrives in each switch traversed by F_j in every time interval $[k \times T_j, (k+1) \times T_j)$ with $k \in \mathbb{N}$. We further assume that the first period of each communication flow aligns with the start of the observation window without any offsets. We

analyze the transmission latency of every packet of F_j in each switch on its route separately. Therefore, a packet P_i transmitted by flow F_j traversing a switch S on its route is defined by its arrival time a_{P_i} in the switch S, its length l_{P_i} (in bytes), an absolute deadline d_{P_i} smaller or equal to the time left until the end-to-end deadline, and a priority level π_{P_i} defined as a numerical value such that a lower numerical value corresponds to a higher priority.

To model non-determinism caused by, for instance, variable forwarding and queuing delays in upstream switches, the exact arrival time a_{P_i} of each packet P_i in a switch S is assumed to be unknown a priori. Instead, we assume a lower and upper bound on the arrival time of P_i in the switch, denoted as $a_{P_i}^{\min}$ and $a_{P_i}^{\max}$, respectively. Similarly, to model the fact that the payload carried by packets may vary over time due to the dynamism of applications communicating through the network, we assume that the length of packet P_i is not exactly known a priori. Only lower and upper bounds on the packet length are known and denoted by $l_{P_i}^{\min}$ and $l_{P_i}^{\max}$, respectively. The transmission time of a packet P_i (in time units) is thus given by $C_{P_i} = \frac{l_{P_i}}{\rho}$ and a lower and upper bound on that time are given by $C_{P_i}^{\min} = \frac{l_{P_i}^{\min}}{\rho}$ and $C_{P_i}^{\max} = \frac{l_{P_i}^{\max}}{\rho}$, respectively, where ρ is the egress port's transmission rate (also sometimes called link speed).

In this paper, we often refer to an execution scenario. An execution scenario is defined as a concrete set of values for the arrival times and transmission times of packets in the switch. Since TSN has a deterministic forwarding policy, there is also a single transmission order of the packets for each execution scenario.

We define the finish time f_{P_i} of a packet P_i in a given execution scenario as the time at which its transmission ends. Adopting a similar definition of latency (or response time) as Audsley et al. [4], the latency of a packet P_i, denoted by R_{P_i}, is defined as the difference between the end of its transmission and its earliest possible arrival time, i.e., $R_{P_i} = f_{P_i} - a_{P_i}^{\min}$. The best- and worst-case latency of a packet P_i, denoted by BR_{P_i} and WR_{P_i}, respectively, are thus the smallest and largest, possible value of P_i's latency in any possible execution scenario that can happen at run-time. If $WR_{P_i} \leq d_{P_i} - a_{P_i}^{\min}$ for all $P_i \in \mathcal{P}$ then all packets will always meet their deadline and the system is deemed schedulable. Otherwise, if there is at least one packet P_i such that $WR_{P_i} > d_{P_i} - a_{P_i}^{\min}$, then the system is deemed unschedulable.

▶ **Lemma 1.** *Given the system model, it is sufficient for a worst-case latency analysis to analyze the latency of packets transmitted on a finite horizon of length H where H is the least-common multiple of all the communication flows' periods and the GCLs periods.*

Proof. Because flows and GCLs are periodic, and because the first period of every flow and GCL starts at time 0, then time H will coincide with the start of a new period of every flow and every GCL. Since every flow has a constrained end-to-end deadline (i.e., $D_j \leq T_j$), all packets injected into the network before time H must have reached their destination at or before H (otherwise they miss their deadline and the system is unschedulable). Since the system state at H and at time 0 is the same (i.e., when a new period of each flow and GCL starts and there is no pending packet), and because the TSN scheduling policy is deterministic (i.e., its decision is always the same for an identical system state), the worst-case latency of packets released after H cannot be worse than that of packets released in $[0, H)$. ◀

Therefore, the analysis must analyze only a finite set of packets $\mathcal{P}$ containing all packets transmitted by all communication flows during the first time interval of length H. We refer to the time interval $[0, H)$ as the observation window.

2.2 Switch model

As shown in Figure 1, TSN divides packets in traffic classes based on their priority. The set of all traffic classes is denoted by $\mathcal{C}$. Each traffic class $\mathcal{C}_a \in \mathcal{C}$ has a unique priority $\pi_{\mathcal{C}_a}$. All packets with the same priority as the class belong to that class. Therefore, the set of packets belonging to traffic class $\mathcal{C}_a$ is denoted by $\mathcal{P}_{\mathcal{C}_a} = \{P_i \mid \pi_{P_i} = \pi_{\mathcal{C}_a}\}$. Similar to the TSN standard, we do not limit the number of classes that may be present in the switch.

Each traffic class has a dedicated FIFO queue. Each packet that arrives in the switch is placed into the queue of the class it belongs to. Each queue has a time-aware shaper implemented as a gate that controls the transmission of packets in the queue. A queue may only transmit packets on the egress port when its gate is open. The opening and closing of a gate is controlled by a gate-control list (GCL) that contains a predefined periodically repeating time-triggered schedule based on which the gates are opened or closed. We represent the schedule recorded in the GCL for a class $\mathcal{C}_a$ as a set of time intervals $\mathcal{G}_{\mathcal{C}_a}$ where the gate of $\mathcal{C}_a$ is open, i.e., $\mathcal{G}_{\mathcal{C}_a} = \{[go_1^{\mathcal{C}_a}, gc_1^{\mathcal{C}_a}) \cup [go_2^{\mathcal{C}_a}, gc_2^{\mathcal{C}_a}) \cup \cdots \cup [go_{n_{\mathcal{C}_a}}^{\mathcal{C}_a}, gc_{n_{\mathcal{C}_a}}^{\mathcal{C}_a})]\}$, where $[go_l^{\mathcal{C}_a}, gc_l^{\mathcal{C}_a})$ is the l^{th} time interval during which $\mathcal{C}_a$'s gate is open and $n_{\mathcal{C}_a}$ denotes the number of times $\mathcal{C}_a$'s gate opens in the observation window of length H. The time intervals in $\mathcal{G}_{\mathcal{C}_a}$ are assumed to be non-overlapping, hence the condition $gc_l^{\mathcal{C}_a} < go_{l+1}^{\mathcal{C}_a}$ must hold $\forall l : 1 \leq l \leq n_{\mathcal{C}_a}$. Note that because $\mathcal{G}_{\mathcal{C}_a}$ records all intervals during which $\mathcal{C}_a$'s gate is open, $\mathcal{C}_a$'s gate is by definition closed between the intervals in $\mathcal{G}_{\mathcal{C}_a}$.

The TSN standard defines a guard-band to prevent a packet to start its transmission if it may not complete transmitting before the gate of its class closes.

In summary, a packet P_i is said to be *ready* for transmission at time t if and only if the following rules are respected:

R1 P_i is at the head of the FIFO queue of its class at time t.

R2 The gate of its class is open at time t.

R3 The gate of its class closes no earlier than time $t + C_{P_i}$.

Whenever the egress port is free, a TSN switch transmits the highest priority ready packet. If no such packet exists, then the egress port is kept idle. In this paper, packet transmissions are assumed to be non-preemptive. That is, if a packet starts transmitting at time t then it will complete its transmission at $t + C_{P_i}$. Once a packet has completed its transmission through the egress port, the Ethernet protocol requires that nothing is transmitted on the egress port for a duration known as the *Inter-Packet Gap* (IPG) denoted by C_{IPG}. Therefore, the egress port becomes free again at time $t + C_{P_i} + C_{\text{IPG}}$.

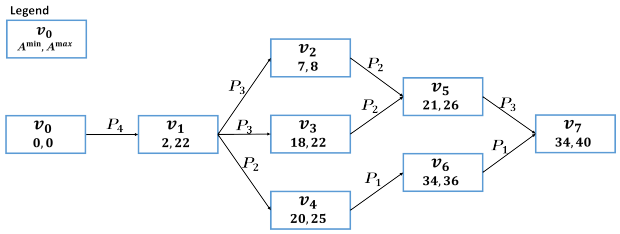
Operators definitions

For conciseness, we use $\min_\infty\{\mathcal{X}\}$ where $\mathcal{X}$ is a set as a short-hand notation for $\min\{\mathcal{X} \cup \infty\}$.

We also define the difference between two sets of time intervals $\mathcal{I}_A$ and $\mathcal{I}_B$ as $\mathcal{I}_A \setminus \mathcal{I}_B = \{t \mid t \in \mathcal{I}_A \ \wedge \ t \notin \mathcal{I}_B\}$, i.e., it removes the overlap between the two sets from $\mathcal{I}_A$.

3 Schedulability Analysis

The schedulability analysis proposed in this paper is inspired by the *Schedule Abstraction Graph* (SAG) analysis framework detailed in [18], which was developed to compute the best- and worst-case response times for non-preemptive tasks scheduled with a job-level fixed priority algorithm on a uniprocessor platform.



Packet P_i	Class $\mathcal{C}_a$	$[a_{P_i}^{min}, a_{P_i}^{max}]$	$[C_{P_i}^{min}, C_{P_i}^{max}]$	$\mathcal{G}_{\mathcal{C}_a}$
P_1	$\mathcal{C}_1$	[1,8]	[11,11]	{[0,4),[23,60)}
P_2	$\mathcal{C}_2$	[11,13]	[4,4]	{[0,10),[17,60)}
P_3	$\mathcal{C}_3$	[3,10]	[4,4]	{[0,7),[14,60)}
P_4	$\mathcal{C}_4$	[0,0]	[1,21]	{[0,60)}

Figure 2 SAG of the packet set $\{P_1, P_2, P_3, P_4\}$ with the properties presented in the table.

In this paper, we adapt the principles of the SAG framework to analyze the best- and worst-case latency of packets non-preemptively transmitted through a TSN switch's egress port shaped by TAS[1].

Our proposed schedulability analysis builds a directed acyclic graph denoted by $G = (V, E)$, where V is the set of vertices and E is the set of edges of the graph (see Figure 2 for an example that will illustrate our analysis and be discussed throughout the paper). The graph captures all possible orders in which packets may be transmitted through the egress port, and all possible states in which the egress may be in after transmitting packets. A vertex $v_k \in V$ represents a system state. A system state is defined by (1) the set of packets $\mathcal{P}(v_k)$ that have already been transmitted through the egress port, and (2) the time at which the egress port will be available to transmit a new packet. Due to the non-deterministic arrival time and length of the incoming packets, the precise availability time of the egress port after sending a set of packets is not exactly known. Instead, the schedulability analysis derives a lower- and an upper-bound on the time at which the egress may become available. We denote those bounds by $A^{\min}(v_k)$ and $A^{\max}(v_k)$, respectively. Thus, we know that in system state v_k, the egress port is certainly unavailable before time $A^{\min}(v_k)$ and is certainly available for transmitting a new packet at or after time $A^{\max}(v_k)$. At any time instant within $[A^{\min}(v_k), A^{\max}(v_k))$, the egress may or may not be available to transmit a packet.

Edges of the graph represent state transitions and are labeled with packets. An edge $e_m \in E$ labeled with a packet P_i connecting a vertex $v_k \in V$ to a vertex $v_l \in V$ models the transmission of the packet P_i when in system state v_k, thereby resulting in the new system state v_l. Note that by its definition, $\mathcal{P}(v_k)$ contains all packets labeling edges on a path ending at vertex v_k.

Key notations that have been presented until now have been summarized in Table 1.

3.1 Challenges

Building a SAG for the analysis of the worst-case latency of packets transmitted by a TSN switch requires a completely different solution to that presented in [18] or any of the follow-up papers on the schedule abstraction. This is mainly due to the fact that new challenges are introduced by the existence of FIFO queues and TAS in TSN switches, as similar concepts are currently not supported by the existing SAG analysis framework.

The SAG analysis presented in [18] assumes that there exists a fixed and consistent priority ordering between jobs, i.e., jobs have either different priorities or if two pending jobs have the same priority, the tie is broken consistently in any execution scenario. In this paper, such an assumption cannot hold as the priority of a communication flow is based on the

[1] Source code of the SAG framework available at `https://github.com/orgs/SAG-org/repositories`

Table 1 Key notations defined in the System Model for use throughout the paper.

Symbol	Description
$\pi_{\mathcal{C}_a}$	Priority of traffic class $\mathcal{C}_a$.
$\mathcal{G}_{\mathcal{C}_a}$	Set of time intervals when the gate of class $\mathcal{C}_a$ is open.
π_{P_i}	Priority of packet P_i.
$[a_{P_i}^{\min}, a_{P_i}^{\max}]$	Bounds on the arrival time of packet P_i into the switch.
$[C_{P_i}^{\min}, C_{P_i}^{\max}]$	Bounds on the transmission times of packet P_i.
d_{P_i}	Deadline of packet P_i.
$[A^{\min}(v_k), A^{\max}(v_k)]$	Bounds on the time at which the egress may become available.
$\mathcal{P}(v_k)$	Set of all packets labeling edges on a path ending at vertex v_k.

traffic class they belong to. This means that all packets of communication flows belonging to the same traffic class have the same priority. Packets with the same priority are placed into a FIFO queue when arriving in the TSN switch. Hence, the tie-breaking rule between packets of the same priority depends on their arrival times, leading to different packets winning the tie depending on the specific execution scenario. We discuss how we address this challenge in Section 4.1.

The next challenge addressed in this paper is how TAS is considered in the SAG algorithm. Since packets may only transmit when the gate of their class is open, the scheduling policy becomes non-work conserving, i.e., the egress port may be kept idle even when there are packets pending in the switch. Furthermore, the interference of higher priority flows becomes much more challenging to account for. We discuss the solution to support TAS in the SAG generation algorithm in Section 4.3.

3.2 SAG generation algorithm

In this section, we explain how to build a SAG for the transmission of packets shaped with TAS. The SAG is built using Algorithm 1. In addition to building the SAG, Algorithm 1 computes bounds on the best- and worst-case latency of every packet in $\mathcal{P}$. If at any point during the analysis, the algorithm detects that the computed bound on the worst-case latency of a packet is larger than its relative deadline, the algorithm stops and returns that the system is unschedulable. Otherwise, if the SAG is successfully built without any deadline miss for any packet, the algorithm concludes that the system is schedulable and returns the best- and worst-case latency of every packet.

Algorithm 1 records the best- and worst-case latency of every packet $P_i \in \mathcal{P}$ in any execution scenario analyzed by the SAG in the variable BR_{P_i} and WR_{P_i}, respectively. The best-case latency BR_{P_i} is initially set to ∞ and the worst-case latency WR_{P_i} is initially set to 0 for every packet $P_i \in \mathcal{P}$ when no execution scenario has been analyzed yet (lines 3 and 4). Algorithm 1 then starts building the SAG G. It starts by adding a single vertex v_0 modeling the system state when no packet has been transmitted yet. That system state thus has an empty set $\mathcal{P}(v_0)$ of packets already-transmitted, and the egress port is certainly available at time 0 for transmitting a first packet. Therefore, we have $A^{\min}(v_0) = A^{\max}(v_0) = 0$ (line 5). The SAG G is then expanded in a breadth-first manner (lines 6 to 25). While there are reachable system states in which not all packets have been transmitted yet, Algorithm 1 picks a leaf vertex v_k of the graph G with the shortest path i.e., the smallest set of transmitted packets. Algorithm 1 finds all packets that can possibly be transmitted next in the system state modeled by v_k. These packets are called possibly eligible packets (refer to Section 4.2.3

for details on how to find the possibly eligible packets). For each possibly eligible packet P_i, we find the time intervals in which P_i can start its transmission considering the shaping imposed by TAS on the gates, interference by higher priority packets and the availability of the egress port. These time intervals are referred to as eligible transmission intervals for P_i in v_k and are stored in the set $\mathcal{E}_{P_i}(v_k)$ (line 8). We explain how to build the set $\mathcal{E}_{P_i}(v_k)$ in Section 4.3. For each time interval in which the packet P_i may start transmitting, i.e., each time interval $\mathcal{E}^j_{P_i}(v_k)$ in $\mathcal{E}_{P_i}(v_k)$, Algorithm 1 computes the earliest and latest time at which P_i may start transmitting in the interval $\mathcal{E}^j_{P_i}(v_k)$, and use those times to compute the earliest and latest time at which the transmission of P_i may complete. Those times are referred to as the earliest finish time $EFT_{i,j}(v_k)$ and latest finish time $LFT_{i,j}(v_k)$, respectively (lines 11 and 10). The EFT and LFT of P_i are then used to update the best- and worst-case latency of P_i (lines 12 and 13), and to compute the resulting system state after transmitting P_i in $\mathcal{E}^j_{P_i}(v_k)$, i.e., the time at which the egress port becomes possibly and certainly available to transmit the next packet (lines 15 and 16). A new vertex v_l is created for that system state and added to the graph G with an edge directed from the previous system state v_k to the newly reached system state v_l. The edge is labeled with packet P_i.

Algorithm 1 Schedule Abstraction Graph generation algorithm with FIFO Queues and TAS.

1: **Input:** packet set $\mathcal{P}$
2: **Output:** Schedule Abstraction Graph $G = (V, E)$
3: $\forall P_i \in \mathcal{P}, WR_{P_i} \leftarrow 0$
4: $\forall P_i \in \mathcal{P}, BR_{P_i} \leftarrow \infty$
5: Initialize the graph G by adding v_0 with $A^{\min}(v_0) = A^{\max}(v_0) = 0$
6: **while** $\exists$ path from v_0 to a leaf vertex v_k s.th. $\mathcal{P}(v_k) \neq \mathcal{P}$ **do**
7: **for** each eligible packet P_i according to Section 4.2.3 **do**
8: $\mathcal{E}_{P_i}(v_k) \leftarrow$ Eq. (12)
9: **for** each interval $\mathcal{E}^l_{P_i}(v_k)$ in $\mathcal{E}_{P_i}(v_k)$ **do**
10: $LFT_{i,j}(v_k) \leftarrow$ Eq. (13)
11: $EFT_{i,j}(v_k) \leftarrow$ Eq. (14)
12: $BR_{P_i} \leftarrow \min\{EFT_{i,j}(v_k) - a^{\min}_{P_i},\ BR_{P_i}\}$
13: $WR_{P_i} \leftarrow \max\{LFT_{i,j}(v_k) - a^{\min}_{P_i},\ WR_{P_i}\}$
14: Create a new vertex v_l
15: $A^{\min}(v_l) \leftarrow$ Eq. (15)
16: $A^{\max}(v_l) \leftarrow$ Eq. (16)
17: Connect v_k to v_l with an edge labeled P_i
18: **if** $\exists$ v_m that can be merged with v_l **then**
19: Merge v_m and v_l
20: Redirect all incoming edges of v_l to v_m
21: Remove v_l from G
22: **end if**
23: **end for**
24: **end for**
25: **end while**

To curb the state space explosion inherent to reachability-based analyses, Algorithm 1 merges vertices together under the following conditions. Two vertices v_l and v_m can be merged with each other if they have the same set of transmitted packets (irrespective of the order in which they were transmitted) and if the intervals of time at which the egress port may potentially become available in both system states are either overlapping or are contiguous. We discuss why and how such system states are merged in Section 6. If there exists a vertex v_m that may be merged with v_l, they will be merged, all incoming edges of v_l are redirected to v_m, and v_l is removed from the graph (lines 18 to 22).

Note that due to the properties of TAS, Algorithm 1 must compute the eligible intervals for the transmission of packets in a given system state v_k. Because more than one such eligible interval may exist for a packet P_i in system state v_k, there may be multiple different

system states that result from the transmission of a single eligible packet P_i in system state v_k. This is very different from previous SAG-based schedulability analyses such as those presented in [18, 21, 22].

In the next sections, we explain how to find possibly eligible packets (Section 4.2), how to compute their eligible transmission intervals (Section 4.3), how to compute their EFT and LFT and derive the new system state after transmitting an eligible packet P_i within an eligible interval (Section 5), and how to merge system states (Section 6).

4 Possibly Eligible Packets and Eligible Transmission Intervals

According to the SAG generation algorithm discussed in Section 3, in a system state v_k, a packet P_i may be transmitted next through the egress if it is the highest-priority packet that is ready in that system state. Ready means that P_i is at the head of its FIFO queue, and the gate of its class is open and will not close until C_i time units later. We discuss the impact of FIFO ordering in Section 4.1, and how to obtain the intervals in which a packet is the highest-priority ready packet in Sections 4.2 and 4.3.

4.1 Accounting for FIFO Queues

Each traffic class has a FIFO queue that stores the packets arriving into the switch. Packets must be at the head of their FIFO queue to possibly be transmitted next on the switch's egress (by Rule R1, see Section 2.2). In the rest of this section, we derive the set of packets that may be at the head of the FIFO queue when the system reached a system state v_k. To do so, we first derive the earliest time when a packet will certainly be in the queue of each traffic class $\mathcal{C}_a \in \mathcal{C}$ (Lemma 2). We denote that time instant by $tr^{\max}_{\mathcal{C}_a}(v_k)$. We then use that time to get the set $top_{\mathcal{C}_a}(v_k)$ of all packets that may have entered $\mathcal{C}_a$'s queue before $tr^{\max}_{\mathcal{C}_a}(v_k)$ (Lemma 3), and we prove that a packet P_i may possibly be at the head of its queue if and only if it is in $top_{\mathcal{C}_a}(v_k)$ (Theorem 6).

▶ **Lemma 2.** *Let the system be in state v_k. The earliest time when a packet is certainly in the queue of class $\mathcal{C}_a$ is*

$$tr^{\max}_{\mathcal{C}_a}(v_k) = \min_{\infty}\{a^{\max}_{P_i} \,|P_i \in \mathcal{C}_a \setminus \mathcal{P}(v_k)\} \tag{1}$$

Proof. The earliest time when a packet is certainly in $\mathcal{C}_a$'s queue is the earliest time a packet of class $\mathcal{C}_a$ certainly arrives in the switch in system state v_k. That time is the earliest maximum arrival time of any packet of class $\mathcal{C}_a$ that was not transmitted yet in state v_k (i.e., any packet $P_l \in \mathcal{C}_a \setminus \mathcal{P}(v_k)$). Thus, $tr^{\max}_{\mathcal{C}_a}(v_k)$ is given by $\min_{\infty}\{a^{\max}_{P_i} \,|P_i \in \mathcal{C}_a \setminus \mathcal{P}(v_k)\}$. ◀

We use $tr^{\max}_{\mathcal{C}_a}(v_k)$ in Lemma 3 to derive the set of packets that may possibly be in $\mathcal{C}_a$'s FIFO queue at $tr^{\max}_{\mathcal{C}_a}(v_k)$.

▶ **Lemma 3.** *Let the system be in state v_k. The set of packets that may possibly be in $\mathcal{C}_a$'s FIFO queue at $tr^{\max}_{\mathcal{C}_a}(v_k)$ is*

$$top_{\mathcal{C}_a}(v_k) = \{P_i |\, P_i \in \mathcal{C}_a \setminus \mathcal{P}(v_k) \wedge a^{\min}_{P_i} \leq tr^{\max}_{\mathcal{C}_a}(v_k)\}. \tag{2}$$

Proof. Only packets of class $\mathcal{C}_a$ that have not yet been transmitted (i.e., all packets in $\mathcal{C}_a \setminus \mathcal{P}(v_k)$) may be in $\mathcal{C}_a{}'s$ FIFO queue at $tr^{\max}_{\mathcal{C}_a}(v_k)$. Moreover, only packets that may potentially arrive in the switch at or before $tr^{\max}_{\mathcal{C}_a}(v_k)$ may be in the queue by $tr^{\max}_{\mathcal{C}_a}(v_k)$. Thus, only packets in $\mathcal{C}_a \setminus \mathcal{P}(v_k)$ with an earliest arrival time $a^{\min}_{P_i}$ such that $a^{\min}_{P_i} \leq tr^{\max}_{\mathcal{C}_a}(v_k)$ may be in $\mathcal{C}_a$'s FIFO queue at $tr^{\max}_{\mathcal{C}_a}(v_k)$. ◀

We now prove with the next two lemmas that all packets in $top_{\mathcal{C}_a}(v_k)$ may be at the head of $\mathcal{C}_a$'s FIFO queue and that no packet outside of $top_{\mathcal{C}_a}(v_k)$ may be at the head of $\mathcal{C}_a$'s FIFO queue when in system state v_k.

▶ **Lemma 4.** *If packet P_i is in $top_{\mathcal{C}_a}(v_k)$, then there is at least one execution scenario where P_i is at the head of $\mathcal{C}_a$'s FIFO queue in system state v_k.*

Proof. A packet P_i is at the head of $\mathcal{C}_a$'s FIFO queue if it is the first not-yet-transmitted packet of $\mathcal{C}_a$ that arrives in the switch. We know that if packet P_i is in $top_{\mathcal{C}_a}(v_k)$, then its earliest arrival time $a_{P_i}^{\min}$ is such that $a_{P_i}^{\min} \leq tr_{\mathcal{C}_a}^{\max}(v_k)$ (by Eq. (2)). Moreover, by Eq. (1), we also know that for all packets $P_l \in \mathcal{C}_a \setminus \mathcal{P}(v_k)$ that have not yet been transmitted, their latest arrival time is larger than or equal to $tr_{\mathcal{C}_a}^{\max}(v_k)$, i.e., $\forall P_l \in \mathcal{C}_a \setminus \mathcal{P}(v_k), a_l^{\max} \geq tr_{\mathcal{C}_a}^{\max}(v_k)$. Therefore, there is at least one execution scenario where P_i arrives at a_{P_i} such that $a_{P_i}^{\min} \leq a_{P_i} \leq tr_{\mathcal{C}_a}^{\max}(v_k)$ and all other packets of class $\mathcal{C}_a$ that have not yet been transmitted arrive at or after $tr_{\mathcal{C}_a}^{\max}(v_k)$. In such execution scenario, P_i is the first to enter $\mathcal{C}_a$'s queue and is thus at the head of the queue. ◀

▶ **Lemma 5.** *If P_i is not in $top_{\mathcal{C}_a}(v_k)$, then P_i cannot be at the head of $\mathcal{C}_a$'s FIFO queue in system state v_k.*

Proof. From Lemma 2, we know that there is certainly a packet in $\mathcal{C}_a$'s FIFO queue at time $tr_{\mathcal{C}_a}^{\max}(v_k)$ in system state v_k. Thus, any packet at the head of $\mathcal{C}_a$'s FIFO queue in system state v_k must (1) not have been transmitted yet in system state v_k (i.e., it must be in $\mathcal{C}_a \setminus \mathcal{P}(v_k)$), and (2) must have arrived in the switch at or before $tr_{\mathcal{C}_a}^{\max}(v_k)$ (i.e., $a_{P_i} \leq tr_{\mathcal{C}_a}^{\max}(v_k)$). However, if P_i is not in $top_{\mathcal{C}_a}(v_k)$, then, by Eq. (2), either $P_i \notin \mathcal{C}_a \setminus \mathcal{P}(v_k)$ or $a_{P_i}^{\min} > tr_{\mathcal{C}_a}^{\max}(v_k)$ and thus $a_{P_i} > tr_{\mathcal{C}_a}^{\max}(v_k)$ (since $a_{P_i}^{\min}$ is a lower bound on P_i's arrival time). Therefore, if P_i is not in $top_{\mathcal{C}_a}(v_k)$, then P_i cannot be at the head of $\mathcal{C}_a$'s FIFO queue in system state v_k. ◀

We use the two lemmas above to derive our main theorem.

▶ **Theorem 6.** *There is at least one execution scenario such that packet P_i will be at the head of $\mathcal{C}_a$'s FIFO queue in system state v_k if and only if $P_i \in top_{\mathcal{C}_a}(v_k)$.*

Proof. The "if" is proven by Lemma 4, and the "only if" is the contrapositive of Lemma 5. ◀

4.2 Potentially eligible packets

After computing all packets that can possibly be on the head of their respective queue, we find which of these packets can potentially be the next packet to be transmitted on the egress in system state v_k. To decide whether a packet P_i is potentially eligible to be the next transmitted packet, we first derive a lower bound, called $t_{P_i}^{\text{lb}}(v_k)$, on the earliest time P_i may start its transmission in the system state v_k (Section 4.2.1), and an upper bound, called $t^{\text{ub}}(v_k)$, on the latest time by which P_i must start to transmit to be the next packet transmitted on the egress (Section 4.2.2). If $t_{P_i}^{\text{lb}}(v_k) > t^{\text{ub}}(v_k)$, then P_i cannot be the next packet transmitted in system state v_k as it would otherwise lead to a contradiction on the definitions of $t_{P_i}^{\text{lb}}(v_k)$ and $t^{\text{ub}}(v_k)$. On the other hand, if $t_{P_i}^{\text{lb}}(v_k) \leq t^{\text{ub}}(v_k)$, then P_i is potentially eligible to be transmitted next in system state v_k.

4.2.1 Lower-bound on P_i's start time

We first prove a lower bound on packet P_i's start of transmission in a system state v_k.

▶ **Lemma 7.** *A packet P_i may not start being transmitted earlier than*

$$t^{\text{lb}}_{P_i}(v_k) = \max\{a^{\min}_{P_i}, A^{\min}(v_k)\}. \tag{3}$$

Proof. P_i may not start being transmitted before the time it arrives in the switch, which is lower bounded by $a^{\min}_{P_i}$. Furthermore, it cannot be transmitted before the earliest time the egress port is available in system state v_k, which is lower bounded by $A^{\min}(v_k)$. Thus, P_i cannot be transmitted earlier than $\max\{a^{\min}_{P_i}, A^{\min}(v_k)\}$. ◀

4.2.2 Upper-bound on P_i's start time

According to the TSN scheduling policy described in Section 2, a packet P_i will certainly be transmitted when the egress port is available, P_i is ready, and it is the highest priority packet that is ready for transmission. Therefore, as soon as there is a ready packet and the egress port is available, a packet will certainly be transmitted. In the following, we compute an upper bound on the time at which the egress port is available and a packet is certainly ready.

According to Rule R1, see Section 2.2, to be ready, a packet must be at the head of its queue. The following lemma directly follows.

▶ **Lemma 8.** *The earliest time when the egress is certainly available and there is certainly a packet at the head of $\mathcal{C}_a$'s FIFO queue in system state v_k is*

$$\hat{t}_{\mathcal{C}_a}(v_k) = \max\{tr^{\max}_{\mathcal{C}_a}(v_k), A^{\max}(v_k)\} \tag{4}$$

Proof. Only packets in the set $top_{\mathcal{C}_a}(v_k)$ of each class $\mathcal{C}_a$ may be ready in system state v_k. According to Lemma 2, a packet may reach the head of $\mathcal{C}_a$'s FIFO queue at the latest at $tr^{\max}_{\mathcal{C}_a}(v_k)$. Furthermore, by definition of $A^{\max}(v_k)$, the egress becomes available at the latest at $A^{\max}(v_k)$ in system state v_k. Thus, the earliest time when the egress is *certainly* available and there is *certainly* a packet at the head of $\mathcal{C}_a$'s FIFO queue is $\max\{tr^{\max}_{\mathcal{C}_a}(v_k), A^{\max}(v_k)\}$. ◀

Then, according to Rules R2 and R3, for a packet P_i to be ready, the gate of its class must be open and must not close until C_i time units later. To find the first time instant when that condition is certainly true, Lemma 9 derives the set of intervals in which P_i may not be able to start to be transmitted because either R2 and/or R3 is not respected.

▶ **Lemma 9.** *A packet P_i of class $\mathcal{C}_a$ may* not *be able to start transmitting in any time interval in the set $\mathcal{G}^{\text{pos}}_{P_i}$ defined below due to not respecting Rule R2 and/or Rule R3.*

$$\mathcal{G}^{\text{pos}}_{P_i} = \left\{[0, go^{\mathcal{C}_a}_1) \cup \left\{ \bigcup_{1 \le l < n_{\mathcal{C}_a}} (gc^{\mathcal{C}_a}_l - C^{\max}_{P_i}, go^{\mathcal{C}_a}_{l+1}) \right\} \cup (gc^{\mathcal{C}_a}_{n_{\mathcal{C}_a}} - C^{\max}_{P_i}, \infty)\right\} \tag{5}$$

Proof. According to Rule R2 packet P_i may not start being transmitted in any interval where the gate of its class is closed. Thus, P_i cannot be transmitted in any of the intervals $\left\{[0, go^{\mathcal{C}_a}_1) \cup \left\{\bigcup_{1 \le l < n_{\mathcal{C}_a}} [gc^{\mathcal{C}_a}_l, go^{\mathcal{C}_a}_{l+1})\right\} \cup [gc^{\mathcal{C}_a}_{n_{\mathcal{C}_a}}, \infty)\right\}$. Furthermore, according to Rule R3, P_i may not start being transmitted during the $C_i - \epsilon$ time units before the gate closes (where ϵ is an arbitrarily small value). Therefore, P_i may not start being transmitted in the time intervals

in the set $\bigcup_{1\leq l\leq n_{\mathcal{C}_a}}(gc_l^{\mathcal{C}_a}-C_i, gc_l^{\mathcal{C}_a})$. The lengths of those intervals are maximized when $C_i = C_{P_i}^{\max}$. Therefore, P_i may not be able to start beng transmitted in any time interval in the set $\left\{[0, go_1^{\mathcal{C}_a}) \cup \left\{\bigcup_{1\leq l<n_{\mathcal{C}_a}}[gc_l^{\mathcal{C}_a}, go_{l+1}^{\mathcal{C}_a})\right\} \cup [gc_{n_{\mathcal{C}_a}}^{\mathcal{C}_a}, \infty)\right\} \cup \left\{\bigcup_{1\leq l\leq n_{\mathcal{C}_a}}(gc_l^{\mathcal{C}_a} - C_{P_i}^{\max}, gc_l^{\mathcal{C}_a})\right\}$

$= \left\{[0, go_1^{\mathcal{C}_a}) \cup \left\{\bigcup_{1\leq l<n_{\mathcal{C}_a}}(gc_l^{\mathcal{C}_a} - C_{P_i}^{\max}, go_{l+1}^{\mathcal{C}_a})\right\} \cup (gc_{n_{\mathcal{C}_a}}^{\mathcal{C}_a} - C_{P_i}^{\max}, \infty)\right\}$. ◀

Let gc_i^{last} be that last time instant before time $\hat{t}_{\mathcal{C}_a}(v_k)$ where P_i may not be able to be transmitted according to $\mathcal{G}_{P_i}^{\text{pos}}$, and let go_i^{next} be the first time instant after gc_i^{last} at which packet P_i may start being transmitted according to $\mathcal{G}_{P_i}^{\text{pos}}$. That is, we have

$$gc_i^{\text{last}} = \max\{t \mid t \in \mathcal{G}_{P_i}^{\text{pos}} \wedge t \leq \hat{t}_{\mathcal{C}_a}(v_k)\} \tag{6}$$

$$go_i^{\text{next}} = \min\{t \mid t \notin \mathcal{G}_{P_i}^{\text{pos}} \wedge t > gc_i^{\text{last}}\} \tag{7}$$

Then, the following lemma holds.

▶ **Lemma 10.** *If $P_i \in top_{\mathcal{C}_a}(v_k)$ is at the head of $\mathcal{C}_a$'s FIFO queue, then the earliest time when P_i is certainly ready and the egress is certainly available is*

$$t_{P_i}^{\text{ready}} = \max\{\hat{t}_{\mathcal{C}_a}(v_k),\ go_i^{\text{next}}\} \tag{8}$$

Proof. According to the TSN scheduling policy presented in Section 2, to be ready, a packet P_i must respect Rules R1 to R3. By Lemma 8, the earliest time P_i certainly respects R1 and the egress is certainly available is at time $\hat{t}_{\mathcal{C}_a}(v_k)$. If $\hat{t}_{\mathcal{C}_a}(v_k) \geq go_i^{\text{next}}$, then by definition of go_i^{next} and Lemma 9, Rules R2 and R3 are certainly respected at $\hat{t}_{\mathcal{C}_a}(v_k)$. Thus, P_i is certainly ready (i.e., it respects R1, R2 and R3), and the egress is certainly available. This proves the first case of Eq. (8). If, on the other hand, $\hat{t}_{\mathcal{C}_a}(v_k) < go_i^{\text{next}}$, then by definition of go_i^{next} and Lemma 9, P_i may not be able to start to transmit, and thus may not respect Rules R2 and R3 until time go_i^{next}. Since by definition of go_i^{next} and Lemma 9, Rules R2 and R3 are certainly respected at go_i^{next}, go_i^{next} is the earliest time R1 to R3 are certainly respected and the egress is certainly available after $\hat{t}_{\mathcal{C}_a}(v_k)$. This prove the second case of Eq. (8). ◀

Lemma 10 refers to a specific packet of class $\mathcal{C}_a$, but, by Lemma 3, we know that any packet in $top_{\mathcal{C}_a}(v_k)$ may be at the head of $\mathcal{C}_a$'s FIFO queue. Thus, the earliest time when the packet at the head of $\mathcal{C}_a$'s FIFO queue is certainly ready and the egress is certainly available in system state v_k is given by

$$\max_{\forall P_i \in top_{\mathcal{C}_a}(v_k)}\{t_{P_i}^{\text{ready}}\}.$$

Since according to the TSN scheduling policy, a packet is transmitted when at least one packet is ready and the egress is available, the earliest time the next packet of any class is certainly being transmitted on the egress in system state v_k is

$$t^{\text{ub}}(v_k) = \min_{\forall \mathcal{C}_a \in \mathcal{C}}\left\{\max_{\forall P_i \in top_{\mathcal{C}_a}(v_k)}\{t_{P_i}^{\text{ready}}\}\right\}. \tag{9}$$

We state now our main result about $t^{\text{ub}}(v_k)$.

▶ **Lemma 11.** *If packet P_i is the next packet transmitted in system state v_k, then P_i is transmitted no later than $t^{\text{ub}}(v_k)$.*

Proof. Since there is certainly a packet transmitted at $t^{\text{ub}}(v_k)$, if P_i is the next packet transmitted, then it must be transmitted at or before $t^{\text{ub}}(v_k)$. ◀

4.2.3 Packet's potential eligibility

We say that a packet P_i is potentially eligible in system state v_k if it may be the next packet transmitted in system state v_k. Lemma 12 below proves a condition that must be respected for a packet P_i to be potentially eligible.

▶ **Lemma 12.** *Packet P_i may be the next packet transmitted in system state v_k only if $t^{\text{lb}}_{P_i}(v_k) \leq t^{\text{ub}}(v_k)$.*

Proof. According to Lemma 7, P_i cannot be transmitted earlier than $t^{\text{lb}}_{P_i}(v_k)$ in system state v_k. Moreover, according to Lemma 11, P_i must start being transmitted before $t^{\text{ub}}(v_k)$ if it is the next packet transmitted in system state v_k. Thus, P_i cannot be the next packet transmitted in system state v_k if $t^{\text{lb}}_{P_i}(v_k) > t^{\text{ub}}(v_k)$. Taking the contrapositive of the last statement proves the lemma. ◀

4.3 Eligible transmission intervals of potential eligible packets

In this section, we elaborate on how to find the eligible transmission intervals of potentially eligible packets in system state v_k. Let P_i be a potentially eligible packet of class $\mathcal{C}_a$. $t^{\text{lb}}_{P_i}(v_k)$ and $t^{\text{ub}}(v_k)$ provide lower and upper bounds on the time interval within which P_i may possibly start being transmitted in system state v_k. However, P_i cannot start being transmitted at every time instance within $[t^{\text{lb}}_{P_i}(v_k), t^{\text{ub}}(v_k)]$ because this interval does not fully consider the shaping imposed by TAS on the gate of class $\mathcal{C}_a$, nor does it account for interference from ready packets of higher-priority classes. Lemmas 13 and 14 account for the former, and Lemma 15 accounts for the latter.

▶ **Lemma 13.** *A packet P_i can certainly not start its transmission in the set of time intervals*

$$\mathcal{G}^{\text{cert}}_{P_i} = \left\{ [0, go_1^{\mathcal{C}_a}) \cup \left\{ \bigcup_{1 \leq l < n_{\mathcal{C}_a}} (gc_l^{\mathcal{C}_a} - C^{\min}_{P_i}, go_{l+1}^{\mathcal{C}_a}) \right\} \cup (gc_{n_{\mathcal{C}_a}}^{\mathcal{C}_a} - C^{\min}_{P_i}, \infty) \right\} \tag{10}$$

Proof. According to Rule R2, packet P_i may never start being transmitted in any interval where the gate of its class is closed. Thus, P_i cannot transmit in any of the intervals $\left\{ [0, go_1^{\mathcal{C}_a}) \cup \left\{ \bigcup_{1 \leq l < n_{\mathcal{C}_a}} [gc_l^{\mathcal{C}_a}, go_{l+1}^{\mathcal{C}_a}) \right\} \cup [gc_{n_{\mathcal{C}_a}}^{\mathcal{C}_a}, \infty) \right\}$. Furthermore, according to Rule R3, P_i may not start being transmitted during the C_i time units before the gate closes. Therefore, P_i may not start being transmitted in the time intervals in the set $\bigcup_{1 \leq l \leq n_{\mathcal{C}_a}} (gc_l^{\mathcal{C}_a} - C_i, gc_l^{\mathcal{C}_a})$. The lengths of those intervals is minimized when $C_i = C^{\min}_{P_i}$. Therefore, P_i can *certainly* not start transmitting in the union of the above two sets of time intervals, thus proving the lemma. ◀

▶ **Lemma 14.** *A potentially eligible packet P_i in system state v_k may only start its transmission at the time instants in the set*

$$\left\{ \left[t^{\text{lb}}_{P_i}(v_k), t^{\text{ub}}(v_k) \right] \right\} \setminus \mathcal{G}^{\text{cert}}_{P_i}$$

Proof. Lemmas 7 and 11 prove that, if P_i is the next packet to be transmitted in system state v_k, then it cannot start transmitting outside the time interval $\left[t^{\text{lb}}_{P_i}(v_k), t^{\text{ub}}(v_k) \right]$. Furthermore, Lemma 13 proves that P_i can certainly not start being transmitted in any interval in the set $\mathcal{G}^{\text{cert}}_{P_i}$. Thus, P_i may only start transmitting at time instants in the set $\left\{ \left[t^{\text{lb}}_{P_i}(v_k), t^{\text{ub}}(v_k) \right] \right\} \setminus \mathcal{G}^{\text{cert}}_{P_i}$. ◀

▶ **Lemma 15.** *Let P_m be the packet in $top_{\mathcal{C}_b}(v_k)$ with the largest possible transmission time, i.e., $P_m = \underset{\forall P_l \in top_{\mathcal{C}_b}(v_k)}{\text{argmax}} \{C_l^{\max}\}$. Then, there is certainly a packet of class $\mathcal{C}_b$ that is ready for transmission in all time intervals in the set*

$$\mathcal{I}_{\mathcal{C}_b}^{\text{hp}}(v_k) = [tr_{\mathcal{C}_b}^{\max}(v_k), \infty) \setminus \mathcal{G}_{P_m}^{\text{pos}}. \tag{11}$$

Proof. A packet is ready if it respects Rules R1, R2 and R3. According to Lemma 3, $top_{\mathcal{C}_b}(v_k)$ contains all packets of class $\mathcal{C}_b$ that may be at the head of $\mathcal{C}_b$'s FIFO queue in system state v_k. Thus, only packets in $top_{\mathcal{C}_b}(v_k)$ may respect R1. Lemma 2 proves that there is certainly a packet at the head of the FIFO queue at time $tr_{\mathcal{C}_b}^{\max}(v_k)$. Thus, R_1 is certainly respected at and after $tr_{\mathcal{C}_b}^{\max}(v_k)$. Lemma 9 proves that $\mathcal{G}_{P_l}^{\text{pos}}$ contains all time intervals in which a packet P_l may not respect R2 and/or R3. According to Eq. (5), $\mathcal{G}_{P_l}^{\text{pos}}$ contains the largest intervals when $C_l^{\max}$ is maximized. Thus, $\mathcal{G}_{P_m}^{\text{pos}}$ contains the largest intervals in which R2 and/or R3 is not respected for the packet P_m as defined in the claim. Thus, the set of time intervals during which R1, R2 and R3 are certainly respected is obtained when P_m is at the head of $\mathcal{C}_b$'s FIFO queue, and is given by $[tr_{\mathcal{C}_b}^{\max}(v_k), \infty) \setminus \mathcal{G}_{P_m}^{\text{pos}}$. ◀

Combining Lemmas 14 and 15, we build the set $\mathcal{E}_{P_i}(v_k)$ that contains all eligible transmission intervals of a potentially eligible packet P_i.

$$\mathcal{E}_{P_i}(v_k) = \left\{ \left[t_{P_i}^{\text{lb}}(v_k), t^{\text{ub}}(v_k)\right] \setminus \mathcal{G}_{P_i}^{\text{cert}} \right\} \setminus \left\{ \bigcup_{\forall\, \mathcal{C}_b \in \mathcal{C} \,|\, \pi_{\mathcal{C}_b} < \pi_{\mathcal{C}_a}} \mathcal{I}_{\mathcal{C}_b}^{\text{hp}}(v_k) \right\} \tag{12}$$

▶ **Example 16.** Let us consider 3 packets P_1, P_2 and P_3 that are yet to be dispatched at state v_x, with parameters as given in Figure 3. Assume the egress becomes possibly available at time 2 and certainly available at time 22. Assume all three packets are on the head of their respective queues. Figure 3 also shows when the gates of each class is closed. For this example, we aim to calculate the eligible transmission intervals of P_3 i.e., $\mathcal{E}_{P_3}(v_k)$.

The earliest time packet P_3 can possibly start being transmitted is when it is arrived and the egress is possibly available (as computed by Eq. (3)). This happens at time 3. Moreover, if P_3 is the first packet being transmitted in state v_x, it must certainly have started being transmitted no later than when the egress is certainly available, and there certainly is a packet in any queue with an open gate for which the guard band did not start yet (as computed by Eq. (9)). That happens at time 22 as the egress is certainly available and P_1, P_2 and P_3 are all certainly arrived in teir queue with an open gate for which the guard band did not start. Thus, P_3 will start being transmitted in the interval [3, 22], only if P_3's gate is open, the guard-band is not started and no higher priority packet is eligible to be transmitted.

As captured by $\mathcal{G}_{P_3}^{\text{cert}}$ (shaded region on P_3's timeline) calculated by Lemma 13, P_3 can certainly not transmit in the interval (5,14) as either its gate is closed or the guard band started. On the other hand, classes $\mathcal{C}_1$ and $\mathcal{C}_2$ can certainly start transmitting a packet at or after time 22 and 16, respectively, since there is certainly a packet in their queue and their gate is open. It is captured by $\mathcal{I}_{\mathcal{C}_1}^{\text{hp}}(v_k)$ and $\mathcal{I}_{\mathcal{C}_2}^{\text{hp}}(v_k)$ calculated by Lemma 15 and depicted by the shaded regions on the timelines of packets P_1 and P_2. Since P_3 cannot start when higher priority classes are able to transmit, the eligible transmission intervals of P_3 are $[3, 5]$ and $[14, 16)$ (i.e., $[3, 22] \setminus (5, 14) \setminus [22, \infty) \setminus [16, \infty)$ as calculted by Eq. (12)).

We now prove that a packet P_i may start transmitting at a time instant t if an only if $t \in \mathcal{E}_{P_i}(v_k)$.

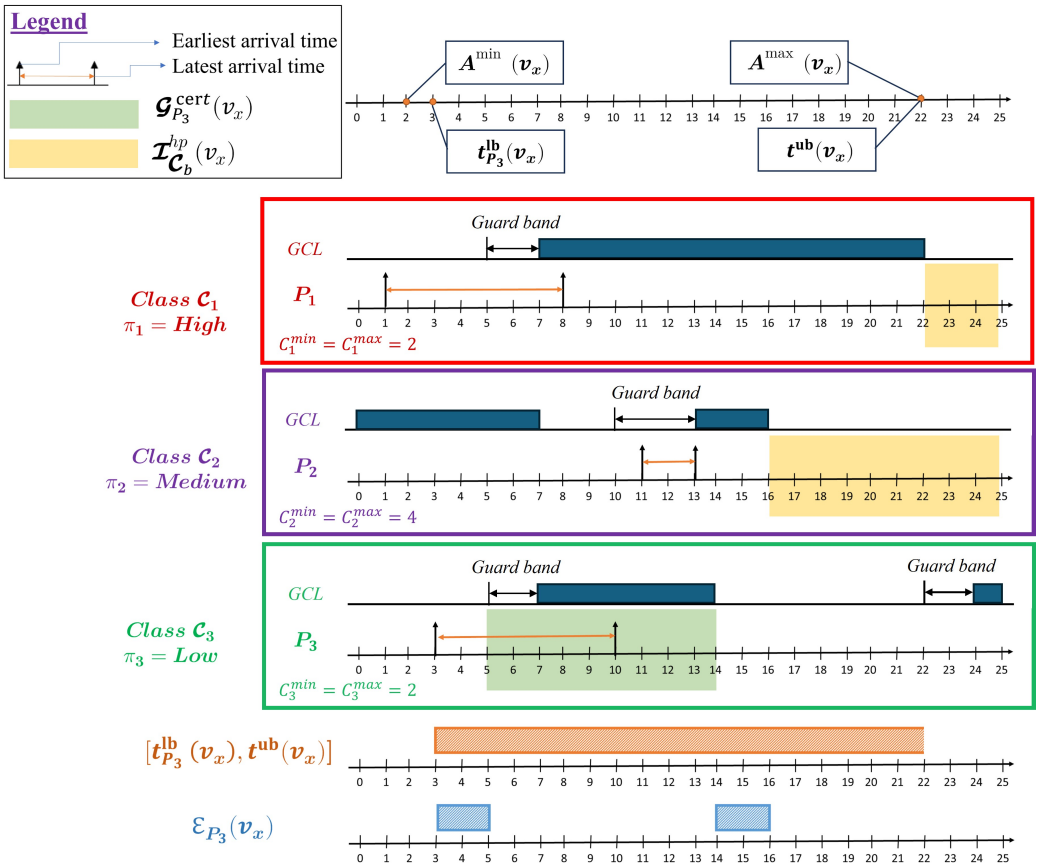


Figure 3 Example that shows how eligible transmission intervals are calculated for packet P_3.

▶ **Lemma 17.** *In system state v_k, a packet $P_i \in top_{\mathcal{C}_a}(v_k)$ can start its transmission at any time instant $t \in \mathcal{E}_{P_i}(v_k)$.*

Proof. For a packet $P_i \in \mathcal{C}_a$ to possibly start its transmission at time t, it must meet the following conditions: (i) it may be ready at t, (ii) all higher priority packets may not be ready at t, and (iii) the egress may be available at t. By Eq. (12), $\mathcal{E}_{P_i}(v_k)$ only contains time instants at or later than $t^{\text{lb}}_{P_i}(v_k)$. Since, by Eq. (3), $t^{\text{lb}}_{P_i}(v_k) \geq A^{\text{min}}(v_k)$, Condition (iii) is satisfied for every time instant $t \in \mathcal{E}_{P_i}(v_k)$. Now, we prove that (i) is respected. To be ready, and thus meet Condition (i), a packet P_i must satisfy R1, R2 and R3. Since $P_i \in top_{\mathcal{C}_a}(v_k)$, it satisfies R1. Since, by Lemma 13, the set $\mathcal{G}^{\text{cert}}_{P_i}$ contains all the time instants where R2 and/or R3 is certainly not respected in all execution scenarios, by taking the contra-positive, there is at least one execution scenario where P_i respects R2 and R3 for every time instant $t \notin \mathcal{G}^{\text{cert}}_{P_i}$. Since, by Eq. (12), $\mathcal{E}_{P_i}(v_k)$ only contains time instants that are not in $\mathcal{G}^{\text{cert}}_{P_i}$, Condition (i) is met. Similarly, since $\mathcal{I}^{\text{hp}}_{\mathcal{C}_b}(v_k)$ contains all instants when a packet of class $\mathcal{C}_b$ is certainly ready, and because Eq. (12) removes all such instants from $\mathcal{E}_{P_i}(v_k)$ for all classes with higher priority than P_i, Condition (ii) is met. Thus, Conditions (i), (ii) and (ii) are met for all $t \in \mathcal{E}_{P_i}(v_k)$, and thus P_i can possibly start at any time instant in $\mathcal{E}_{P_i}(v_k)$. ◀

▶ **Lemma 18.** *In system state v_k, a packet $P_i \in top_{\mathcal{C}_a}(v_k)$ cannot start its transmission at any time instant $t \notin \mathcal{E}_{P_i}(v_k)$.*

Proof. Let us assume that packet $P_i \in \mathcal{C}_a$ begins its transmission at time t such that $t \notin \mathcal{E}_{P_i}(v_k)$. Then, Conditions (i), (ii) and (iii) as stated in Lemma 17 must be satisfied at t. We analyze three cases: $t < t^{\text{lb}}_{P_i}(v_k)$, $t > t^{\text{ub}}(v_k)$ or $t \in [t^{\text{lb}}_{P_i}(v_k), t^{\text{ub}}(v_k)]$. **Case 1:** If $t < t^{\text{lb}}_{P_i}(v_k)$ then, by Eq. (3), $t < A^{\text{min}}(v_k)$ or $t < a^{\text{min}}_{P_i}$. If the former is true, the egress canot be available at t, contradicting Condition (iii). If the latter is true, then P_i cannot have arrived in the switch yet and thus cannot be ready, contradicting Condition (i). **Case 2:** If $t > t^{\text{ub}}(v_k)$, Lemma 11 proves that P_i cannot be the next packet transmitted in system state v_k. Thus, either Condition (i) or (ii) is not respected. **Case 3:** If $t \in [t^{\text{lb}}_{P_i}(v_k), t^{\text{ub}}(v_k)]$ and $t \notin \mathcal{E}_{P_i}(v_k)$, then, by Eq. (12), $t \in \mathcal{G}^{\text{cert}}_{P_i}$ or $\exists \mathcal{C}_b$ such that $t \in \mathcal{I}^{\text{hp}}_{\mathcal{C}_b}(v_k)$ and the priority of $\mathcal{C}_b$ is higher than P_i's priority. If the former is true, then Lemma 13 proves that P_i is not ready at t, contradicting Condition (i). If the latter is true, then Lemma 15 proves that a higher priority packet of class $\mathcal{C}_b$ is certainly ready, contradicting Condition (ii).

Therefore, we reached a contradiction for all cases. Meaning that if P_i is the next packet transmitted in system state v_k, P_i cannot start transmitting at any time instant $t \notin \mathcal{E}_{P_i}(v_k)$. ◀

▶ **Theorem 19.** *$\mathcal{E}_{P_i}(v_k)$ contains all the instants at which $P_i \in top_{\mathcal{C}_a}(v_k)$ may start transmitting and does not contain any instant at which P_i may not start transmitting in system state v_k.*

Proof. It is a direct application of Lemmas 17 and 18. ◀

5 Evolving System States

As a result of Theorem 19, if $\mathcal{E}_{P_i}(v_k)$ is not empty, then packet P_i may be the next packet transmitted through the egress in system state v_k. P_i's transmission makes the system evolve to a new state. Since the availability of the egress in the new system state v_p depends on when P_i starts transmitting. Since, according to our discussion in the previous section, there may be more than one continuous time interval in which P_i may start transmitting (i.e., there may be more than one time interval in $\mathcal{E}_{P_i}(v_k)$), Alg. 1 analyzes each continuous time interval in which P_i may start separately, i.e., a new node v_p encoding the new system state after transmitting P_i is created for every interval in $\mathcal{E}_{P_i}(v_k)$ (line 9 in Alg. 1).

We denote the start of the j^{th} time interval in $\mathcal{E}_{P_i}(v_k)$ by $\mathcal{E}^{j,\text{start}}_{P_i}(v_k)$, and its end by $\mathcal{E}^{j,\text{end}}_{P_i}(v_k)$. By Theorem 19, P_i may start transmitting at any time within $[\mathcal{E}^{j,\text{start}}_{P_i}(v_k), \mathcal{E}^{j,\text{end}}_{P_i}(v_k)]$. Thus, P_i's transmission may end at any time within $[\mathcal{E}^{j,\text{start}}_{P_i}(v_k) + C_{P_i},\ \mathcal{E}^{j,\text{end}}_{P_i}(v_k) + C_{P_i}]$. We also know from the definition of the guard-band that P_i certainly finished its transmission when the gate of its class closes (see rule R3). Therefore, the latest time P_i may finish its transmission when starting its transmission in the j^{th} interval in $\mathcal{E}_{P_i}(v_k)$ is either when P_i starts at $\mathcal{E}^{j,\text{end}}_{P_i}(v_k)$ and transmits for its maximum transmission time $C^{\text{max}}_{P_i}$, or when the gate of P_i's class (say $\mathcal{C}_a$) closes for the first time after P_i started transmitting. That is, it is given by $LFT_{i,j}(v_k)$ as defined in Eq. (13).

$$LFT_{i,j}(v_k) = \min\{\mathcal{E}^{j,\text{end}}_{P_i}(v_k) + C^{\text{max}}_{P_i},\ gc^{\text{next}}_i\} \tag{13}$$

where $gc^{\text{next}}_i = \min\{t \mid t \notin \mathcal{G}_{\mathcal{C}_a} \ \wedge t > \mathcal{E}^{j,\text{end}}_{P_i}(v_k)\}$.

Similarly, the earliest time P_i may finish its transmission when starting its transmission in the j^{th} interval in $\mathcal{E}_{P_i}(v_k)$ is when it starts at $\mathcal{E}^{j,\text{start}}_{P_i}(v_k)$ and transmits for its minimum transmission time.

$$EFT_{i,j}(v_k) = \mathcal{E}^{j,\text{start}}_{P_i}(v_k) + C^{\text{min}}_{P_i} \tag{14}$$

Using the earliest and latest finish time of P_i, we get that the earliest and latest availability time of the egress $A^{\min}(v_p)$ and $A^{\max}(v_p)$ after transmitting P_i are given by Eqs. (15) and (16), respectively.

$$A^{\min}(v_p) = EFT_{i,j}(v_k) + C_{\text{IPG}} \tag{15}$$
$$A^{\max}(v_p) = LFT_{i,j}(v_k) + C_{\text{IPG}} \tag{16}$$

where C_{IPG} is the length of the inter-packet gap, i.e., the minimum time duration defined by the Ethernet protocol that must separate the transmission of any two packets.

▶ **Lemma 20.** *The $LFT_{i,j}(v_k)$ and $EFT_{i,j}(v_k)$ calculated with Eqs.* (13) *and* (14) *are exact lower and upper-bounds on the finish time of packet P_i if P_i starts transmitting in the j^{th} interval of $\mathcal{E}_{P_i}(v_k)$.*

Proof. To prove a bound is exact, we must prove that there is at least one execution scenario such that the bound is reached, and that no execution scenario may go beyond the bound.

We first prove the claim for $EFT_{i,j}(v_k)$. Since $C^{\min}_{P_i}$ is a valid transmission time for P_i, if P_i starts at $\mathcal{E}^{j,\text{start}}_{P_i}(v_k) \in \mathcal{E}_{P_i}(v_k)$ and executes for $C^{\min}_{P_i}$ it finishes at $EFT_{i,j}(v_k)$ as defined by Eq. (14). Furthermore, to finish earlier than $EFT_{i,j}(v_k) = \mathcal{E}^{j,\text{start}}_{P_i}(v_k) + C^{\min}_{P_i}$, P_i must either start earlier than $\mathcal{E}^{j,\text{start}}_{P_i}(v_k)$ or transmit for less than $C^{\min}_{P_i}$. Any of those would either contradict that $\mathcal{E}^{j,\text{start}}_{P_i}(v_k)$ is the start of $\mathcal{E}_{P_i}(v_k)$, or that $C^{\min}_{P_i}$ is a lower bound on P_i's transmission time.

We now prove the claim for $LFT_{i,j}(v_k)$. Since $C^{\max}_{P_i}$ and lower values are valid values for the transmission time of P_i, if P_i starts at $\mathcal{E}^{j,\text{end}}_{P_i}(v_k) \in \mathcal{E}_{P_i}(v_k)$ and executes for $C_i = LFT_{i,j}(v_k) - \mathcal{E}^{j,\text{end}}_{P_i}(v_k) \leq C^{\max}_{P_i}$, it finishes at $LFT_{i,j}(v_k)$ as defined by Eq. (13). Furthermore, to finish later than $LFT_{i,j}(v_k) = \min\{\mathcal{E}^{j,\text{end}}_{P_i}(v_k) + C^{\max}_{P_i},\ gc_i^{\text{next}}\}$, P_i must either start later than $\mathcal{E}^{j,\text{end}}_{P_i}(v_k)$ or transmit for more than $C^{\max}_{P_i}$ or continues to transmit until later than gc_i^{next}. Any of those would either contradict that $\mathcal{E}^{j,\text{end}}_{P_i}(v_k)$ is the end of $\mathcal{E}_{P_i}(v_k)$, or that $C^{\max}_{P_i}$ is an upper bound on P_i's transmission time, or that rule R3 of TSN does not allow a packet to start transmitting if it would finish later than the next closing event of P_i's gate, i.e., after gc_i^{next}. ◀

▶ **Corollary 21.** *$A^{\min}(v_p)$ and $A^{\max}(v_p)$ calculated with Eq.* (15) *and* (16) *are exact lower and upper-bounds on the availability time of the egress if P_i start transmitting in the j^{th} interval of $\mathcal{E}_{P_i}(v_k)$.*

6 Merge of System States

As said in Section 3.2, to curb the growth of the graph, if there are two system states v_x and v_y with the same set of transmitted packets $\mathcal{P}(v_x) = \mathcal{P}(v_y)$, and with overlapping or contiguous availability intervals, Alg. 1 merges v_x and v_y into an equivalent state v_z (lines 18-22).

▶ **Lemma 22.** *Let v_x and v_y be two states. If $\mathcal{P}(v_x) = \mathcal{P}(v_y)$ and $A^{\min}(v_x) \leq A^{\min}(v_y) \leq A^{\max}(v_x)$, then a state v_z with $\mathcal{P}(v_z) = \mathcal{P}(v_x) = \mathcal{P}(v_y)$, and $A^{\min}(v_z) = A^{\min}(v_x)$ and $A^{\max}(v_z) = \max\{A^{\max}(v_x), A^{\max}(v_y)\}$ covers all execution scenarios covered by v_x and v_y and no execution scenario that is not covered by either v_x or v_y.*

Proof. We first prove that all execution scenarios covered by v_z are also covered by either v_x or v_y. If $A^{\min}(v_x) \leq A^{\min}(v_y) \leq A^{\max}(v_x)$, then the time interval $[A^{\min}(v_x), A^{\max}(v_x)) \cup [A^{\min}(v_y), A^{\max}(v_y))$ is a continuous interval. Thus, there is at least one execution scenario covered by v_x and/or v_y in which all packets in $\mathcal{P}(v_x) = \mathcal{P}(v_y)$ are transmitted and the egress becomes available at any time $t \in [A^{\min}(v_x),\ \max\{A^{\max}(v_x), A^{\max}(v_y)\})$.

We now prove that all execution scenarios covered by v_x and v_y are covered by v_z. In all execution scenarios covered by v_x and v_y, the egress is certainly available at time $\max\{A^{\max}(v_x), A^{\max}(v_y)\}$, and in no execution scenario covered by v_x and v_y, the egress becomes available before $A^{\min}(v_x)$. Since v_z covers all scenario where the egress may become available from $A^{\min}(v_z) = A^{\min}(v_x)$ to $A^{\max}(v_z) = \max\{A^{\max}(v_x), A^{\max}(v_y)\}$, it covers all execution scenarios covered by v_x and v_y. ◀

7 Proof of Correctness

In this section, we establish that Alg. 1 analyzes all possible execution scenarios and returns exact response time bounds for every packet in $\mathcal{P}$.

▶ **Lemma 23.** *If Alg. 1 does not create an edge labeled with packet P_i from system state v_k, then there exists no execution scenario where P_i is the next packet transmitted in v_k.*

Proof. If Algorithm 1 does not create an edge labeled with packet P_i from system state v_k, then either P_i is not possibly eligible, i.e., $t^{\text{lb}}_{P_i}(v_k) > t^{\text{ub}}(v_k)$ (line 7 of Alg. 1) or the set of eligible transmission intervals $\mathcal{E}_{P_i}(v_k)$ is empty (line 9 of Alg. 1). By contradiction, assume that there exists an execution scenario where P_i is the next packet transmitted in system state v_k and $t^{\text{lb}}_{P_i}(v_k) > t^{\text{ub}}(v_k)$ and/or $\mathcal{E}_{P_i}(v_k) = \emptyset$. Then, it contradicts Lemma 12 and/or Theorem 19. ◀

▶ **Lemma 24.** *If Alg. 1 creates an edge labeled with packet P_i originating from system state v_k, then there exists an execution scenario where P_i is the next packet transmitted in v_k.*

Proof. Algorithm 1 creates an edge labeled with packet P_i originating from system state v_k when $t^{\text{lb}}_{P_i}(v_k) \leq t^{\text{ub}}(v_k)$ (line 7 of Alg. 1) and $\mathcal{E}_{P_i}(v_k) \neq \emptyset$ (line 9 of Alg. 1). If $\mathcal{E}_{P_i}(v_k) \neq \emptyset$ then we always have $t^{\text{lb}}_{P_i}(v_k) \leq t^{\text{ub}}(v_k)$ by Eq. (12), thereby making the condition $t^{\text{lb}}_{P_i}(v_k) \leq t^{\text{ub}}(v_k)$ irrelevant to prove the claim. Moreover, Theorem 19 proves that $\mathcal{E}_{P_i}(v_k)$ does not contain any instant at which P_i cannot start transmitting in v_k. Therefore, if $\mathcal{E}_{P_i}(v_k) \neq \emptyset$, there is an execution scenario where P_i starts transmitting in v_k. ◀

▶ **Theorem 25.** *The SAG generated by Alg. 1 contains all reachable system states and the response time bounds calculated by Alg. 1 are exact.*

Proof. We prove the claim by proving that the SAG generated by Algorithm 1 contains all states reachable by the system and does not contain any state that is not reachable by the system. We prove it by induction. Furthermore, for each system state, we prove that the bounds computed by Alg. 1 on the finish times of packets are exact.

Base case. The state v_0 where the egress is free and no packet is transmitted yet is the only possible system state when the system starts and Alg. 1 properly adds it to the SAG.

Induction step. Let v_k be a reachable system state that was already added to the SAG by Alg. 1, Lemmas 23 and 24 prove that Alg. 1 adds a new state v_l connected to v_k by an edge labeled with P_i if and only if there is an execution scenario where P_i starts executing in v_k. Moreover, Lemma 20 and Corollary 21 prove that the bounds computed by Alg. 1 on the finish time of P_i and the resulting availability of the egress are exact. Thus, Alg. 1 only adds a state v_l if it is reachable by the system, and all states reachable from v_k are added to the SAG by Alg. 1.

Applying the induction step until Alg. 1 ends proves the theorem. ◀

8 Evaluation

The analysis we proposed in the previous sections can analyze TSN switches configured such that gates controlled by Time-Aware Shapers (TAS) adhere to any imaginable schedule, including those where multiple gates are open simultaneously, a flexibility that goes beyond what existing response time analyses for TAS can support. Therefore, we claim that the strengths of our analysis are: (1) its flexibility, enabling design exploration of unconventional configurations, and (2) its exactness. We already proved the exactness of the analysis in Section 7. Therefore, in this section, we investigate whether stepping outside conventional boundaries on how TAS gates are configured might bring new benefits since one may question the necessity of such a comprehensive analysis for such a well-studied concept as TAS.

Experimental Setup. We generate systems made of 8 traffic classes (i.e., the default number of classes mentioned in the TSN standard). We assume each traffic class comprises f flows where $f = 2$, $f = 4$, or $f = 8$ depending on the experiments (i.e., there are between 16 and 64 flows equally distributed between the 8 traffic classes). For each experiment, we start by randomly generating a Gate Control List (GCL) for the time-aware shapers. To do so, for each traffic class $\mathcal{C}_a \in \mathcal{C}$, we generate a utilization values U_a using the Emberson and Davis' tool [12] such that the sum of the classes utilization $\sum_{\mathcal{C}_a \in \mathcal{C}} U_a$ is equal to 100% of the switch egress. That is, U_a represents the portion of the egress bandwidth reserved for class $\mathcal{C}_a$. Then, for each class $\mathcal{C}_a \in \mathcal{C}$, we randomly pick a period T_a from the log-uniformly distributed set of values $\{x \times 10^y \mid x \in [1, 9] \wedge y \in [2, 4]\}$. The total duration O_a during which the gate of class $\mathcal{C}_a$ is open within each period of length T_a is then calculated as $O_a = U_a \times T_a$. Given that a class comprises f communication flows, we assume the gate of each class $\mathcal{C}_a$ opens f times within each period T_a, with each gate opening lasting $\frac{O_a}{f}$ time.

We then use the CW-EDF scheduling algorithm [20, 19] to build the time-triggered schedule deciding the exact times at which each gate of each class is opened and closed over the hyperperiod of the traffic classes. This forms our baseline GCL. Note that CW-EDF enforces that no two gates are open simultaneously.

To test alternative non-conventional GCL configurations, we construct variations of the baseline GCL. We adjust the opening times of each gate in the GCL so that it opens when the previous gate in the GCL has not closed yet. We investigate three distinct scenarios, setting the the time two gates are open simultaneously at 50%, 99%, and 100% of the duration of the previous gate's opening. An example of how the GCL would look like after a 50% overlap is added is shown in the bottom part of Figure 4. The initial GCL before adding the 50% overlap is shown in the upper part of the figure.

In our experiments, we assume the communication flows share the same period as their traffic class. To model the fact that communication flow of a class rarely use 100% of the bandwidth reserved for that class, we distribute 80% of each class bandwidth between the

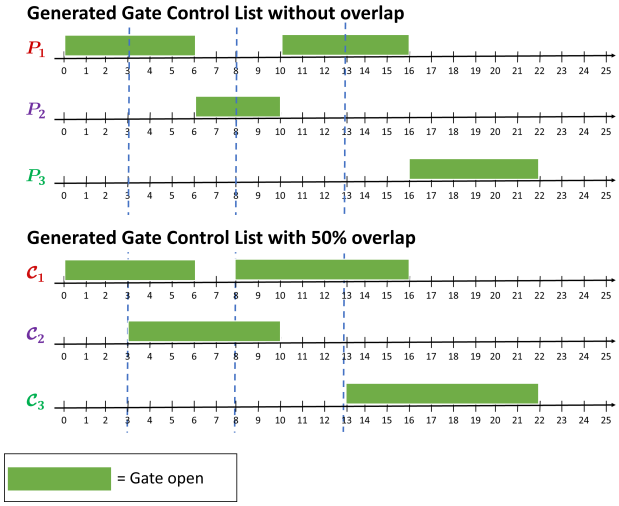


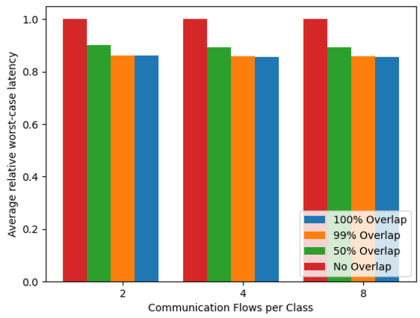
Figure 4 Example that shows how overlapping gates are configured at 50% overlap.

flows of that class using Emberson and Davis' tool [12] and enforcing that the maximum transmission time of each packet does go over the time the class' gate opens. We model the variation in transmission times of the communication flow by assuming that the minimum transmission time is set to 50% of its maximum transmission time.

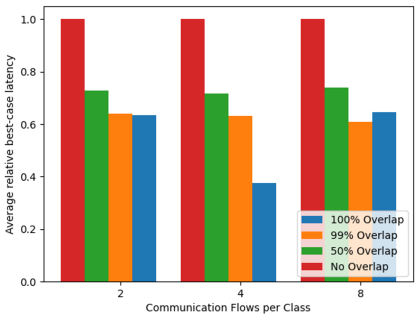
Results. Our experiments are designed to explore whether allowing gates to overlap can lead to lower latency for both best-case and worst-case scenarios in packet transmission. The underlying idea is that in situations where there is uncertainty – be it in the timing of packet arrivals at the switch ingress or in the length of their transmission times – overlapping gates could help some classes benefit from the reserved bandwidth of other classes that might otherwise go unused. By potentially starting the transmission of a packet of a class sooner as another class transmission is concluding, we expect to see a significant drop in the overall time packets spend in the switch queues. To test this hypothesis, we examine how the worst-case and best-case latency change when gates are allowed to overlap.

Figure 5(a)-(d) shows the impact of gate overlap on the latency of packets transmitted through a switch. We investigated two distinct scenarios: the first involves communication flows with varying transmission times but no release jitter (Figure 5(a)-(b)), while the second adds an additional layer of complexity by adding release jitter equal to 5% of each flow's period (Figure 5(c)-(d)). Figure 5(a)-(b) further show the impact of the number of communication flows per traffic class on latency. The plots of Figure 5 are the result of running 40 experiments per scenario. The latency is plotted relative to the latency in the baseline scenario with no overlap, that is, if the average best-case latency is 0.5 for a given scenario, then it means that, in average, the best-case latency of packets was half the best-case latency of the same packets in comparison to the scenario without gate overlap.

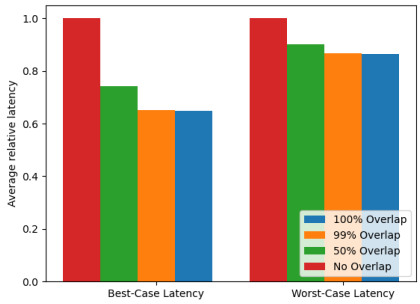
The results across all scenarios are consistent: introducing gate overlap significantly reduces the best-case latency of packets (Figure 5(b)-(d)), but probably more surprisingly, it also improves the worst-case latency of packets (Figure 5(a),(c) and (d)). These results are the first that allow to formally evaluate the cost of using strictly non-overlapping GCLs that under-utilizes the available bandwidth, thereby particularly impacting best-case performance.



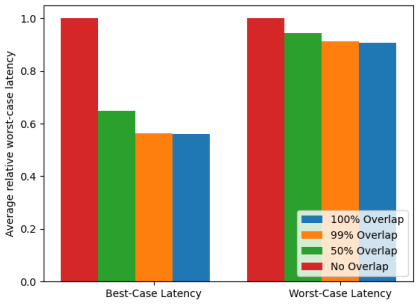
(a) Relative worst-case latency under transmission time variation.



(b) Relative best-case latency under transmission time variation.



(c) Relative best- and worst-case latency when there is release jitter for 2 flows per traffic class.



(d) Relative best- and worst-case latency when there is release jitter for 4 flows per traffic class.

Figure 5 Packets' latency comparison for different gate overlap scenarios.

While we see a clear improvement of the average latency across all classes, we should consider that TAS is usually used to shape traffic classes that deal with high priority or control traffic that cannot tolerate interference by other traffic classes. Strict non-overlapping gates are a way to ensure the absence of such interference. By allowing gate overlaps across traffic classes, we bring the risk to lose that expected property of TAS. Thus, we tested the impact of overlapping gates on the potential additional interference between classes by checking whether the worst-case latency of packets may increase when gate overlaps are allowed. The result of our tests showed that none of the packets experienced a missed deadline even when gates overlap by 100%.

One concern with overlapping gates is that they introduce greater variability in the response times of TAS packets, which is undesirable. Increased variability in a packet's departure from one switch can lead to larger uncertainity bounds on the arrival times at the next switch in the network. The higher the uncertainty, the more difficiult it is to ascertain predictability. However, our analysis remains exact for TAS packets, even when cosnidering jitter in their arrival times. Thus, the SAG method effectively maintains the predictability of TAS packets, even when overlapping gates cause increased variability in the response times.

The fact that we do not see any decrease in the ability to meet transmission deadlines in scenarios with gate overlap suggests a promising TAS configuration strategy where it is possible to ensure that high-priority packets are always sent on time without having to

compromise as much on bandwidth usage. This insight could only be possibly obtained thanks to proposing a new analysis able to test such TAS configuration scenarios. Our new analysis is thus crucial for design-space exploration that may now explore non-conventional TAS gates configurations.

9 Conclusion

We presented the first best-case and worst-case latency analysis for packets shaped with Time-Aware Shapers with no restriction on their GCLs configuration. The analysis is a variation of the SAG approach. We extend the SAG in non-obvious ways to support the analysis of FIFO scheduling and TAS. We proved that the analysis is exact, meaning that it returns tight bounds on the best-case and worst-case latency of each analyzed packet.

Our evaluation section highlights the utility of our new analysis, which allows to analyze out-of-the-box solutions for configuring TAS. In our evaluation section, we extracted just one of potentially many insights on how TAS could be configured to improve best-case and average performance without degrading the worst-case performance of communication flows in comparison to the conventional non-overlaping configuration of TAS. However, there are many different settings and parameters that could be explored with our new analysis to further our understanding of Time-Aware Shapers and potentially reveal more efficient ways to use them. Their exploration is out of the scope of this paper, but our analysis is a first step towards this broader goal, offering a way to explore any possible TAS configuration.

Another extention to this work would be to apply the analysis accross an entire netwrok of swicthes. The best and worst-case latency that we obtain for each packet in a single switch can model the minimum and maximum arrival time of each packet into the next switch in the network, enabling us to analyze the end-to-end latency of packets in a network of switches.

References

1 Time-sensitive networking (TSN) task group. URL: `https://1.ieee802.org/tsn/`.

2 IEEE standard for local and metropolitan area networks–bridges and bridged networks, 2022. IEEE Std 802.1Q-2022 (Revision of IEEE Std 802.1Q-2018).

3 Mohammad Ashjaei, Lucia Lo Bello, Masoud Daneshtalab, Gaetano Patti, Sergio Saponara, and Saad Mubeen. Time-sensitive networking in automotive embedded systems: State of the art and research opportunities. *Journal of systems architecture*, 117:102137, 2021.

4 Neil Audsley, Alan Burns, Mike Richardson, Ken Tindell, and Andy J. Wellings. Applying new scheduling theory to static priority preemptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

5 Lucia Lo Bello, Mohammad Ashjaei, Gaetano Patti, and Moris Behnam. Schedulability analysis of time-sensitive networks with scheduled traffic and preemption support. *Journal of Parallel and Distributed Computing*, 144:153–171, 2020.

6 Unmesh D Bordoloi, Amir Aminifar, Petru Eles, and Zebo Peng. Schedulability analysis of ethernet AVB switches. In *2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 1–10. IEEE, 2014.

7 Jingyue Cao, Pieter JL Cuijpers, Reinder J Bril, and Johan J Lukkien. Independent yet tight WCRT analysis for individual priority classes in ethernet AVB. In *Proc. 24th International Conference on Real-Time Networks and Systems*, pages 55–64, 2016.

8 Joan Adrià Ruiz De Azua and Marc Boyer. Complete modelling of AVB in network calculus framework. In *Proc. 22nd International Conference on Real-Time Networks and Systems*, pages 55–64, 2014.

9 Libing Deng, Guoqi Xie, Hong Liu, Yunbo Han, Renfa Li, and Keqin Li. A survey of real-time ethernet modeling and design methodologies: From AVB to TSN. *ACM Computing Surveys (CSUR)*, 55(2):1–36, 2022.

10 Jonas Diemer, Jonas Rox, and Rolf Ernst. Modeling of ethernet AVB networks for worst-case timing analysis. *IFAC Proceedings Volumes*, 45(2):848–853, 2012.

11 Jonas Diemer, Daniel Thiele, and Rolf Ernst. Formal worst-case timing analysis of ethernet topologies with strict-priority and AVB switching. In *7th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 1–10. IEEE, 2012.

12 Paul Emberson, Roger Stafford, and Robert I Davis. Techniques for the synthesis of multiprocessor tasksets. In *WATERS*, pages 6–11, 2010.

13 Voica Gavriluţ and Paul Pop. Scheduling in time sensitive networks (TSN) for mixed-criticality industrial applications. In *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*, pages 1–4. IEEE, 2018.

14 Voica Gavriluţ and Paul Pop. Traffic-type assignment for TSN-based mixed-criticality cyber-physical systems. *ACM Transactions on Cyber-physical Systems*, 4(2):1–27, 2020.

15 Pourya Gohari, Jeroen Voeten, and Mitra Nasri. Reachability-based Response-Time Analysis of Preemptive Tasks under Global Scheduling. In *36th Euromicro Conference on Real-Time Systems (ECRTS 2024)*, pages 3:1–3:23, 2024.

16 Xiaoting Li and Laurent George. Deterministic delay analysis of AVB switched ethernet networks using an extended trajectory approach. *Real-Time Systems*, 53(1):121–186, 2017.

17 Rouhollah Mahfouzi, Amir Aminifar, Soheil Samii, Ahmed Rezine, Petru Eles, and Zebo Peng. Stability-aware integrated routing and scheduling for control applications in ethernet networks. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 682–687. IEEE, 2018.

18 Mitra Nasri and Björn B. Brandenburg. An Exact and Sustainable Analysis of Non-preemptive Scheduling. In *RTSS*, pages 12–23, 2017.

19 Mitra Nasri and Björn B. Brandenburg. Offline equivalence: A non-preemptive scheduling technique for resource-constrained embedded real-time systems (outstanding paper). In *2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 75–86, 2017. doi:10.1109/RTAS.2017.34.

20 Mitra Nasri and Gerhard Fohler. Non-work-conserving non-preemptive scheduling: motivations, challenges, and potential solutions. In *ECRTS*, pages 165–175, 2016.

21 Mitra Nasri, Geoffrey Nelissen, and Björn B. Brandenburg. A Response-Time Analysis for Non-Preemptive Job Sets under Global Scheduling. In *ECRTS*, pages 9:1–9:23, 2018.

22 Mitra Nasri, Geoffrey Nelissen, and Björn B. Brandenburg. Response-time analysis of limited-preemptive parallel DAG tasks under global scheduling. In *ECRTS*, pages 21:1–21:23, 2019.

23 Geoffrey Nelissen, Joan Marce-i Igual, and Mitra Nasri. Response-Time Analysis for Non-Preemptive Periodic Moldable Gang Tasks. In *ECRTS*, pages 12:1–12:22, 2022.

24 Suhail Nogd, Geoffrey Nelissen, Mitra Nasri, and Björn B. Brandenburg. Response-Time Analysis for Non-Preemptive Global Scheduling with FIFO Spin Locks. In *RTSS*, pages 115–127, 2020.

25 Maryam Pahlevan and Roman Obermaisser. Genetic algorithm for scheduling time-triggered traffic in time-sensitive networks. In *2018 IEEE 23rd international conference on emerging technologies and factory automation (ETFA)*, volume 1, pages 337–344. IEEE, 2018.

26 Rene Queck. Analysis of ethernet AVB for automotive networks using network calculus. In *2012 IEEE International Conference on Vehicular Electronics and Safety (ICVES 2012)*, pages 61–67. IEEE, 2012.

27 Michael Lander Raagaard and Paul Pop. Optimization algorithms for the scheduling of IEEE 802.1 time-sensitive networking (TSN). tech. univ. denmark, lyngby. Technical report, Denmark, Tech. Rep, 2017.

28 Sayra Ranjha, Geoffrey Nelissen, and Mitra Nasri. Partial-Order Reduction for Schedule-Abstraction-based Response-Time Analyses of Non-Preemptive Tasks. In *RTAS*, pages 121–132, 2022.

29 Daniel Thiele, Rolf Ernst, and Jonas Diemer. Formal worst-case timing analysis of ethernet TSN's time-aware and peristaltic shapers. In *2015 IEEE Vehicular Networking Conference (VNC)*, pages 251–258. IEEE, 2015.

30 Luxi Zhao, Paul Pop, and Silviu S Craciunas. Worst-case latency analysis for IEEE 802.1 qbv time sensitive networks using network calculus. *IEEE Access*, 6:41803–41815, 2018.

31 Luxi Zhao, Paul Pop, Zhong Zheng, Hugo Daigmorte, and Marc Boyer. Latency analysis of multiple classes of AVB traffic in TSN with standard credit behavior using network calculus. *IEEE Transactions on Industrial Electronics*, 68(10):10291–10302, 2020.

32 Yimi Zhao, Srinidhi Srinivasan, Geoffrey Nelissen, and Mitra Nasri. Work-in-progress: Generating counter-examples to schedulability using the schedule abstraction. In *2023 IEEE Real-Time Systems Symposium (RTSS)*, pages 459–462. IEEE, 2023.