

Analysis of TSN Credit-Based Shapers using the Schedule Abstraction Graph Methodology

Srinidhi Srinivasan*, Geoffrey Nelissen*, Mario Günzel†, and Nirvana Meratnia*

*Eindhoven University of Technology, The Netherlands

†Max Planck Institute for Software Systems (MPI-SWS), Germany

*{s.srinivasan1, g.r.r.j.p.nelissen, n.meratnia}@tue.nl, †mguenzel@mpi-sws.org

Abstract—IEEE Time-Sensitive Networking (TSN) is widely adopted to support time-critical communication in data-intensive safety- and mission-critical domains such as autonomous driving and smart manufacturing. Among its mechanisms, the Credit-Based Shaper (CBS) regulates when packets may be forwarded by adjusting credits based on traffic activity. This paper presents a new worst-case latency analysis for CBS using the Schedule Abstraction Graph (SAG) framework. We show how the SAG framework can be extended to model CBS. Our results demonstrate that this extended SAG-based analysis may use system-level information about the traffic to improve the accuracy of the schedulability and worst-case latency analysis of CBS.

I. INTRODUCTION

Modern cyber-physical systems, such as autonomous vehicles, warehouse robots, and avionics, exchange safety-critical data with strict timing and reliability requirements. Since conventional Ethernet cannot guarantee bounded latency under load, IEEE 802.1 Time-Sensitive Networking (TSN) [1] extends Ethernet with mechanisms for real-time traffic transmission. In TSN, communication flows are grouped into traffic classes by priority, and traffic shaping controls when and how much each class may transmit to keep latency bounded. Two TSN traffic shapers are the Time-Aware Shaper (TAS) and the Credit-Based Shaper (CBS). TAS uses a time-triggered gate schedule to give critical traffic exclusive link access. CBS instead uses credits: a class gains credit while waiting and spends credit while transmitting, preventing unbounded bursts and limiting delay.

Several worst-case analysis techniques derive safe latency upper bounds for CBS-shaped traffic. These include Network Calculus [2], [3], [4], Compositional Performance Analysis (CPA) [5], [6], eligibility-interval analyses [7], the trajectory approach [8], and conventional response-time analyses [9], [10]. Related work has analysed TAS using Network Calculus, CPA, and the Schedule Abstraction Graph (SAG) [11], [12], [13]. Recent work extends eligibility-interval analysis to jointly handle TAS, CBS, and preemptive interference [14], while Network-Calculus-based methods have also integrated multiple TSN mechanisms [4]. Beyond TSN, worst-case delay analyses for switched Ethernet and AFDX networks have used Network Calculus, offset-based scheduling, and timed automata [15], [16], [17], [18], [19]. [20] proposes an UPPAAL-based approach to analyze TSN traffic shapers but is limited to deterministic systems without uncertainty in arrival or packet transmission times.

This work focuses on CBS-shaped traffic and compares our approach with state-of-the-art analyses. We assume traffic arrival patterns are known a priori and analyse worst-case latency under these assumptions.

We build on the Schedule Abstraction Graph framework, which has been effective for response-time analysis in real-time task systems [21], [22], [23], [24], [25], [26], [27] and TSN communication [13]. SAG performs state-space exploration but avoids excessive computational cost by focusing only on key decision points. Rather than examining every time instant, the graph advances only to moments at which relevant scheduling decision may occur. This selective exploration, combined with pruning and partial order reduction techniques, greatly reduces unnecessary states exploration. We modify the SAG analysis framework to explore properties that matter for the analysis of CBS, particularly the evolution of credit of the traffic classes at each egress port of TSN switches. We present the theoretical framework for this analysis and techniques to keep the model compact and scalable. Our evaluation shows that the proposed analysis achieves significantly higher schedulability ratios than existing methods while supporting multiple CBS classes and their interactions.

Section II, introduces the system model, followed by a high-level overview of the analysis methodology in Section III. Section IV presents how eligible packets are identified for graph construction, and Section V describes how new states are generated. To manage the state space, Section VI explains our state-merging strategy. Section VII establishes correctness, and Section VIII discusses scalable implementation techniques, Section IX reports the experimental evaluation, and Section X concludes the paper and discusses future work.

II. SYSTEM MODEL

This section defines the system model used in our analysis. We consider Time-Sensitive Networking (TSN) switches with multiple full-duplex communication streams, each connecting an ingress port to an egress port.

Each egress port maintains one queue per traffic class. Communication flows are assigned to a traffic class and, upon arrival, are placed in the queue corresponding to their destination egress port and class. We assume that each queue is shaped by a Credit-Based Shaper (CBS), which regulates packet transmission through the egress port. Figure 1 illustrates this architecture.

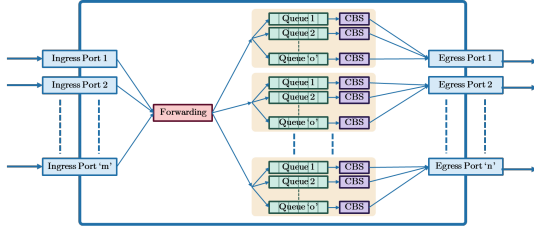


Fig. 1: Switch architecture with m ingress ports, n egress ports, and o queues per egress port (one per traffic class)

a) *Traffic Classes*: Let $\mathcal{C} = \{C_1, \dots, C_{|\mathcal{C}|}\}$ be the set of traffic classes in each TSN switch. Each class has a unique priority, and each egress port has one queue per class. The transmission of each queue is controlled by a credit-based shaper, which uses a dynamically evolving credit value to determine when packets can be sent. The credit of a class increases when the queue is not transmitting and decreases during transmission. This is governed by the idle slope, the credit replenishment rate, and the send slope, the credit consumption rate. A class may transmit only when its credit is non-negative, which regulates bandwidth usage and provides fair link access across classes.

The queue of class C_a (for $1 \leq a \leq |\mathcal{C}|$) at egress port p is defined as $Q_{p,a} = (\pi_{C_a}, \alpha_{p,a}^+)$, where $\pi_{C_a} \in \mathbb{N}_{\geq 0}$ is the class priority, with lower values denoting higher priority, and $\alpha_{p,a}^+ > 0$ is the idle slope. When a queue has negative credit or waits to transmit a pending packet, its credit increases at rate $\alpha_{p,a}^+$. If the queue has no pending packet once its credit becomes non-negative, the accumulated credit is reset to 0.

To ensure that all classes can be serviced without starvation, the following condition $\sum_{a=1}^{|\mathcal{C}|} \alpha_{p,a}^+ \leq 1$ must hold. The send slope (i.e., credit consumption rate) $\alpha_{p,a}^-$ of the queue of class C_a of port p is given as $\alpha_{p,a}^- = \alpha_{p,a}^+ - 1$. When a packet of class C_a is transmitted, the queue's credit decreases at rate $\alpha_{p,a}^-$.¹

b) *Flows and Packets*: A finite set of flows $\mathcal{F} = \{F_1, \dots, F_{|\mathcal{F}|}\}$ is transmitted through the switch. Each flow $F_i \in \mathcal{F}$ is assigned to a traffic class C_a and releases a sequence of packets $P_{i,j}$, where $j \in \mathbb{N}$ denotes the j -th packet of the flow. Each flow has a period T_i and may release multiple packets at different times within each period. The same packet arrival pattern repeats every period, with the earliest and latest arrival times shifted by T_i . The hyperperiod H is the least common multiple of the flow periods. The set $\mathcal{P}$ contains all packets needed for the analysis from its start to the end of a safe observation window. Section VIII-B explains how to determine a safe bound on this window. Each packet $P_{i,j}$ is specified by

$$P_{i,j} = (a_{P_{i,j}}^{\min}, a_{P_{i,j}}^{\max}, d_{P_{i,j}}, \ell_{P_{i,j}}^{\min}, \ell_{P_{i,j}}^{\max}).$$

¹Note that to reduce notation clutter, from now on, we will omit to specify the index p of the egress port when referring to the idle slope, send slope, and other parameters. Since all properties discussed hereafter always refer to a single egress port, it does not generate any ambiguity.

Here, $a_{P_{i,j}}^{\min}, a_{P_{i,j}}^{\max} \in \mathbb{R}$, with $a_{P_{i,j}}^{\max} \geq a_{P_{i,j}}^{\min}$, are the earliest and latest absolute arrival times at the switch, $d_{P_{i,j}} \in \mathbb{R}$ is the absolute deadline for completing transmission through the egress port, and $\ell_{P_{i,j}}^{\min}, \ell_{P_{i,j}}^{\max} \in \mathbb{R}_{>0}$, with $\ell_{P_{i,j}}^{\min} \leq \ell_{P_{i,j}}^{\max}$, are lower and upper bounds on packet length in bytes. The corresponding minimum and maximum transmission durations (seconds), $C_{P_{i,j}}^{\min} \in \mathbb{R}_{\geq 0}$ and $C_{P_{i,j}}^{\max} \in \mathbb{R}_{>0}$ (with $C_{P_{i,j}}^{\min} \leq C_{P_{i,j}}^{\max}$), are derived from these bounds as follows:

$$C_{P_{i,j}}^{\min} = \frac{\ell_{P_{i,j}}^{\min}}{\rho} \quad \text{and} \quad C_{P_{i,j}}^{\max} = \frac{\ell_{P_{i,j}}^{\max}}{\rho}, \quad (1)$$

where $\rho \in \mathbb{R}_{\geq 0}$ is the egress port transmission rate in bytes per second.

c) *Transmission Scenario*: Since only bounds are specified for each packet's arrival time and transmission duration, there is uncertainty about when a packet may arrive and how long it may take to transmit. Moreover, credit evolution in CBS depends on transmission ordering and accumulated credit history, and there is no known critical instant that trivially yields the worst-case behaviour. As a result, different combinations of arrival times and lengths within these bounds can lead to completely different system behaviours with no means to know a priori which will yield the worst-case. To ensure that the analysis captures all possible behaviours, we must consider every valid combination of these values. The complete set of such combinations is formalised using the notion of transmission scenarios, each representing one possible concrete realisation of packet timing and length.

Definition II.1 (Transmission Scenario). A transmission scenario γ is defined by a specific arrival time $a_{P_{i,j}}$ and packet length $C_{P_{i,j}}$ for each packet $P_{i,j} \in \mathcal{P}$, namely

$$\gamma = \{(P_{i,j}, a_{P_{i,j}}, C_{P_{i,j}}) \mid P_{i,j} \in \mathcal{P}\} \quad (2)$$

with $a_{P_{i,j}} \in [a_{P_{i,j}}^{\min}, a_{P_{i,j}}^{\max}]$ denoting a concrete release time and $C_{P_{i,j}} \in [C_{P_{i,j}}^{\min}, C_{P_{i,j}}^{\max}]$ denoting a concrete transmission duration for $P_{i,j}$.

We assume that if multiple packets arrive at the same time at a queue, their order in the FIFO queue is consistently determined by a possibly unknown but systematic and deterministic tie-breaking rule (e.g., based on packet, flow or ingress port index). Hence, for a given transmission scenario γ , the execution of CBS is deterministic. **Since this tie-breaking rule may not be known a priori, by default, the analysis considers all possible tie-breaking outcomes, i.e., all corresponding transmission scenarios.**

Let $f_{P_{i,j}}(\gamma)$ denote the time at which packet $P_{i,j}$ finishes its transmission in transmission scenario γ . Since CBS is a deterministic algorithm, $f_{i,j}(\gamma)$ is uniquely defined, yet possibly different for each scenario γ .

Let Γ represent the complete collection of possible transmission scenarios. The set of packets is considered schedulable if no transmission scenario γ in Γ involves a packet missing its deadline, i.e., $\forall \gamma \in \Gamma, f_{P_{i,j}}(\gamma) \leq d_{P_{i,j}}$. Conversely, if at least one packet in any scenario does not meet its deadline, the set is said to be unschedulable, i.e., $\exists \gamma \in \Gamma, f_{P_{i,j}}(\gamma) > d_{P_{i,j}}$. We define the latency of a packet $P_{i,j}$ in scenario γ as the time

from its earliest possible arrival to its finish time in γ , i.e., $f_{P_{i,j}}(\gamma) - a_{P_{i,j}}^{\min}$, and we define the best-case and worst-case latency of $P_{i,j}$ as $\text{BCL}_{P_{i,j}} = \min_{\gamma \in \Gamma} \{f_{P_{i,j}}(\gamma) - a_{P_{i,j}}^{\min}\}$ and $\text{WCL}_{P_{i,j}} = \max_{\gamma \in \Gamma} \{f_{P_{i,j}}(\gamma) - a_{P_{i,j}}^{\min}\}$, respectively.

III. ANALYSIS METHODOLOGY

To determine schedulability, we explore all valid combinations of packet arrivals, transmission durations, and CBS scheduling decisions. Each transmission updates the system state, and exhaustive exploration of all reachable states yields safe best- and worst-case latency bounds. We perform this efficiently using the Schedule Abstraction Graph (SAG) methodology [22], [21], [13], which we extend here to CBS-shaped traffic.

The SAG is a directed acyclic graph whose vertices represent abstract system states and whose edges represent transitions caused by packet transmissions. Construction starts from an initial state with empty queues. Whenever the scheduler may choose a packet to transmit, all feasible choices are explored, producing successor states. This continues until no new states are generated, resulting in a graph that captures all valid transmission scenarios and enables derivation of safe latency bounds. Each vertex v_m corresponds to an abstract state S_m , and an edge from S_m to $\bar{S}$ labelled by $P_{i,j}$ means that transmitting $P_{i,j}$ in S_m leads to $\bar{S}$.

Each abstract state S_m stores:

- 1) the set $\mathcal{T}$ of packets already transmitted in the scenarios represented by S_m ;
- 2) a time interval $[A^{\min}, A^{\max}]$, where $A^{\min}$ lower-bounds the earliest possible time the egress port may become available, and $A^{\max}$ upper-bounds the earliest time by which availability is guaranteed;
- 3) for each class C_a , a set ϵ_{C_a} of possible finish-time/credit pairs (t, c) for the most recently transmitted packet of that class; if no packet of that class has been transmitted, then $\epsilon_{C_a} = \{(0, 0)\}$.

Thus, S_m represents all concrete states in which the same packet set $\mathcal{T}$ has been transmitted, the egress port availability lies in $[A^{\min}, A^{\max}]$, and the class credits lie in the corresponding sets ϵ_{C_a} .

Example III.1. Figure 2 illustrates how graph states are built. The switch has two traffic classes, C_1 and C_2 , with C_1 having higher priority. The packet set contains four packets, each assigned to one of the two classes. For state S_m to be reachable, packets $P_{1,1}$ and $P_{2,1}$ must already have been transmitted, so $\mathcal{T} = \{P_{1,1}, P_{2,1}\}$. After these transmissions, the state may become available at time 5 and is certainly available at time 6, giving $[A^{\min}, A^{\max}] = [5, 6]$.

For each class, ϵ_{C_a} is shown as a polygon in the credit-time plane. Consider C_a , which has transmitted $P_{1,1}$. The red polygon contains all possible credit-time pairs at the finish times of $P_{1,1}$. Since $P_{1,1}$ may arrive between time 0 and 1 and transmit for 2 to 3 time units, it can finish at different times with different credits, depending on the send slope. These possible completion points form the polygon. Similarly,

Algorithm 1 Schedule Abstraction Graph Construction

Require: Packet set $\mathcal{P}$, and $\forall C_a \in \mathcal{C}, \pi_{C_a}$ and α_a^+

```

1: for each packet  $P_{i,j} \in \mathcal{P}$  do
2:    $\text{BCL}_{P_{i,j}} \leftarrow \infty$  ▷ Initialize the BCL for  $P_{i,j}$ 
3:    $\text{WCL}_{P_{i,j}} \leftarrow 0$  ▷ Initialize the WCL for  $P_{i,j}$ 
4: end for
5:  $\mathcal{S}_0 \leftarrow \{\mathcal{T} = \emptyset, [A^{\min}, A^{\max}] = [0, 0], \epsilon_{C_a} = (0, 0) \ \forall C_a \in \mathcal{C}\}$ 
   ▷ Initialise the system state
6:  $L^{\text{cur}} \leftarrow \{\mathcal{S}_0\}$  ▷ Initialise current layer of the SAG
7:  $L^{\text{next}} \leftarrow \emptyset$  ▷ Initialise next layer of the SAG
8: while  $L^{\text{cur}} \neq \emptyset$  do
9:   for each state  $S_m \in L^{\text{cur}}$  do
10:    Find the set of eligible packets ▷ Refer to Section IV
11:    for each eligible packet  $P_{i,j}$  do
12:      Compute  $f_{P_{i,j}}^{\min}$  and  $f_{P_{i,j}}^{\max}$  using Eq. 17
13:       $\text{BCL}_{P_{i,j}} \leftarrow \min\{f_{P_{i,j}}^{\min} - a_{P_{i,j}}^{\min}, \text{BCL}_{P_{i,j}}\}$ 
14:       $\text{WCL}_{P_{i,j}} \leftarrow \max\{f_{P_{i,j}}^{\max} - a_{P_{i,j}}^{\min}, \text{WCL}_{P_{i,j}}\}$ 
15:      ▷ Update BCL and WCL
16:      if  $f_{P_{i,j}}^{\max} > d_{P_{i,j}}$  then
17:        return Unschedulable
18:      end if
19:      Create a new state  $\bar{S}$  ▷ Refer to Section V
20:      if  $\exists S_k \in L^{\text{next}}$  that can be merged with  $\bar{S}$  then
21:        Merge  $S_k$  and  $\bar{S}$  ▷ Refer to Section VI
22:      else
23:         $L^{\text{next}} \leftarrow L^{\text{next}} \cup \{\bar{S}\}$ 
24:      end if
25:    end for
26:  end for
27:   $L^{\text{cur}} \leftarrow L^{\text{next}}$ 
28:   $L^{\text{next}} \leftarrow \emptyset$ 
29: end while
30: return Schedulable

```

extending the graph from S_m to $\bar{S}$ by transmitting $P_{1,2}$ updates the relevant credit region. The following sections explain this transition in detail. $\square$

Although the polygons in the example are drawn as bounded, the theoretical credit-time regions may contain infinitely many pairs. While this could raise scalability concerns, Section VIII explains how we interpret and store these regions so that the overall analysis remains efficient and scalable.

Algorithm 1 constructs the SAG from the packet set $\mathcal{P}$, class priorities, and idle slopes.

Initialisation Phase (lines 1-9 of Algorithm 1): For each packet in $\mathcal{P}$, the Best-Case Latency (BCL) is initialised to ∞ and the Worst-Case Latency (WCL) to 0. The SAG begins with the initial state $\mathcal{S}_0$ that represents the state where no packet has been transmitted yet, so the set of transmitted packets is empty, i.e., $\mathcal{T} = \emptyset$, the egress port is possibly and certainly available at time 0, so $[A^{\min}, A^{\max}] = [0, 0]$, and every traffic class starts with credit 0, hence $\epsilon_{C_a} = \{(0, 0)\}$.

The SAG is built breadth-first, one layer at a time. To avoid storing the full graph, the algorithm keeps only L^{cur} , containing the states to be explored in the current layer, and L^{next} , containing the states generated for the next layer. The construction alternates between expansion, which generates successor states, and merging, which combines compatible states to limit growth.

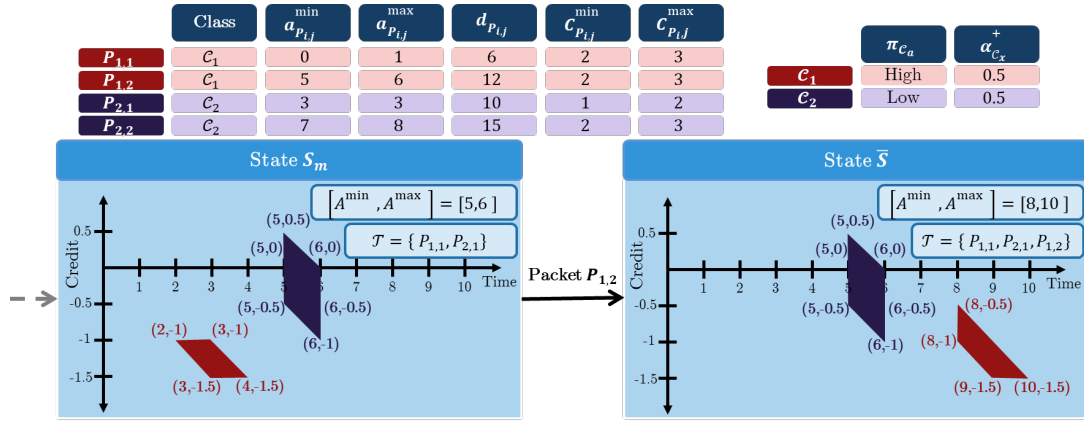


Fig. 2: State Transition Diagram for Example III.1

Expansion Phase (lines 10–21 of Algorithm 1): For each state in L^{cur} , the algorithm identifies all possible next scheduling decisions, as described in Section IV. Each valid decision generates a successor state which is added to L^{next} , as explained in Section V. The best-case latency (BCL) and worst-case latency (WCL) of each newly scheduled packet are updated, and the analysis continues unless a deadline miss is found.

Merge Phase (lines 22–26 of Algorithm 1): Before adding a new state to L^{next} , the algorithm checks whether it can be merged with an existing state by comparing their contents. The merge rules are given in Section VI.

IV. ELIGIBLE PACKETS

The main step in the expansion phase is to identify which packets may be transmitted next. A packet can be eligible only if it is at the head of its FIFO queue and its traffic class has non-negative credit. However, these two conditions alone do not guarantee that the packet will be scheduled next. We must also ensure that no higher-priority packet becomes eligible earlier, and that no other packet is already able to start transmission sooner due to being ready and having access to the egress port. This section combines these conditions to derive the full set of eligible packets.

A. Accounting for FIFO Queues

Each traffic class has a FIFO queue, and a packet must be at the head of this queue to be possibly eligible for transmission. To identify the packets that may be at the head of class C_a in abstract state S_m , we first bound the time at which the head packet entered the queue. These bounds are denoted by $tr_{C_a}^{\min}$ and $tr_{C_a}^{\max}$. We then prove that every packet that could possibly arrive before $tr_{C_a}^{\max}$ has a chance to be at the top of the queue.

Lemma 1. If the system is in abstract state S_m , then for any concrete state represented by S_m , the earliest time when a not yet transmitted packet of class C_a may possibly have entered its FIFO queue is lower bounded by

$$tr_{C_a}^{\min} = \min\{a_{P_{i,j}}^{\min} \mid P_{i,j} \notin \mathcal{T}, P_{i,j} \text{ is of } C_a\} \quad (3)$$

Proof. Only packets of class C_a that were not transmitted yet can still be in C_a 's FIFO queue in every concrete state in the abstract state S_m , that is, only packets in $C_a \setminus \mathcal{T}$. Since $a_{P_{i,j}}^{\min}$ denotes the earliest possible arrival time of a packet $P_{i,j}$ at the switch, the earliest time a packet in $C_a \setminus \mathcal{T}$ may possibly arrive at the switch and thus enter its FIFO queue is $\min\{a_{P_{i,j}}^{\min} \mid P_{i,j} \notin \mathcal{T}, P_{i,j} \text{ is of } C_a\}$. $\square$

Similarly, we bound when a packet of class C_a is *certainly* present in its FIFO queue.

Lemma 2. If the system is in abstract state S_m , then for any concrete state represented by S_m , the earliest time at which a not yet transmitted packet of class C_a will *certainly* enter its FIFO queue is upper bounded by

$$tr_{C_a}^{\max} = \min\{a_{P_{i,j}}^{\max} \mid P_{i,j} \notin \mathcal{T}, P_{i,j} \text{ is of } C_a\} \quad (4)$$

Proof. Only packets of class C_a that were not transmitted yet can still be in C_a 's FIFO queue for any concrete state in abstract state S_m , that is, only packets in $C_a \setminus \mathcal{T}$. Since $a_{P_{i,j}}^{\max}$ is an upper bound on the arrival time of $P_{i,j}$ at the switch, at least one packet in $C_a \setminus \mathcal{T}$ must certainly have arrived at the switch and entered the FIFO queue at $\min\{a_{P_{i,j}}^{\max} \mid P_{i,j} \notin \mathcal{T}, P_{i,j} \text{ is of } C_a\}$. $\square$

We use $tr_{C_a}^{\max}$ to determine the set top_{C_a} of packets that may possibly be at the head of class C_a 's FIFO queue.

Theorem 1. If the system is in abstract state S_m , then for any concrete state represented by S_m , the packet at the top of C_a 's FIFO queue must belong to the set

$$top_{C_a} = \{P_{i,j} \mid P_{i,j} \notin \mathcal{T}, P_{i,j} \text{ is of } C_a \wedge a_{P_{i,j}}^{\min} \leq tr_{C_a}^{\max}\} \quad (5)$$

Proof. Only packets that have not yet been transmitted (i.e., those not in $\mathcal{T}$) can appear at the head of the FIFO queue. Furthermore, since by Lemma 2, $tr_{C_a}^{\max}$ is the earliest time at which a packet of class C_a is guaranteed to enter the FIFO queue, any packet whose earliest possible arrival time $a_{P_{i,j}}^{\min}$ is greater than $tr_{C_a}^{\max}$ will enter the FIFO after another packet and thus cannot be at the top of the queue. Conversely, if a packet $P_{i,j}$ may possibly arrive before or at $tr_{C_a}^{\max}$ (i.e., $a_{P_{i,j}}^{\min} \leq tr_{C_a}^{\max}$) then it can arrive before any other packet and thus

reach the top of the queue. Combining those two properties, we have that the set of packets that can be at the head of C_a 's FIFO queue for any concrete state in abstract state $\mathcal{S}_m$ is $\{P_{i,j} \mid P_{i,j} \notin \mathcal{T}, P_{i,j} \text{ is of } C_a \wedge a_{P_{i,j}}^{\min} \leq tr_{C_a}^{\max}\}$. $\square$

Eq. (5) does not use any information about the tie-breaking rule for simultaneous arrivals. Therefore, SAG assumes that no such information is available and considers all possible orderings of tied packets. Consequently, every packet that could be placed first under at least one possible ordering is included in top_{C_a} . If the switch is deterministic and its tie-breaking policy is known, this information can be incorporated into Eq. (5) to further restrict top_{C_a} by excluding packets that cannot be selected first due to the tie-breaker. Doing so prunes impossible SAG branches (i.e., paths in the state-space), thus reducing pessimism and increasing the schedulability ratio. This refinement applies only when two or more packets of C_a satisfy $a_{P_{i,j}}^{\min} = tr_{C_a}^{\max}$, in which case the tie-breaking policy determines which of these packets can be placed first in the FIFO queue.

Once possible head packets are identified, we determine whether each may begin transmission before any other packet. For each $P_{i,j} \in top_{C_a}$, we compute lower and upper bounds on its possible start time, assuming it is the first packet transmitted in $\mathcal{S}_m$. We then prove that $P_{i,j}$ may begin transmission before any other packet in abstract state $\mathcal{S}_m$ only if the lower bound is smaller than or equal to the upper bound.

B. Transmission Start Time Bounds

The start time of a packet in top_{C_a} depends on egress-port availability, the time at which its class credit becomes non-negative, and the presence of eligible higher-priority packets.

We proceed in four steps. First, we define a function that gives when a class credit becomes non-negative. Second, we derive a lower bound on the start time of any packet of class C_a . Third, we derive an upper bound assuming C_a is the first class to transmit in $\mathcal{S}_m$. Finally, we combine these results to identify eligible packets and their finish-time bounds.

Non-negative Credit Attainment Function: As defined in Section II, ϵ_{C_a} contains all possible credit values of class C_a at all possible finish times of its last transmitted packet. From this set, we derive the earliest and latest times at which the class credit may become non-negative after its last transmission.

We define $zc_{C_a}(t, c)$ for a time-credit pair (t, c) , where t is a time instant and c is the credit at that instant, as the earliest time no earlier than t at which the credit of C_a becomes non-negative, under the assumption that the credit at time t is c :

$$zc_{C_a}(t, c) = \begin{cases} t & \text{if } c \geq 0, \\ t - \frac{c}{\alpha_a^+} & \text{otherwise.} \end{cases} \quad (6)$$

where α_a^+ is the idle slope of C_a .

Applying zc_{C_a} to all pairs in ϵ_{C_a} gives the set of possible times at which C_a 's credit becomes non-negative in any transmission scenario that led to the abstract system abstract

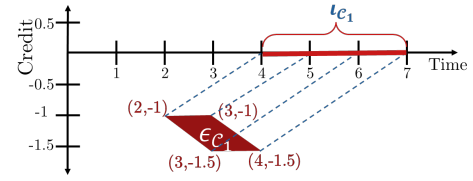


Fig. 3: Example showing the calculation of $t_{C_a}^+$

state $\mathcal{S}_m$. We call this set of time instants $t_{C_a}^+$. Formally, we define

$$t_{C_a}^+ = \{t' \mid \exists (t, c) \in \epsilon_{C_a}, t' = zc_{C_a}(t, c)\} \quad (7)$$

We can then define the earliest time $t_{C_a}^{+, \min}$ and latest time $t_{C_a}^{+, \max}$ at which C_a 's credit becomes non-negative in abstract state $\mathcal{S}_m$ as follows.

$$t_{C_a}^{+, \min} = \min\{t \mid t \in t_{C_a}^+\} \quad (8)$$

$$t_{C_a}^{+, \max} = \max\{t \mid t \in t_{C_a}^+\} \quad (9)$$

Example IV.1. Figure 3 illustrates the transformation from ϵ_1 to t_1 , using the state-transition example from Fig. 2. The set ϵ_1 contains all points in the red region. For each point below the x-axis, zc_{C_a} projects it to the x-axis along a line whose slope equals the idle slope of C_1 . This gives $t_1 = [4, 7]$, so C_1 's credit becomes non-negative earliest at $t_{C_a}^{+, \min} = 4$ and latest at $t_{C_a}^{+, \max} = 7$. $\square$

Lower Bound on Start of Transmission of Class C_a : The earliest time at which class C_a can begin the transmission of any packet in abstract state $\mathcal{S}_m$ is determined by three factors: the earliest possible time $tr_{C_a}^{\min}$ at which a packet of that class may enter its FIFO queue, the earliest time $A^{\min}$ at which the egress port becomes available in $\mathcal{S}_m$, and the earliest time $t_{C_a}^{+, \min}$ at which C_a 's credit becomes non-negative.

Lemma 3. For any traffic class C_a , and for all concrete state represented by the abstract state $\mathcal{S}_m$, the start time of transmission of any packet of that class is lower bounded by

$$t_{C_a}^{\text{lb}} = \max\{tr_{C_a}^{\min}, A^{\min}, t_{C_a}^{+, \min}\}. \quad (10)$$

Equivalently, no packet of class C_a in any concrete state in the abstract state $\mathcal{S}_m$ can start transmission before time $t_{C_a}^{\text{lb}}$.

Proof. For a packet of class C_a to start its transmission at time t , three necessary conditions must be satisfied:

- 1) **Packet Arrival:** A packet must be in C_a 's FIFO queue. By Lemma 1, $tr_{C_a}^{\min}$ is the earliest such time. Therefore, the start time t must satisfy $t \geq tr_{C_a}^{\min}$.
- 2) **Egress Port Availability:** The egress port must be available for transmission. Since $A^{\min}$ is the earliest time at which the port is available in $\mathcal{S}_m$, we have $t \geq A^{\min}$.
- 3) **Non-negative Credit:** Credit for class C_a must be non-negative. By definition, $t_{C_a}^{+, \min}$ is the earliest time the credit becomes non-negative. Thus, $t \geq t_{C_a}^{+, \min}$.

Therefore, $t_{C_a}^{\text{lb}} = \max\{tr_{C_a}^{\min}, A^{\min}, t_{C_a}^{+, \min}\}$. Any earlier time would violate at least one of the three conditions above. $\square$

Upper Bound on Start of Transmission of Class C_a :

Assuming C_a is the first class transmitting a packet in any concrete state in abstract state $\mathcal{S}_m$, the latest time this transmission may start is upper bounded by two critical factors. First, if any other class becomes eligible for transmission earlier than C_a , then C_a cannot be the next to transmit since the switch always transmits the earliest eligible packet. Second, if any class with a higher priority than C_a becomes eligible before C_a start transmitting, then C_a loses its opportunity to be the first transmitting.

To formalise these properties, we introduce two key lemmas. In Lemma 5, we define t^{eet} , which captures the earliest time at which some class becomes certainly eligible to transmit, potentially preventing C_a from being selected next. In Lemma 6, we define $t_{C_a}^{\text{high}}$ to capture the effect of higher priority classes becoming eligible, which further limits when C_a can transmit.

As established earlier, a traffic class becomes eligible to transmit a packet only when three conditions are met: there is a packet in its queue, the egress port is available, and the class credit is non-negative. To determine the time at which a class is certainly eligible, we consider the earliest time that each of these conditions is guaranteed to hold.

Lemma 4. For any class C_a and any concrete state represented by the abstract state $\mathcal{S}_m$, the time at which C_a becomes eligible to transmit is upper bounded by $\max\{tr_{C_a}^{\text{max}}, A^{\text{max}}, t_{C_a}^{+, \text{max}}\}$

Proof. The three following conditions must be satisfied:

- 1) Packet Arrival: A packet must be in C_a 's FIFO queue. By Lemma 2, there is certainly a packet in the FIFO queue at $tr_{C_a}^{\text{max}}$.
- 2) Egress Port Availability: The egress port must be available for transmission. By definition, the egress port is certainly available at and after A^{max} in $\mathcal{S}_m$.
- 3) Non-negative Credit: Credit for class C_a must be non-negative. By definition, $t_{C_a}^{+, \text{max}}$ is the latest time C_a 's credit becomes non-negative in $\mathcal{S}_m$.

It results that all three conditions are certainly respected at and after $\max\{tr_{C_a}^{\text{max}}, A^{\text{max}}, t_{C_a}^{+, \text{max}}\}$. $\square$

Lemma 5. For every concrete state represented by the abstract state $\mathcal{S}_m$, there exists a class that is certainly eligible for transmission at or before

$$t^{\text{eet}} = \min_{\forall C_a \in \mathcal{C}} \{\max\{tr_{C_a}^{\text{max}}, A^{\text{max}}, t_{C_a}^{+, \text{max}}\}\} \quad (11)$$

and thus t^{eet} is an upper bound on the time by which some class is certainly eligible.

Proof. By Lemma 4, at least one class $C_a \in \mathcal{C}$ is certainly eligible at $\max\{tr_{C_a}^{\text{max}}, A^{\text{max}}, t_{C_a}^{+, \text{max}}\}$ in $\mathcal{S}_m$. Thus, a class is certainly eligible at $\min_{\forall C_a \in \mathcal{C}} \{\max\{tr_{C_a}^{\text{max}}, A^{\text{max}}, t_{C_a}^{+, \text{max}}\}\}$. $\square$

We now capture the effect of higher priority classes on when C_a may start transmitting for every concrete state in abstract state $\mathcal{S}_m$.

Lemma 6. For every concrete state represented by the abstract state $\mathcal{S}_m$, the time at which a class with higher-priority than C_a becomes eligible is upper bounded by

$$t_{C_a}^{\text{high}} = \min_{\forall C_b \in \mathcal{C} | \pi_{C_b} < \pi_{C_a}} \{\max\{tr_{C_b}^{\text{max}}, t_{C_b}^{+, \text{max}}\}\} \quad (12)$$

Proof. For any class C_b with higher priority than C_a (i.e., $\pi_{C_b} < \pi_{C_a}$), consider the two conditions that do not depend on egress-port availability: a packet of C_b is guaranteed to be at the head of its queue no later than $tr_{C_b}^{\text{max}}$, and its credit is guaranteed to become non-negative no later than $t_{C_b}^{+, \text{max}}$. Once both of these conditions are satisfied, class C_b can transmit as soon as the egress port becomes available. Thus, from time $t_{C_b}^{\text{high}}$ onward, whenever the egress port becomes available, at least one higher-priority class is already ready to transmit. $\square$

We now use the bounds proven in Lemmas 3, 5 and 6 to decide whether a traffic class C_a may possibly be the first to transmit a packet in any concrete state present in the abstract state $\mathcal{S}_m$.

Theorem 2. There may exist a concrete state represented by the abstract state $\mathcal{S}_m$ where class C_a is the first class to transmit a packet, only if

$$t_{C_a}^{\text{lb}} \leq t^{\text{eet}} \wedge t_{C_a}^{\text{lb}} < t_{C_a}^{\text{high}} \quad (13)$$

Proof. By Lemma 5, t^{eet} is the earliest time at which some class is certainly eligible to transmit. Therefore, if C_a is the first class to transmit, it must transmit at or before t^{eet} , implying that it must become eligible at or before t^{eet} . Since, by Lemma 3, $t_{C_a}^{\text{lb}}$ is a lower bound on when C_a becomes eligible in state $\mathcal{S}_m$, we must have $t_{C_a}^{\text{lb}} \leq t^{\text{eet}}$.

Similarly, by Lemma 6, C_a cannot be the highest-priority eligible class from $t_{C_a}^{\text{high}}$ onward in any concrete state in abstract state $\mathcal{S}_m$. Thus, if C_a has not started transmission strictly before $t_{C_a}^{\text{high}}$, it will no longer be chosen by the switch. Hence, to be the first class to start a transmission in $\mathcal{S}_m$, C_a must be eligible strictly before $t_{C_a}^{\text{high}}$, thereby requiring $t_{C_a}^{\text{lb}} < t_{C_a}^{\text{high}}$. $\square$

Eligible Packets: We can now use the properties proven above to decide whether there can be a concrete state represented by the abstract state $\mathcal{S}_m$ in which a packet $P_{i,j}$ is the first packet transmitted.

Theorem 3. There may be a concrete state represented by the abstract state $\mathcal{S}_m$ where a packet $P_{i,j}$ of class C_a is the first packet transmitted only if $P_{i,j} \in \text{top}_{C_a} \wedge \max\{a_{P_{i,j}}^{\text{min}}, t_{C_a}^{\text{lb}}\} \leq t^{\text{eet}} \wedge \max\{a_{P_{i,j}}^{\text{min}}, t_{C_a}^{\text{lb}}\} < t_{C_a}^{\text{high}}$.

Proof. For $P_{i,j}$ to be the first packet transmitted in some concrete state present in abstract state $\mathcal{S}_m$, it must respect the following properties:

- 1) Its class must be able to be the first class that transmit a packet. By Theorem 2, we must then have $t_{C_a}^{\text{lb}} \leq t^{\text{eet}} \wedge t_{C_a}^{\text{lb}} < t_{C_a}^{\text{high}}$.
- 2) $P_{i,j}$ must be at the head of its FIFO queue in some concrete state present in abstract state $\mathcal{S}_m$, that is, we must have $P_{i,j} \in \text{top}_{C_a}$.

- 3) It must have entered in the FIFO queue before any other class than C_a starts transmitting. Since $a_{P_{i,j}}^{\min}$ is a lower bound on when $P_{i,j}$ arrived at the switch and thus enters its FIFO queue, and because by Theorem 2, t^{eet} and $t_{C_a}^{\text{high}}$ are upper bounds on when C_a must start transmitting to be the first to transmit, it requires that $a_{P_{i,j}}^{\min} \leq t^{\text{eet}} \wedge a_{P_{i,j}}^{\min} < t_{C_a}^{\text{high}}$.

Applying all four properties yields the claim. $\square$

A packet that passes the requirements of Theorem 3 is called an eligible packet in abstract state $\mathcal{S}_m$. For every eligible packet $P_{i,j}$ of class C_a , we can compute a lower bound $s_{P_{i,j}}^{\min}$ and upper bound $s_{P_{i,j}}^{\max}$ on its start time in $\mathcal{S}_m$ as follows.

$$s_{P_{i,j}}^{\min} = \max\{a_{P_{i,j}}^{\min}, t_{C_a}^{\text{lb}}\} \quad (14)$$

$$s_{P_{i,j}}^{\max} = \min\{t^{\text{eet}}, t_{C_a}^{\text{high}}\} \quad (15)$$

The upper bound $s_{P_{i,j}}^{\max}$ is a direct consequence of Theorem 2, and the lower bound $s_{P_{i,j}}^{\min}$ comes from the fact that, to start its transmission, a packet must have entered its FIFO queue, which happens at the earliest at $a_{P_{i,j}}^{\min}$, and its class must be eligible, which happens at the earliest at $t_{C_a}^{\text{lb}}$ according to Lemma 3.

Since packet transmissions are non-preemptive, the earliest and latest finish time bounds on $P_{i,j}$'s transmission can be computed by adding the minimum and maximum transmission duration to the start time bounds.

$$f_{P_{i,j}}^{\min} = s_{P_{i,j}}^{\min} + C_{P_{i,j}}^{\min} \quad (16)$$

$$f_{P_{i,j}}^{\max} = s_{P_{i,j}}^{\max} + C_{P_{i,j}}^{\max} \quad (17)$$

After a packet has been transmitted, the abstract system state must be updated to reflect this change. The procedure for constructing the updated state is in the following section.

V. GENERATION OF NEW STATES

Once all eligible packets are identified, we determine how transmitting a packet $P_{i,j} \in C_a$ transforms an abstract state $\mathcal{S}_m$ into a new abstract state $\bar{\mathcal{S}}$.

The new abstract state $\bar{\mathcal{S}}$ is built as follows: (1) The set of transmitted packets is updated to include packet $P_{i,j}$, i.e., $\mathcal{T}(\bar{\mathcal{S}}) = \mathcal{T}(\mathcal{S}_m) \cup \{P_{i,j}\}$. (2) The availability of the egress port is updated to reflect the finish time of $P_{i,j}$. Since the port becomes available immediately after the transmission completes, we have $A^{\min}(\bar{\mathcal{S}}) = f_{P_{i,j}}^{\min}$ and $A^{\max}(\bar{\mathcal{S}}) = f_{P_{i,j}}^{\max}$. (3) The sets $\epsilon_{C_o}, \forall C_o \in \mathcal{C} \setminus C_a$ remain unchanged since the last packet transmitted by every class $C_o \in \mathcal{C} \setminus C_a$ is still the same as in $\mathcal{S}_m$. (4) The set ϵ_{C_a} must be updated in ways that are discussed in the rest of this section.

For each time-credit pair $(t, c) \in \epsilon_{C_a}$, we must determine the corresponding time-credit pairs at the possible finish times of $P_{i,j}$. To do so in two steps, we first compute the credit at the possible transmission start times $s \in [s_{P_{i,j}}^{\min}, s_{P_{i,j}}^{\max}]$ of the packet, and then calculate the credit when the packet completes its transmission.

The credit at a given start time depends on when the packet arrived, since credit evolves differently depending on whether

the queue is empty or non-empty. The TSN standard states that the credit evolves according to the following CBS rules:

- **Rule 1 (Reset when queue empty).** If the credit is non-negative when the queue becomes empty, it is reset to zero. The credit start increasing again at rate α_a^+ from the next packet arrival.
- **Rule 2 (Accumulation).** If the credit is negative or the credit is positive but the queue is non-empty, the credit increases continuously at rate α_a^+ until the packet at the top of the queue starts to be transmitted.
- **Rule 3 (Decrease during transmission).** While a packet is transmitted, credit decreases at rate of the send slope.

Rules 1 and 2 define how to calculate the value of the credit at the start of a packet transmission, and Rule 3 defines how the credit evolves from the start to the finish time of the transmission. In the following, we describe the first step of this two-step procedure.

Lemma 7. The credit accumulation function

$$g_{C_a}(t, c, a_{P_{i,j}}, s) = \begin{cases} (s, \alpha_a^+(s - a_{P_{i,j}})), & \text{if } (c \geq 0 \wedge t < a_{P_{i,j}}) \vee \\ & (c < 0 \wedge t - c/\alpha_a^+ < a_{P_{i,j}}) \\ (s, \alpha_a^+(s - t) + c), & \text{otherwise.} \end{cases} \quad (18)$$

gives the credit of class C_a at time s , starting from credit c at time t , when a packet arrives at time $a_{P_{i,j}}$. The two cases correspond directly to the behaviours defined by **Rule 1** and **Rule 2**.

Proof. Consider a coordinate (t, c) in ϵ_{C_a} , the arrival interval of the packet $a_{P_{i,j}} \in [a_{P_{i,j}}^{\min}, t_{C_a}^{\max}]$ and $s \in [s_{P_{i,j}}^{\min}, s_{P_{i,j}}^{\max}]$.

Case 1: Credit becomes non-negative before $a_{P_{i,j}}$ and no packet is waiting. If $c \geq 0$ and $t < a_{P_{i,j}}$, then the queue is empty before $a_{P_{i,j}}$, so, according to Rule 1, the credit was reset and maintained at zero until the packet arrives at $a_{P_{i,j}}$. Likewise, if $c < 0$ but the recovered credit would become zero at $t_0 = t - c/\alpha_a^+ < a_{P_{i,j}}$, the queue is still empty at t_0 , so the shaper holds the credit at zero until $a_{P_{i,j}}$. In both situations, the credit is zero at $a_{P_{i,j}}$ and starts to increase with rate α_a^+ from $a_{P_{i,j}}$ to s , thus yielding a credit of $\alpha_a^+(s - a_{P_{i,j}})$ at time s . This proves the first case of this lemma.

Case 2: A packet is waiting when the credit becomes non-negative. If neither of the above conditions holds, then the time when the credit becomes non-negative occurs at or after $a_{P_{i,j}}$, meaning a packet is already queued. Hence, CBS does not reset the credit, which increases linearly with slope α_a^+ from its initial value c at time t to reach $\alpha_a^+(s - t) + c$ at time s . This proves the second case of this lemma. $\square$

Example V.1. To illustrate the behaviour of the transformation function g , we continue the scenario from Figure 2. Assume the system is in state $\bar{\mathcal{S}}$ and the next eligible packet is $P_{2,2}$. Since $P_{2,2}$ arrives in $[7, 8]$ and the egress port is available in $[8, 10]$, Eqs. (10) and (13) give $t_{C_a}^{\text{lb}} = 8$ and $t_{C_a}^{\text{ub}} = 10$.

The credit accumulation function $g_{C_a}(t, c, a_{P_{i,j}}, s)$ is applied to all the coordinates in the set $\epsilon_{C_b} = \{(5, 0.5), (5, 0), (5, -0.5), (6, 0), (6, -0.5), (6, -1)\}$,

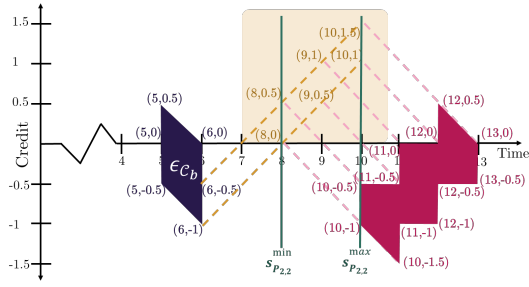


Fig. 4: Example showing how g_{C_2} projects the coordinates in ϵ_{C_b} from Example III.1 to the possible start times of $P_{2,2}$ (blue polygon projected to the yellow region), and how h_{C_2} projects the output of g_{C_2} to the possible finish times of $P_{2,2}$ (Yellow region projected to the pink polygon)

with $a_{P_{i,j}} \in [7, 8]$ and $s \in [8, 10]$. The resulting transformed coordinates are shown in Figure 4. All three behaviours of the CBS shaper appear:

Credit drops or remains at zero: For $(5, 0.5)$, $(5, 0)$, $(6, 0)$, the credit either falls to zero or stays at zero because, at times 5 and 6, the C_b queue is empty. This corresponds to the case $t < a_{P_{i,j}} \wedge c \geq 0$.

Credit reaches zero and stays there until arrival: For $(5, -0.5)$, the credit becomes zero at time 6 and remains zero because the queue is still empty. This matches the condition $t - \frac{c}{\alpha_b^-} < a_{P_{i,j}} \wedge c \leq 0$, ensuring no credit increase before the packet arrives.

Credit becomes non-negative, then accumulates: For $(6, -0.5)$ and $(6, -1)$, the credit reaches zero at times 7 and 8, respectively. A packet may already be queued at those times, so credit may begin to accumulate. In the case of $(6, -0.5)$, the credit becomes non-negative at time 7, but the packet may or may not have arrived yet (arrival is only guaranteed by time 8). Thus, two possibilities must be considered: one where the packet is present at time 7 and one where it only arrives at time 8. In both cases, once the packet is certainly present, the credit increases as expected. $\square$

Now that we have determined the credit values at all possible start times in $[s_{P_{i,j}}^{\min}, s_{P_{i,j}}^{\max}]$, we proceed to the second step: computing the credit at the finish times based on the transmission duration $C_{P_{i,j}} \in [C_{P_{i,j}}^{\min}, C_{P_{i,j}}^{\max}]$ of packet $P_{i,j}$. According to CBS, once a transmission begins, the credit reduces according to the send slope α_a^- . If the state at the start of transmission is (t, c) and the packet transmits for $C_{P_{i,j}}$ time units, then the time advances to $t + C_{P_{i,j}}$ while the credit decreases by $\alpha_a^- \cdot C_{P_{i,j}}$. The function

$$h_{C_a}(t, c, C_{P_{i,j}}) = (t + C_{P_{i,j}}, \alpha_a^- \cdot C_{P_{i,j}} + c) \quad (19)$$

captures this update, yielding the new credit-time coordinate at the end of the transmission interval.

Example V.2. Following the application of the transformation function g_{C_a} , we now demonstrate how the resulting coordinates are further processed using the function h_{C_a} for the same example as in Example V.1. For each output coordinate (t, c)

produced by g_{C_a} (marked separately in the Figure), we evaluate all possible transmission durations $C_{P_{i,j}} \in [C_{P_{i,j}}^{\min}, C_{P_{i,j}}^{\max}]$.

The function h_{C_a} computes the finish-time coordinate of a packet, based on the current time t , credit c , shaping parameter α_a^- , and transmission duration $C_{P_{i,j}}$. This results in a set of new coordinates, each representing a possible finish time and credit value after transmitting a packet with duration $C_{P_{i,j}}$.

The complete set of such coordinates, obtained by applying h_{C_a} to every $g(t, c, a_{P_{i,j}}, s)$ and all values of $C_{P_{i,j}} \in [C_{P_{i,j}}^{\min}, C_{P_{i,j}}^{\max}]$, forms the set of finish-time coordinates for the corresponding class. This transformed set is given by the space marked in pink in Figure 4. $\square$

Using the functions g_{C_a} and h_{C_a} , the updated credit-time space for class C_a in the new state $\bar{S}$ is computed as

$$\epsilon_{C_a}(\bar{S}) = \left\{ \begin{array}{l} (t, c) \in \epsilon_{C_a}(S_m), \\ a_{P_{i,j}} \in [a_{P_i}^{\min}, tr_{C_a}^{\max}], \\ s \in [s_{P_{i,j}}^{\min}, s_{P_{i,j}}^{\max}], \\ C_{P_{i,j}} \in [C_{P_i}^{\min}, C_{P_i}^{\max}] \end{array} \right\} \quad (20)$$

Theorem 4. The transformation from the credit-time space $\epsilon_{C_a}(S_m)$ in state S_m to the updated space $\epsilon_{C_a}(\bar{S})$ in state $\bar{S}$ captures all possible transmission scenarios that occur after the transmission of a packet.

Proof. The transformation has two stages.

First, $g_{C_a}(t, c, a_{P_{i,j}}, s)$ maps each point $(t, c) \in \epsilon_{C_a}(S_m)$ to the corresponding credit at the start time s of $P_{i,j}$, assuming $P_{i,j}$ entered the FIFO queue at time $a_{P_{i,j}}$. As shown in Lemma 7, this function exactly follows CBS behaviour in all cases, including whether the queue is empty when the credit becomes non-negative at or after t .

Second, $h_{C_a}(t, c, C_{P_{i,j}})$ updates this intermediate point by simulating the transmission of $P_{i,j}$ for duration $C_{P_{i,j}}$. During this transmission, the credit decreases at rate α_a^- , as required by CBS.

The transformation ranges over all admissible values over all admissible values of $a_{P_{i,j}}$, s , and $C_{P_{i,j}}$, namely, $a_{P_{i,j}} \in [a_{P_i}^{\min}, tr_{C_a}^{\max}]$, $s \in [s_{P_{i,j}}^{\min}, s_{P_{i,j}}^{\max}]$, $C_{P_{i,j}} \in [C_{P_i}^{\min}, C_{P_i}^{\max}]$.

Thus, all possible packet arrivals, start times, and transmission durations are considered. Therefore, $\epsilon_{C_a}(\bar{S})$ contains all valid time-credit pairs for class C_a in the successor state $\bar{S}$, and the transformation safely captures all scenarios resulting from transmitting a packet from S_m . $\square$

VI. MERGING OF STATES

Exploring every possible scheduling decision may cause state-space explosion. To mitigate this, we merge states with similar contents, since they lead to similar future scheduling decisions.

Two states, S_1 and S_2 , are considered similar if:

- 1) The set of transmitted packets is identical, i.e., $\mathcal{T}(S_1) = \mathcal{T}(S_2)$.
- 2) The availability intervals of the egress port are either overlapping or contiguous, i.e.,

$$[A^{\min}(\mathcal{S}_1), A^{\max}(\mathcal{S}_1)] \cup [A^{\min}(\mathcal{S}_2), A^{\max}(\mathcal{S}_2)] = [\min\{A^{\min}(\mathcal{S}_1), A^{\min}(\mathcal{S}_2)\}, \max\{A^{\max}(\mathcal{S}_1), A^{\max}(\mathcal{S}_2)\}]$$

If these conditions hold, the two states are merged into a state $\mathcal{S}'$ defined as follows:

- 1) The set of finished jobs is kept identical, $\mathcal{T}(\mathcal{S}') = \mathcal{T}(\mathcal{S}_1) = \mathcal{T}(\mathcal{S}_2)$.
- 2) The egress port availability interval is the union of the intervals, $A^{\min}(\mathcal{S}') = \min\{A^{\min}(\mathcal{S}_1), A^{\min}(\mathcal{S}_2)\}$, and $A^{\max}(\mathcal{S}') = \max\{A^{\max}(\mathcal{S}_1), A^{\max}(\mathcal{S}_2)\}$.
- 3) The credit-time space of the merged node $\mathcal{S}'$ is defined on a per-class basis. For each traffic class $\mathcal{C}_a$, we take the union of the corresponding credit-time spaces from $\mathcal{S}_1$ and $\mathcal{S}_2$, $\epsilon_{\mathcal{C}_a}(\mathcal{S}') = \epsilon_{\mathcal{C}_a}(\mathcal{S}_1) \cup \epsilon_{\mathcal{C}_a}(\mathcal{S}_2)$.

The merged state $\mathcal{S}'$ replaces $\mathcal{S}_1$ and $\mathcal{S}_2$, and all incoming edges to either state are redirected to $\mathcal{S}'$.

VII. PROOF OF CORRECTNESS

The previous sections showed that each individual scheduling decision in the SAG is modelled safely. We now extend this argument to the full graph. We prove that the SAG explores all possible scheduling decisions and their combinations, so every valid transmission scenario corresponds to some path generated by Alg. 1. We then show that the resulting graph provides safe lower and upper latency bounds for every packet.

Theorem 5. The SAG explores all reachable concrete system states across all transmission scenarios in Γ .

Proof. We prove by induction over the SAG layers that every concrete system state reachable under any transmission scenario in Γ is represented by some abstract state.

Let L^l be the l -th SAG layer. For each abstract state $\mathcal{S}_m \in L^l$, let $\Gamma(\mathcal{S}_m)$ denote a superset of concrete transmission scenarios $\gamma \in \Gamma$ whose concrete system state after exactly l packet transmissions is represented by $\mathcal{S}_m$. We show that

$$\bigcup_{\mathcal{S}_m \in L^l} \Gamma(\mathcal{S}_m) \supseteq \Gamma \quad \text{for all } l = 0, 1, 2, \dots$$

Base case ($l = 0$): Initially, no packet has been transmitted and the egress port is free. This unique initial concrete system state for all transmission scenarios is represented by $\mathcal{S}_0 \in L^0$. Hence,

$$\Gamma(\mathcal{S}_0) \supseteq \Gamma \Rightarrow \bigcup_{\mathcal{S}_m \in L^0} \Gamma(\mathcal{S}_m) \supseteq \Gamma.$$

Induction Step ($l \mapsto l+1$): Assume that at layer l ,

$$\bigcup_{\mathcal{S}_m \in L^l} \Gamma(\mathcal{S}_m) \supseteq \Gamma$$

Thus, for every transmission scenario $\gamma \in \Gamma$, the concrete state after l packet transmissions is represented by some $\mathcal{S}_m \in L^l$.

Consider any such abstract state $\mathcal{S}_m \in L^l$ and any transmission scenario $\gamma \in \Gamma(\mathcal{S}_m)$. Let $P_{i,j}$ be the next packet transmitted in γ . By Theorem 3, $P_{i,j}$ satisfies the eligibility conditions implied by the abstract information in $\mathcal{S}_m$. Therefore, Alg. 1 includes $P_{i,j}$ among the eligible packets considered for $\mathcal{S}_m$.

For this eligible packet, the algorithm creates a successor abstract state $\bar{\mathcal{S}} \in L^{l+1}$. By construction, $\bar{\mathcal{S}}$ represents the concrete system state reached after transmitting $P_{i,j}$ in transmission scenario γ . If successor states are merged, the merge rules preserve this representation.

Therefore, for every $\gamma \in \Gamma(\mathcal{S}_m)$, there exists a successor $\bar{\mathcal{S}} \in L^{l+1}$ such that $\gamma \in \Gamma(\bar{\mathcal{S}})$. Hence,

$$\bigcup_{\bar{\mathcal{S}} \in L^{l+1}} \Gamma(\bar{\mathcal{S}}) \supseteq \bigcup_{\mathcal{S}_m \in L^l} \Gamma(\mathcal{S}_m) \supseteq \Gamma.$$

By induction, each layer L^l of the SAG represents all concrete system states reachable after l packet transmissions in all transmission scenarios in Γ . Therefore, across all its layers, the SAG explores all reachable concrete system states in all transmission scenarios in Γ . $\square$

Theorem 6. Algorithm 1 returns safe bounds on the *BCL* and *WCL* of every packet.

Proof. By Theorem 5, every packet transmission in every scenario in Γ is represented by some abstract state in the SAG. For any abstract state $\mathcal{S}_m$ and packet $P_{i,j}$ transmitted from that state, Algorithm 1 computes $f_{P_{i,j}}^{\min}$ and $f_{P_{i,j}}^{\max}$, which bound the finish time of $P_{i,j}$ for all concrete states represented by $\mathcal{S}_m$. The algorithm then updates the global best- and worst-case latencies of $P_{i,j}$ by taking the minimum and maximum relative finish times over all explored abstract states.

Since all concrete transmissions are represented and each computed finish-time interval safely bounds the corresponding concrete finish times, taking the minimum and maximum over all abstract states yields safe lower and upper bounds on the BCL and WCL of each packet across all scenarios in Γ . $\square$

VIII. IMPROVING THE ANALYSIS SCALABILITY

Implementing Alg. 1 introduces challenges not present in previous SAG-based analyses. Earlier work tracked scalar time-interval bounds, whereas our method must store and update two-dimensional time-credit regions $\epsilon_{\mathcal{C}_a}$ for every class. This increases memory usage and run time, since each state may contain many, possibly unbounded, time-credit pairs. In addition, unlike prior systems, no finite observation window is known to guarantee completeness for CBS analysis. Without such a bound, the analysis may need to explore infinitely many packet arrivals. We address these issues through two improvements: (1) a compact, safe over-approximation of the regions $\epsilon_{\mathcal{C}_a}$, and (2) a safe termination mechanism that avoids requiring an explicit observation window.

A. Optimising State Representation

When a new state is created, each class stores a credit-time region $\epsilon_{\mathcal{C}_a}$. Storing this region exactly is impractical, so we use a controlled over-approximation based on the structure of SAG updates. Since $\epsilon_{\mathcal{C}_a}$ contains all possible credit-time pairs for the most recently transmitted packet of class $\mathcal{C}_a$, the interval $[f_{P_{i,j}}^{\min}, f_{P_{i,j}}^{\max}]$ safely bounds all possible finish times of the region. By construction, no finish time generated by the

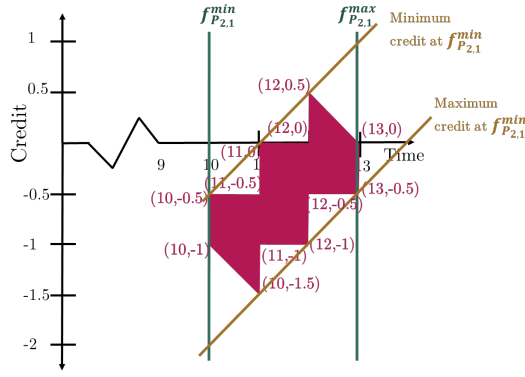


Fig. 5: Overapproximating the set of credit-time coordinates into a compact parallelogram representation

analysis can lie outside this interval, so it fully captures the admissible domain of ϵ_{C_a} .

To bound the credit values, we use the fact that credit-time trajectories evolve only along the idle slope or the send slope. Projecting every point of ϵ_{C_a} along the idle slope to $x = f_{P_{i,j}}^{\min}$ and $x = f_{P_{i,j}}^{\max}$ yields conservative lower and upper credit limits. These projected intervals define a parallelogram that safely encloses the full region ϵ_{C_a} and preserves all information needed for future transitions, as shown in Fig. 5. Thus, each region can be stored compactly using the four values that define this parallelogram.

When two states are merged, their credit-time regions are combined by spanning both finish-time intervals and both projected credit ranges. The resulting parallelogram encloses both original regions, so the merge remains a sound overapproximation.

This representation may include transmission scenarios that cannot occur. However, since all real scenarios are still covered and the approximation only adds extra paths, the analysis remains sound.

Example VIII.1. In Example V.1, we derived the final set of coordinates that must be retained after transmitting a packet. Figure 5 shows how this set is approximated. The earliest and latest finishing times, marked by the vertical green lines, define $[f_{P_{i,j}}^{\min}, f_{P_{i,j}}^{\max}] = [10, 13]$. Projecting all coordinates back to $f_{P_{i,j}}^{\min}$ along the idle slope gives the credit interval $[-2, -0.5]$. These two intervals, together with the idle slope, define the parallelogram that replaces the full coordinate set. $\square$

B. Safe Termination in the Absence of an Observation Window

Previous SAG analyses rely on safe observation windows, i.e., finite intervals such that if no deadline is missed within the interval, no deadline can be missed later. To the best of our knowledge, no such bound is known for CBS. We therefore use convergence across hyperperiods to decide when the analysis can safely stop.

At the end of each hyperperiod H , we record the set of abstract states reached by the SAG and compare it with the sets recorded at the end of previous hyperperiods.

Let the k -th hyperperiod boundary be $k \times H$, with $k \in \mathbb{N}$. Let $\mathcal{S}_A$ and $\mathcal{S}_B$ be abstract states reached at the k_1 -th and k_2 -th hyperperiod boundaries, respectively. We say that $\mathcal{S}_A$ is subsumed by $\mathcal{S}_B$ iff (i) for every traffic class C_a and every $(t, c) \in \epsilon_{C_a}(\mathcal{S}_A)$, there exists $(t', c') \in \epsilon_{C_a}(\mathcal{S}_B)$ such that $c = c'$ and $t - k_1H = t' - k_2H$, (ii) for every flow $F_i \in \mathcal{F}$, $\min\{a_{P_{i,j}}^{\min} \mid P_{i,j} \notin \mathcal{T}(\mathcal{S}_A)\} - k_1H = \min\{a_{P_{i,p}}^{\min} \mid P_{i,p} \notin \mathcal{T}(\mathcal{S}_B)\} - k_2H$ and (iii) $[A^{\min}(\mathcal{S}_A), A^{\max}(\mathcal{S}_A)] \subseteq [A^{\min}(\mathcal{S}_B), A^{\max}(\mathcal{S}_B)]$ ².

If every abstract state reached at the current hyperperiod boundary is subsumed by a previously recorded state, then all future scheduling decisions from the current states have already been explored, and the SAG can safely terminate. Otherwise, the newly discovered states are recorded and exploration continues. This provides safe termination without requiring a known observation window.

IX. EXPERIMENTAL RESULTS



Fig. 6: Network architecture with a single switch

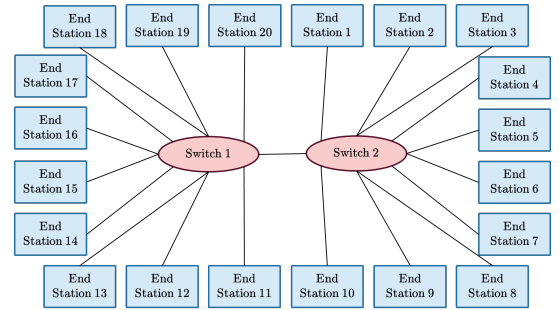


Fig. 7: Network architecture with two switches

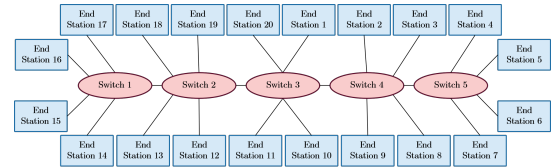
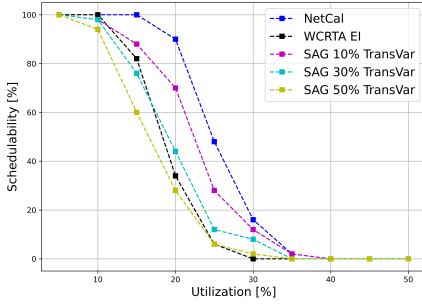


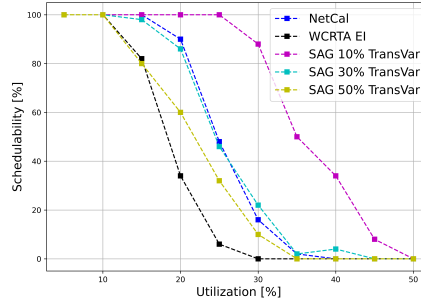
Fig. 8: Network architecture with five switches

We now evaluate the accuracy and scalability of the proposed SAG analysis for CBS-shaped TSN networks. We compare it with two existing approaches: worst-case response-time analysis based on eligible intervals (WCRTA-EI) [14] and a network-calculus-based method (NetCal) [4]. We use three network configurations and, for these comparisons, restrict the setup to two CBS-shaped classes because the implementations provided by the authors of [14] and published in [28] support at most two such classes, although the theoretical analyses

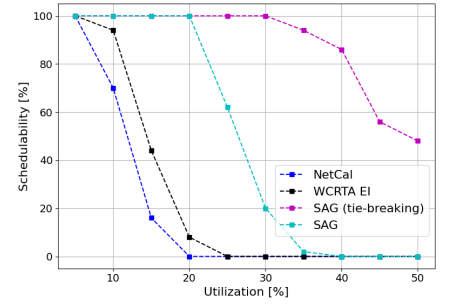
²Note that we do not need to explicitly compare the concrete content of the queues in states $\mathcal{S}_A$ and $\mathcal{S}_B$ because their content at the hyperperiod boundary is directly implied by the arrival times of the packets that are still pending at the beginning of the next hyperperiod. Thus, if all flows have the same relative arrival time intervals for their next undispached packet, the content of the queues will be identical too.



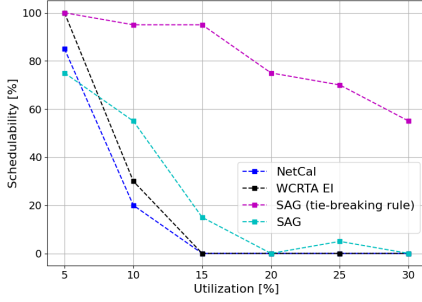
(a) Schedulability ratio for the network in Fig. 6



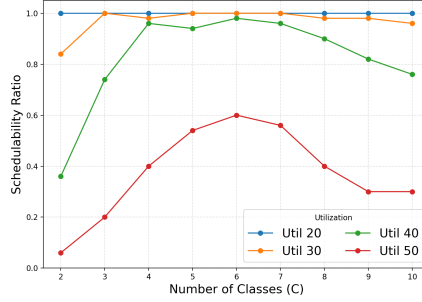
(b) Schedulability ratio for the network in Fig. 6 with tie-breaking mechanism



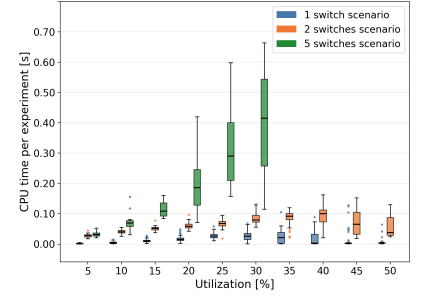
(c) Schedulability ratio for the network in Fig. 7



(d) Schedulability ratio for the network in Fig. 8



(e) Impact of the number of traffic classes on schedulability



(f) Runtime comparison of the three network configurations shown in Fig. 6, 7, and 8 with the tie-breaking rule enabled

Fig. 9: Experimental Results

in [14], [4] support more. We also use our analysis to study the effect of varying the number of traffic classes.

We use the network and traffic generator from [28] to create the topologies and traffic configurations. The three network setups are shown in Fig. 6, Fig. 7 and Fig. 8. All traffic is shaped by CBS. The generator produces up to 100 periodic flows over full-duplex 100 Mbps links. Message sizes range from 500 to 1500 bytes, periods are at most 10 000 μ s, and each flow deadline equals its period. For the configurations in Fig. 6 and Fig. 7, we generated 50 unique flow sets each and for the configuration in Fig. 8, we generated 20 unique flow sets.

We assume known arrival patterns and phasing in all experiments. Such traffic is traditionally shaped using Time-Aware Shapers (TAS) with fixed transmission windows. However, these windows must be conservatively sized and may waste bandwidth. CBS avoids this inefficiency while still providing deterministic shaping, making it suitable for our setting.

We evaluate two SAG variants. The first uses Eq. (5) directly and assumes that no information about the tie-breaking rule is known when identifying packets that may be at the head of a FIFO queue. Therefore, for packets with equal arrival times, this variant conservatively considers all possible FIFO orderings. The second variant modifies Eq. (5) such that, if two or more packets of C_a have $a_{P_{i,j}}^{\min} = tr_{C_a}^{\max}$, only the packet with the smallest flow ID is added to top_{C_a} . This corresponds to a deterministic tie-breaking rule where ties between simultaneous arrivals are resolved in favour of the

packet whose flow has the smallest identifier.

Each experiment ran on a Linux platform, took from a few tens to several hundreds of milliseconds, and used about a dozen megabytes of memory. We do not compare runtime and memory with [14] and [4], since their implementations were only available as Windows executables and ran on a less powerful laptop, making such a comparison unfair to them.

Evaluation of a single switch full-duplex network.

We first evaluate the network in Fig. 6. WCRTA-EI and NetCal assume fixed packet sizes for packets of the same flow, whereas our analysis supports uncertainty in transmission duration. We therefore test three levels of transmission-duration variation: 10%, 30%, and 50%. Traffic is divided into two CBS-shaped classes, each allocated 50% of the bandwidth. Each flow has release jitter with probability 50%, and jitter values are randomly selected from 1–100 μ s.

As shown in Fig. 9a, the generic SAG analysis achieves schedulability ratios comparable to WCRTA-EI, sometimes higher and sometimes lower depending on the transmission-duration variation. Fig. 9b shows that exploiting tie-breaking information improves SAG schedulability. This demonstrates that SAG can use implementation-specific information to reduce pessimism.

The comparison with NetCal also highlights differences between packet-level and flow-level analysis. SAG analyzes individual packets, whereas NetCal operates at the flow level, potentially introducing different sources of pessimism. Understanding why NetCal outperforms SAG in some scenarios and

SAG outperforms NetCal in others remains an open problem. Nevertheless, these differences suggest that the two approaches offer complementary strengths in schedulability analysis.

Evaluation of a Multi-Switch Network.

We next evaluate the larger acyclic topologies in Fig. 7 and Fig. 8, again comparing SAG with WCRTA-EI and NetCal. In these experiments, packets are assumed to have no arrival jitter at the first switch they enter. The analysis then bounds each packet's departure time from that switch, and the resulting departure interval determines its arrival jitter at the next switch. This process is repeated along each packet's path, accounting for the jitter accumulated across successive switches. Since the topology is acyclic, the egress ports can be analysed in topological order, ensuring that the arrival intervals of all packets entering a port are known before that port is analysed. The packet deadlines are defined at the network-wide level rather than being specific to an individual egress port. This experiment assumes 10% transmission-duration variation.

As shown in Fig. 9c and Fig. 9d, SAG achieves a higher schedulability ratio than both baselines in nearly all cases, both with and without tie-breaking information. Incorporating tie-breaking further reduces pessimism, although such rules may not always be known.

Multi-switch support is obtained by executing SAG iteratively along each flow route. At each switch, SAG computes best- and worst-case completion times and forwards them as arrival-time jitter bounds to the next switch. This continues until each packet has been analysed along its full path. For acyclic networks such as Fig. 7 and Fig. 8, this gives a straightforward way to analyse multi-hop topologies.

WCRTA-EI [14] and NetCal [4] are offset-agnostic and assume unknown phasing between flows. This allows them to support broader traffic models, including sporadic traffic, but prevents them from exploiting known phasing and arrival information. In contrast, SAG uses this information to obtain tighter results. Prior work on switched Ethernet and AFDX networks [15], [16], [17], [18], [19] has shown that incorporating end-system scheduling and flow offsets can reduce pessimism in worst-case delay analysis, but these ideas have not yet been applied to CBS.

Note that this paper only assumed acyclic network topologies. For cyclic network topologies, there is no topological order in which to analyse the switches egress ports. One possible approach is to iteratively repeat the analysis, updating the arrival-jitter bounds of packets entering in each switch at each iteration, until the response-time bounds no longer change. Proving that this process converges and produces safe bounds is outside the scope of this work.

Exploring the Impact of the Number of Traffic Classes.

Finally, we study how the number of traffic classes affects schedulability. Starting with two CBS-shaped classes, we increase the number of classes up to ten by randomly reassigning flows while keeping all other parameters fixed. Each class receives an idle slope proportional to the total traffic load assigned to it.

As shown in Fig. 9e, schedulability initially improves as the number of classes increases because fewer flows share each FIFO queue, reducing interference. However, with too many classes, each class receives a smaller bandwidth share, reducing schedulability. In this configuration, the best result occurs with six classes, balancing reduced contention with sufficient bandwidth per class.

Impact of Network Size and Utilization on Analysis Runtime.

The plot compares total SAG analysis CPU time per experiment for the one- and two-switch networks up to 50% utilization and the five-switch network up to 30% utilization. Runtime generally increases with the number of switches, but all plotted measurements remain below one second. The five-switch network also shows larger differences in runtime between experiments. At low utilization, less contention means fewer scheduling possibilities need to be explored. For the one- and two-switch networks, runtime is highest at intermediate utilization and falls at higher utilization. This decrease is consistent with more experiments stopping early when a link fails the schedulability check, leaving the remaining links unanalysed.

X. CONCLUSION

This work extends the Schedule Abstraction Graph (SAG) methodology to analyze communication flows with a known arrival pattern shaped by Credit-Based Shaping (CBS). We introduced a scalable and flexible schedulability analysis for CBS-based TSN networks and showed through comparison with existing methods that it maintains accuracy, outperforming state-of-the-art analyses, especially for multi-switches networks. These experiments show that the analysis proposed in this paper properly balances scalability with accuracy. Future work will integrate this analysis with existing approaches for Time-Aware Shaping [13] to provide a unified framework covering the major TSN shapers. We also plan to extend the method to incorporate preemptive scheduling effects reusing results of [26].

ACKNOWLEDGMENT

This result is part of a project (PropRT) that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 865170). This work has been funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project No. 569077889 (PEACH).

REFERENCES

- [1] "IEEE standard for local and metropolitan area networks—bridges and bridged networks," *IEEE Std 802.1Q-2022 (Revision of IEEE Std 802.1Q-2018)*, pp. 1–2163, 2022.
- [2] R. Queck, "Analysis of ethernet AVB for automotive networks using network calculus," in *2012 IEEE International Conference on Vehicular Electronics and Safety (ICVES 2012)*, pp. 61–67, IEEE, 2012.
- [3] J. A. R. De Azua and M. Boyer, "Complete modelling of AVB in network calculus framework," in *Proc. 22nd International Conference on Real-Time Networks and Systems*, pp. 55–64, 2014.
- [4] L. Zhao, P. Pop, Z. Zheng, H. Daigmore, and M. Boyer, "Latency analysis of multiple classes of AVB traffic in TSN with standard credit behavior using network calculus," *IEEE Transactions on Industrial Electronics*, vol. 68, no. 10, pp. 10291–10302, 2020.

- [5] J. Diemer, J. Rox, and R. Ernst, "Modeling of ethernet AVB networks for worst-case timing analysis," *IFAC Proceedings Volumes*, vol. 45, no. 2, pp. 848–853, 2012.
- [6] J. Diemer, D. Thiele, and R. 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)*, pp. 1–10, IEEE, 2012.
- [7] J. Cao, P. J. Cuijpers, R. J. Bril, and J. 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*, pp. 55–64, 2016.
- [8] X. Li and L. George, "Deterministic delay analysis of AVB switched ethernet networks using an extended trajectory approach," *Real-Time Systems*, vol. 53, no. 1, pp. 121–186, 2017.
- [9] L. L. Bello, M. Ashjaei, G. Patti, and M. Behnam, "Schedulability analysis of time-sensitive networks with scheduled traffic and preemption support," *Journal of Parallel and Distributed Computing*, vol. 144, pp. 153–171, 2020.
- [10] U. D. Bordoloi, A. Aminifar, P. Eles, and Z. Peng, "Schedulability analysis of ethernet AVB switches," in *2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications*, pp. 1–10, IEEE, 2014.
- [11] D. Thiele, R. Ernst, and J. Diemer, "Formal worst-case timing analysis of ethernet TSN's time-aware and peristaltic shapers," in *2015 IEEE Vehicular Networking Conference (VNC)*, pp. 251–258, IEEE, 2015.
- [12] L. Zhao, P. Pop, and S. S. Craciunas, "Worst-case latency analysis for IEEE 802.1 qbv time sensitive networks using network calculus," *IEEE Access*, vol. 6, pp. 41803–41815, 2018.
- [13] S. Srinivasan, G. Nelissen, R. J. Bril, and N. Meratnia, "Analysis of TSN Time-Aware Shapers Using Schedule Abstraction Graphs," in *36th Euromicro Conference on Real-Time Systems (ECRTS 2024)* (R. Pellizzoni, ed.), vol. 298 of *Leibniz International Proceedings in Informatics (LIPIcs)*, (Dagstuhl, Germany), pp. 16:1–16:24, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024.
- [14] 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 *2024 IEEE Real-Time Systems Symposium (RTSS)*, pp. 135–147, 2024.
- [15] M. Adnan, J.-L. Scharbarg, J. Ermont, and C. Fraboul, "Model for worst case delay analysis of an afdx network using timed automata," in *2010 IEEE 15th Conference on Emerging Technologies & Factory Automation (ETFA 2010)*, pp. 1–4, 2010.
- [16] X. Li, J.-L. Scharbarg, and C. Fraboul, "Partially synchronizing periodic flows with offsets improves worst-case end-to-end delay analysis of switched ethernet," in *Leveraging Applications of Formal Methods, Verification, and Validation* (T. Margaria and B. Steffen, eds.), (Berlin, Heidelberg), pp. 228–242, Springer Berlin Heidelberg, 2010.
- [17] X. Li, J.-L. Scharbarg, F. Ridouard, and C. Fraboul, "Existing offset assignments are near optimal for an industrial afdx network," *SIGBED Rev.*, vol. 8, p. 49–54, Dec. 2011.
- [18] A. Soni, X. Li, J.-L. Scharbarg, and C. Fraboul, "Integrating Offset in Worst Case Delay Analysis of Switched Ethernet Network With Deficit Round Robbin," in *Proceedings of ETFA 2018*, (Turin, Italy), pp. 353–359, Sept. 2018.
- [19] M. Boyer, L. Santinelli, N. Navet, J. Migge, and M. Fumey, "Integrating end-system frame scheduling for more accurate AFDX timing analysis," in *ERTS 2014 proceedings*, (Toulouse, France), Feb. 2014.
- [20] J. Lv, Y. Zhao, X. Wu, Y. Li, and Q. Wang, "Formal analysis of TSN scheduler for real-time communications," *IEEE Trans. Reliab.*, vol. 70, no. 3, pp. 1286–1294, 2021.
- [21] M. Nasri and B. B. Brandenburg, "An exact and sustainable analysis of non-preemptive scheduling," in *2017 IEEE Real-Time Systems Symposium (RTSS)*, pp. 12–23, 2017.
- [22] M. Nasri, G. Nelissen, and B. B. Brandenburg, "A Response-Time Analysis for Non-Preemptive Job Sets under Global Scheduling," in *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)* (S. Altmeyer, ed.), vol. 106 of *Leibniz International Proceedings in Informatics (LIPIcs)*, (Dagstuhl, Germany), pp. 9:1–9:23, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.
- [23] S. Srinivasan, M. Günzel, and G. Nelissen, "Response-time analysis for limited-preemptive self-suspending and event-driven delay-induced tasks," in *2024 IEEE Real-Time Systems Symposium (RTSS)*, pp. 29–42, 2024.
- [24] Y. Zhao, S. Srinivasan, G. Nelissen, and M. Nasri, "Work-in-progress: Generating counter-examples to schedulability using the schedule abstraction," in *2023 IEEE Real-Time Systems Symposium (RTSS)*, pp. 459–462, 2023.
- [25] S. Ranjha, G. Nelissen, and M. Nasri, "Partial-Order Reduction for Schedule-Abstraction-based Response-Time Analyses of Non-Preemptive Tasks," in *RTAS*, pp. 121–132, 2022.
- [26] P. Gohari, J. Voeten, and M. Nasri, "Reachability-Based Response-Time Analysis of Preemptive Tasks Under Global Scheduling," in *36th Euromicro Conference on Real-Time Systems (ECRTS 2024)* (R. Pellizzoni, ed.), vol. 298 of *Leibniz International Proceedings in Informatics (LIPIcs)*, (Dagstuhl, Germany), pp. 3:1–3:24, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024.
- [27] P. Gohari, M. Mikołajczyk, T. van de Klundert, J. Voeten, M. Nasri, and G. Nelissen, "Response-time Analysis of Conditional DAG Tasks under Limited-Preemptive Global Scheduling," in *33rd International Conference on Real-Time Networks and Systems (RTNS 2025)*, 2025.
- [28] D. Bujosa, M. Ashjaei, and S. Mubeen, "Conan-tsn: An integrated toolchain for configuration and analysis of tsn networks," in *2025 IEEE 30th International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–8, 2025.