Egress-TT Configurations for TSN Networks

Pierre-Julien Chaine, Marc Boyer -Onera, Claire Pagetti, Franck Wartel

Egress-TT Configurations for TSN Networks

Pierre-Julien CHAINE – Airbus – Toulouse, France
pierre-julien.chaine@airbus.com – ORCID: 0000-0002-5062-8694

Marc BOYER – ONERA / DTIS
Université de Toulouse, F-31055 Toulouse, France
Marc.Boyer@onera.fr – ORCID: 0000-0003-2910-4738

Claire PAGETTI – ONERA / DTIS
Université de Toulouse, F-31055 Toulouse, France
Claire.Pagetti@onera.fr – ORCID: 0000-0001-7265-1839

Franck WARTEL – Airbus Defence and Space – Toulouse, France
franck.wartel@airbus.com – ORCID: 0000-0002-5265-647X

Abstract

Latency and jitter are two main requirements when controlling cyber-physical systems, especially in case of remote control through a network. When jitter is a major concern, one can either control the jitter all along the path, or compensate it by buffering at reception. This is usually done by the destination host, but it can also be done by the last network node. We call this approach *Egress TT*. This paper presents *Egress TT*, its benefits in terms of configuration time and applicative constraints. It also presents two possible implementations within a Time Sensitive Networking context.

1 Introduction

The control of a cyber-physical system by a computer is often done by sending messages through a network. The adequate behaviour of the system depends not only on the latency in the network but also on its variability, called jitter.

One common way to provide low jitter in embedded networks consists in controlling it all along the path, by carefully choosing the emission instants (also known as time slots) of all messages on all hops, *i.e* computing a *schedule*. Such time-triggered behaviour is called subsequently *End-to-End TT*.

Another way, mostly done at application level, consists in compensating the latency variability by buffering messages and delivering them at the correct instant. This is the approach of LET - Logical Execution Time [13]-. Our proposal, that we call *Egress TT* (called LETT by [3]), consists in doing this buffering at the egress node of the network. The purpose of that is to reduce

the development effort at application level by relocating the on-time delivery capability into the network. In these approaches, latency is traded for jitter. This is however not always an issue: in some real-time systems, the requirement is not to have the smallest latency, but rather to respect a deadline constraint. Therefore a system designer will be able to choose between one approach or the other depending on its needs.

Moreover, *End-to-End TT* has some drawbacks. First determining its schedules is computationally expensive and scalability is a real issue. Second, it requires that all hops along the path are able to respect this schedule. It also has advantages as the determinism and the ability to guarantee small latencies.

TSN is composed of several standards, extending Ethernet with several real-time mechanisms, among which IEEE 802.1Qbv [16] that allows to implement the End-to-End TT approach [8, 27, 10, 14] and benefits from its advantages. But when considering TSN, a third drawback appears, related to the queue-based implementation of TSN. If two frames f_1 and f_2 are supposed to be emitted one after the other, they have to be put in the queue in the same order. Moreover, if f_1 is not produced, its slot can be used by f_2, breaking the pre-computed schedule, as identified by [8] and recalled in section 3.2. This means that the End-to-End TT approach applied to TSN has an impact on data production and task scheduling.

Egress TT tackles the drawbacks listed for End-to-End TT, especially in a TSN context. First, in *Egress TT*, not all devices need to implement TSN but only the very last device in the path of any flow. The others can rely on standard Ethernet (cf. [15]). This will facilitate the upgrade of networks by not replacing all devices with TSN-capable ones. Second, as we propose in this paper, *Egress TT* configuration can be implemented using only priority based arbitration along the path, except on the last hop port in the path of any flow. This approach drastically reduces the computation effort thanks to the reduction of the number of schedules having to be computed (the experiments will show that scalability is not anymore an issue). Lastly, *Egress TT* was also designed to minimize the constraints on the scheduling of the tasks in charge of data production.

We propose two possible implementations of *Egress TT* over TSN networks, *Exclusive Queue Allocation* and *Size Based Isolation*, in order to fulfil fault-tolerance requirements. Indeed, when a message of a flow is lost, it should not affect any other flow. This comes with a price: the number of flows in a last hop port is limited. We have proposed a constraint programming formalization to compute *Egress TT* configurations and made a thorough experiments campaign to assess the quality of the approach, in particular compared to End-to-End TT. Experiments confirm that latencies are traded off for jitter i.e. *Egress TT* configurations induce greater latencies than End-to-End TT. Nevertheless, they offer other advantages (see above) that are more significant in industrial systems.

The paper is organized as follows: Section 2 introduces the system model and Section 3 presents TSN 802.1Qbv. Section 4 describes the problem state-

ment. Section 5 introduces the concepts of *Egress TT* and section 6 details the constraint programming methodology to compute *Egress TT* configurations. Finally, we compare *Egress TT* to End-to-End TT via a large campaign of experiments in Section 7.

2 System Model

2.1 Host model

At emission, when an application produces a message, it is put into a mailbox (ISO L7) and then placed in the appropriate queue at MAC level (ISO L2), waiting for emission.

Definition 1 (Deposit / Emission instants)**.** *Let m be a message. We define the* deposit instant $T_{SAP}(m)$ *as the instant at which m is deposited in the L2 service access point. We define the* emission instant $T_e(m)$ *as the instant at which the first bit of m is emitted on the medium.*

Definition 2 (Reception / Delivery instants)**.** *Let m be the a message. We define the* reception instant $T_r(m)$ *as the instant at which the last bit of m is received at receiver end-station physical level. We define the* delivery instant $T_d(m)$ *as the instant at which m is provided to the receiver application.*

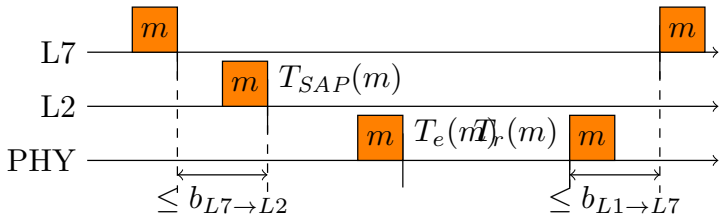


Figure 1: Frame behaviour on the network

Hypothesis 1 (Restricting the problem at the flow level)**.** *We assume we know two bounds $b_{L7\to L2}$, $b_{L1\to L7}$ on the time needed to cross the host layers: forall message m: $T_e(m) - T_{SAP}(m) \leq b_{L7\to L2}$, $T_d(m) - T_r(m) \leq b_{L1\to L7}$.*

This is illustrated in Fig. 1. As a consequence, the configuration problem can be defined and solved solely at the network level.

2.2 Flow-level model

Definition 3 (Flow)**.** *Let $\mathbb{F}$ be the set of flows. A flow $f \in \mathbb{F}$ is characterized by the tuple $\langle Src_f, LDests_f, Size_f, P_f \rangle$ where:*

- *Src_f is the source end-station which emits the messages;*

- $LDests_f$ *is the set of receiver end-stations;*
- $Size_f$ *is the constant size in bytes of one message.*
- P_f *is the period of the flow. Thus, a flow* f *is a sequence of messages (or frames). We denote by* f_l *the* l*-th message of* f*;*

Hypothesis 2. *In this paper, one message equals one frame i.e. there is no applicative fragmentation.*

Remark 1. *Let* τ_{f_l} *the transmission duration of* f_l *i.e. the duration between the emission of the first and the last bit of* f_l*. Since the message size per flow is constant, the transmission duration per flow is constant too.*

Definition 4 ($Ref(f_l)$)**.** *We define the* reference instant *of* f_l *as* $Ref(f_l) = l \times P_f$ *and thus the message* f_l *will be enqueued during the interval* $T_{SAP}(f_l) \in [Ref(f_l), Ref(f_{l+1})[$*.*

Definition 5 (P_{MAF})**.** *The system period is the hyper-period of all the flows and it is denoted by* P_{MAF}*.*

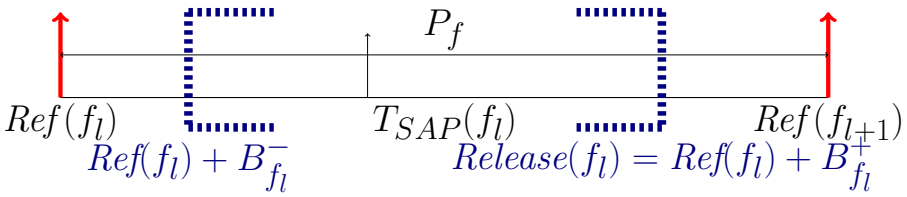


Figure 2: $Ref(f_l)$, $T_{SAP}(f_l)$ and $Prod(f_l)$

Applications come with a set of flow contracts, where each flow contract consists of a temporal window for messages production (see Fig. 2) so that they respect their performance, safety and development requirements (see Section 2.3). Such a contract is bargained off-line between applications and platform providers. It is expected that applications always respect their contracts and that the on-board network ensures the quality of service of each application as long as it fulfils its contracts.

Definition 6 (Application Flow Contract)**.** *Let* f_l *be the* l*-th message of* f*. The production contract associated to* f_l *is the interval* $Prod(f_l) = [Ref(f_l) + B^-_{f_l}, Ref(f_l) + B^+_{f_l}] \subseteq [Ref(f_l), Ref(f_{l+1})[$*, where* $B^-_{f_l}$ *(resp.* $B^+_{f_l}$*) is the earliest (resp. latest) production offset. The upper bound of* $Prod(f_l)$ *is called* Release instant *and denoted* $Release(f_l) = Ref(f_l) + B^+_{f_l}$*. The production traffic contract for a flow* f*, denoted* $Prod(f)$*, is defined by* $Prod(f) = \cup_{l \in \mathbb{N}} Prod(f_l)$*.*

This definition entails that $\forall f \in \mathbb{F}, \forall l \in \mathbb{N}, T_{SAP}(f_l) \in Prod(f_l)$ i.e. $Ref(f_l) + B^-_{f_l} \leq T_{SAP}(f_l) \leq Release(f_l)$.

Hypothesis 3 (Synchronization)**.** *We assume that emitters and egress nodes are synchronized and the synchronization error is insignificant with respect to the order of magnitude of the requirements presented hereafter.*

2.3 Flows' Requirements

Performance Requirement 1 (Deadline)**.** *Let a flow $f \in \mathbb{F}$, it comes with a deadline constraint so that $T_r(f_l) \leq f_l.deadline$ and $f_l.deadline \leq Ref(f_l) + P_f$.*

Definition 7 (Reception Jitter)**.** *The* reception jitter *[25] or* jitter *between two frames f_l and f_m is defined as the variability of their reception instants, it is denoted $Jit_{f_{l,m}}$ such that $\forall f \in \mathbb{F}, \forall l, m \in \mathbb{N}, Jit_{f_{l,m}} = |(T_r(f_l) - Ref(f_l)) - (T_r(f_m) - Ref(f_m))|$. The overall jitter of a flow is denoted Jit_f such that $\forall f \in \mathbb{F}, Jit_f = \max_{l,m} Jit_{f_{l,m}}$.*

Performance Requirement 2 (Jitter)**.** *A flow f also has a jitter constraint defined as $f.jitter \in \mathbb{N} \cup \{\text{NA}\}$ where NA stands for not applicable (thus no jitter constraint) and otherwise $f.jitter$ is the maximum accepted jitter.*

We refer to the flows with jitter constraints as *jitter flows* and to the others as *no jitter flows*, and we denote $\mathbb{F}_j = \{f \in \mathbb{F} | f.jitter \neq NA\}$ the set of jitter flows.

In addition to performance, safety requirements are often required. In particular ARINC 664 [1] or TTEthernet [24] networks offer Fault Isolation mechanisms. Among the faults supported by those networks, we restrict ourselves to message loss.

Definition 8 (Message loss independence)**.** *A system is considered as* message loss independent *if for all flow f, the loss of messages of f has no negative impact on the performance (deadline/jitter) of the other flows.*

Safety Requirement 1. *Any configuration of the network should fulfill the* message loss independence *requirement.*

A configuration shall also have the slightest impact on applications, meaning that the application development should be the least impacted by the network configuration.

Development Effort Requirement 1. *Any configuration should minimize the constraints on data production*[1]*, i.e. minimizing $B^-_{f_l}$ (ideally down to 0) and maximizing $B^+_{f_l}$ (ideally up to P_f).*

3 TSN configuration model

The embedded network is composed of several TSN or Ethernet-capable devices and links. A configuration is the composition of the local configurations of each device.

[1] see Def. 6

3.1 Output ports model

Each device (end-station or switch) is composed of a certain number of output ports. An output port is composed of up to eight internal queues, also known as *traffic classes*. These queues have priorities, and come with several mechanisms to do traffic shaping, bandwidth sharing, etc.

Definition 9 (Output port)**.** *We denote the set of output ports in the network by* $\mathbb{P}$*. An output port* $p = (q_0, \ldots, q_7, TS)$ *is composed of eight*[2] *internal queues* q_j *and a* Transmission Selection *(TS). Each queue* q *is associated with a Transmission Gate (*TG_q*).*

We summarize the output port model in Fig. 3. Both internal queues and TS will rule when frames access the medium. This TSN representation also applies to Ethernet output port with restricted options as detailed just after.
Transmission Selection Algorithm. TSN offers to add a Transmission Selection Algorithm (TSA) after each queue, but we do not consider them in this paper and just mention it for completness.
Transmission Gates. This mechanism, also referred to as *Time Aware Shaper*, adds the possibility for internal queues, in both switches and end-stations, to be regulated according to time-driven rules. In effect, there is a gate associated to each internal queue which can be opened or closed. The schedule switching from open to closed and back is pre-computed off-line, is periodic and is called a *Gate Control List* (GCL).

Definition 10 (Gate Control List)**.** *Let* p *a port, its associated* gate control list*, denoted* $GCL(p)$[3]*, is defined by the list* $[e_0, \ldots, e_m - 1]$ *of* m *events* $e_i = \langle s_i, t_i, d_i \rangle$ *where*

- $s_i = \langle s_{i,0}, \ldots, s_{i,7} \rangle$ *is the status of the gates* $s_{i,j} \in \{o, C\}$ *where* o *stands for open and* C *stands for closed,*
- $t_i \in \mathbb{N}$ *is the time offset from the start of* e_0 *at which event* e_i *starts.*
- $d_i = t_{i+1} - t_i$ *is the duration during which the gate state* s_i *will hold.*

In particular, the period of repetition of the pattern is $\sum d_i$ *and* $gcd(d_i)$ *is called gate granularity.*

Hypothesis 4 (Gate Control List period)**.** *The system we consider is periodic, it is sufficient to compute the gate schedules on the hyper-period of all its flows. Therefore,* $\sum d_i = P_{MAF}$*.*

Remark 2. *Ethernet-capable devices do not have transmission gates and this is equivalent to the gates being open all the time, i.e.* $\forall p, GCL(p) = [e_0] = [\langle \langle o, \ldots, o \rangle, 0, P_{MAF} \rangle]$*.*

[2]Without loss of generality, we assumed a fixed number of queues.
[3]This is a simplification of IEEE 802.1Q standard where *OperControlList* is the only considered parameter.

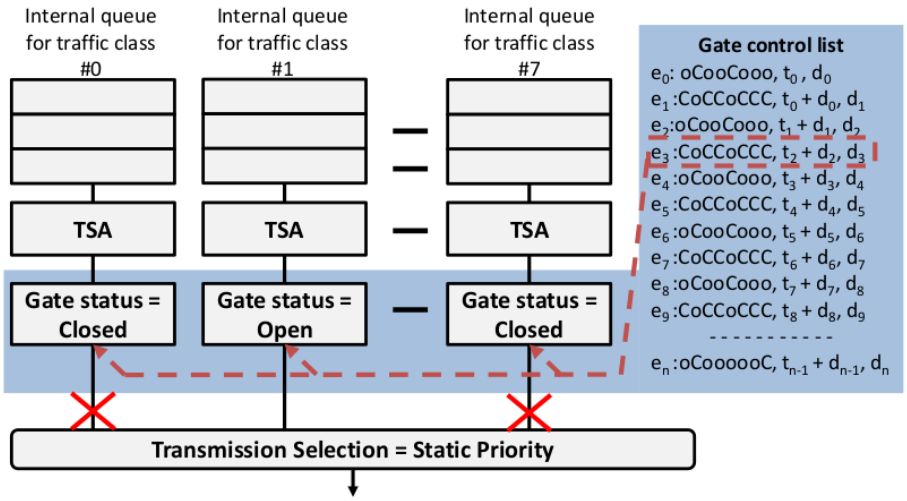


Figure 3: Scheduled Traffic Parameters

Transmission Selection A frame is emitted when it is available for transmission (cf. Def. 11) and has the highest priority among frames available for transmission (with #7 the highest priority and #0 the lowest).

Definition 11 (Frame available for transmission)**.** *A frame (or message) m in queue q of output port p is "*Available for transmission*" at instant t when:*

1. *The frame is the head of q,*
2. *TG_q is open at instant t,*
3. *TG_q remains open long enough to transmit the frame.*

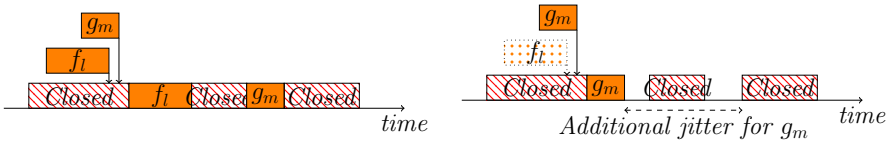
Remark 3 (Frame Preemption)**.** *We do not consider TSN standard for frame preemption [17] in this study. Such evolution could be done with inspiration from [29] by slightly redefining the equation of Def 11.*

3.2 Frame loss problem due to Transmission Gates

TSN relies on transmission gates mechanism schedules per queue and may generate non deterministic behaviour at message level in the presence of failure. This concept is illustrated with two figures: in Fig. 4a, the nominal expected behaviour (so as to cope with jitter requirements for g_m) is shown. Fig. 4b presents a scenario where message f_l is lost, leading g_m to be sent in place of f_l, creating an unwanted jitter.

This is not compatible with safety requirement 1. Two constraints exist in the literature [8] to cope with this issue.

- *flow isolation*: a queue is dedicated to a flow from its first to its last message in an hyper-period. Therefore, at each instant, only messages



(a) Nominal expected behaviour

(b) Message Loss

Figure 4: Need for Queue/Flow/Frame Isolation

from a single flow can be present in the queue and interleaving of frames from different flows is not allowed;

- *frame isolation*: a queue can be shared by several flows, but at each instant, only messages from a single flow can be present in the queue.

All cases prevent messages from different flows to be in the queue at the same time. Thus, a message loss cannot affect the behaviour of messages of other flows.

4 Problem Statement

We now formulate what computing valid configurations (i.e. satisfying requirement in section 2.3) means at system level.

4.1 System configuration

A system configuration is composed of a flow-level configuration and a network-level configuration.

Definition 12 (Flow configuration)**.** *The configuration of a flow f is $Config(f) = [(p_1, FtQM_{p_1}), \ldots, (p_l, FtQM_{p_l})]$ where*

- *$Path_f = p_1, \ldots, p_l$ is the path followed by f, that is the sequence of output ports that are crossed;*
- *$FtQM_{p_j}$ is the associated* Flow to Queue Mapping *on each port p_j. In particular, since a port is defined by $p = (q_0, \ldots, q_7, TS)$, $FtQM_p(f) \in \{q_0, \ldots, q_7\}$;*

Hypothesis 5. *We assume that the routing of the flows along the switches is fixed and static. Such an hypothesis is standard in the literature for embedded systems. This could be relaxed in a future work.*

A network-level configuration consists in finding a configuration for all the output ports.

Definition 13 (Port configuration)**.** *The configuration of a port $p \in \mathbb{P}$ is equivalent to finding a GCL configuration. Thus $Config(p) = GCL(p)$.*

In summary, computing a system configuration consists in determining:

$$\begin{cases} \forall f \in \mathbb{F}, \forall p \in Path_f, FtQM_p(f) \\ \forall p \in \mathbb{P}, GCL(p) \end{cases} \tag{1}$$

We want to compute *valid configurations* that fulfil all the requirements given in Section 2.3.

4.2 Optimization Criteria

The development effort requirement is ensured with an optimization criteria: maximize the production window (cf. Def. 6) of the configurations. We propose the following optimization criteria over a given window (the MAF):

$$maximize \sum_{\forall f \in \mathbb{F} \ s.t. \ f.jitter \neq NA} (Release(f_l) - Ref(f_l))^2 \tag{2}$$

Rationale 1. *We chose a quadratic cost function to reduce the solution to homogeneous solutions only, where no flow is compensating for another flow (e.g. one flow has a tiny production window and another flow compensate with a huge one) as requested by our industrial use case. This function could be modified depending on specific use case requirements.*

5 *Egress TT* Overview

The usual way to solve the problem presented in the previous section is to compute End-to-End TT configurations. Our approach, *Egress TT*, is slightly different and offers a better trade-off with respect to our requirements.

5.1 What is *Egress TT*?

Most of existing works focus on *End-to-End TT* configurations (fully time triggered) using the Transmission Gates mechanism [16].

Definition 14 (End-to-End TT configurations)**.** *All flows are scheduled in a time triggered way on all ports in their path. The reception and transmission instants of any frame of any flow in all input/output ports are fixed, and known a priori.*

Fig. 5 illustrates an End-to-End TT configuration with one emitter, one receiver and two switches (SWA and SWB). By fixing the transmission instant of all frames in all hops, the latency and jitter of the flows are controlled along the path. In order to cope with the safety requirement, most methodologies rely on *Flow Isolation* or *Frame Isolation*. Unfortunately, those require lower bound on the message emission (i.e. $B^-_{f_l} \neq 0$) and therefore, the existing End-to-End TT configurations do not comply with Development requirement 1.

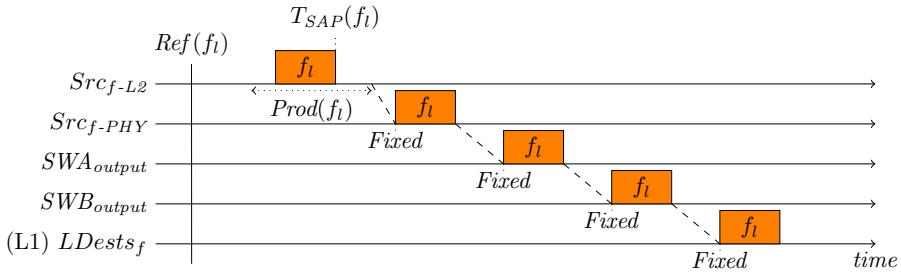


Figure 5: End-to-End TT Configuration

Definition 15 (*Egress TT* Configurations)**.** *Jitter flows are scheduled using Transmission Gates in last hop ports only. In all other output ports, the gates, if any, are always open.*

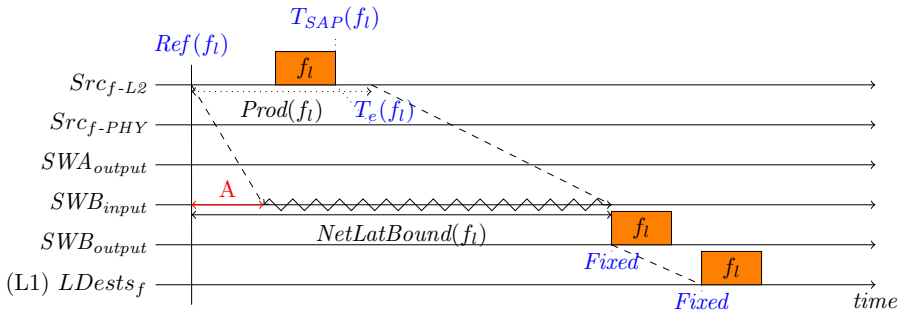


Figure 6: *Egress TT* Configuration

Fig. 6 illustrates an *Egress TT* configuration with one emitter, one receiver and two switches (SWA and SWB), the last hop being the output queue of SWB. Per definition 6, message f_l can be deposited at any time during the interval $Prod(f_l)$. Its emission date $T_e(f_l)$ is unknown a-priori. The network traversal delay of f_l can be bounded and let $NetLatBound(f_l)$ be such a bound. $NetLatBound(f_l)$ includes the delay from deposit ($T_{SAP}(f_l)$) to emission ($T_e(f_l)$). We illustrate theses instants and durations in Fig. 6 (A represents the best traversal delay). The purpose of these configurations is that whenever f_l is emitted by the application, it will be delivered to the destination end-station at a fixed time.

Practically, there is no time-triggered schedule before the last hop on the flow meaning that a message can encounter classical delays due to blocking by other flows. The last hop will be in charge of absorbing the upstream network jitter (if any) and delivering the message at the right time to satisfy the very low jitter requirement. To ensure a low jitter reception for f_l, it is sufficient to:

- be received in the correct queue of its last hop port before its schedule,

- be in head of that queue at f_l schedule,
- not be emitted to the destination before its schedule.

Definition 16 (*NetLatBound*)**.** *An upper bound on the worst case duration, from deposit to last hop emission, is denoted* $\forall f \in \mathbb{F}, \forall l \in \mathbb{N}, NetLatBound(f_l)$.

$NetLatBound(f_l)$ could be estimated for instance with classical worst case traversal time method such as *Response Time Analysis*[19] or *Network Calculus*[32]. In Def. 21, we propose a formula for the computation of $NetLatBound(f_l)$. This formula is quite simple and rather pessimistic but we believe it is sufficient to demonstrate the concept of *Egress TT* configurations. In fact, any method could be used to approximate $NetLatBound(f_l)$ as long as it is compatible with the tools used to generate the network configurations. In any case, the less pessimistic the bound is, the higher the chances to find suitable configurations will be.

5.2 Exclusive Queue Allocation and Size Based Isolation

In order to satisfy the safety requirement in *Egress TT* configurations for TSN networks, we introduce two constraints: *Exclusive Queue Allocation* and *Size Based Isolation*.

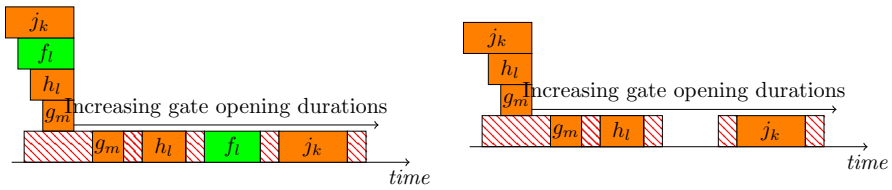
Definition 17 (Exclusive Queue Allocation)**.** *Each jitter flow is paired with one dedicated queue in its last hop port. No other flow can use that queue.*

Being alone in the queue removes the possible non-determinism induced by TSN Time Aware Shaper mechanism (see. 3.2). However, respecting Exclusive Queue Allocation comes at a cost: an end-station cannot receive more that eight jitter flows (or seven if it also receives no jitter traffic). With Size Based Isolation, we relax that constraint so that several messages from different flows are allowed to exist in the queue at the same time. However, it is necessary to manage the messages behaviour to satisfy the safety objective.

Definition 18 (Size Based Isolation)**.** *All frames sharing the same queue on last hop port shall be enqueued in increasing frame size order.*

By ensuring that frames are enqueued in increasing order (size might be artificially modified with padding at applicative level), if a frame is lost, the following frame will not be emitted in the slot of the lost frame since its size is bigger than the opening of the gate. Instead, the frame will be, as expected, emitted in its allocated slot. This concept is illustrated in Fig. 7: we show the nominal situation in 7a and the behaviour in case of message loss in 7b. Even when f_l is lost, j_k is not sent in place of f_l.

Being unable to impose an order between messages coming from different sources in the last hop port without a negative impact on the application development, we impose that flows sharing a queue in a last hop port shall come from the same emitter and share the same path. In this situation, the application impact is slightly increased: in addition to the traffic contract, the emitter will have to ensure an emission order.



(a) Nominal

(b) Loss of f_l

Figure 7: Isolation by Message Size

6 *Egress TT* Configurations

Let us now formalize how to compute valid *Egress TT* configurations for TSN networks.

6.1 First approach with Exclusive Queue Allocation

When a port is not a last hop, there is nothing much to do, hence we must focus on the *Last Hop Ports*.

Last hop ports. We distinguish the output ports which are the last hop of some flows and the others.

Definition 19 (Last Hop Ports)**.** *For a flow f following the path $p_1, \ldots, p_l$, we denote by $LH_f = p_l$ the last hop port. The set of last hop ports is $\mathbb{LH}= \{p \in \mathbb{P} | \exists f \in \mathbb{F}, LH_f = p\}$ and the set of last jitter ports is $\mathbb{LH}_j= \{p \in \mathbb{P} | \exists f \in \mathbb{F}_j, LH_f = p\}$.*

The configuration for the ports $\mathbb{P} \setminus \mathbb{LH}_j$ is equivalent to Ethernet-capable port configuration i.e. their gates are always open.

$$\forall p \in \mathbb{P} \setminus \mathbb{LH}_j, GCL(p) = \langle \langle o, o, o, o, o, o, o, o \rangle, 0, P_{MAF} \rangle$$

In any last hop, that is in port $p = (q_0, \ldots, q_7) \in \mathbb{LH}_j$, gate schedules follow an *exclusive gating pattern*[5]:

- jitter flows and no jitter flows are placed in different queues;
- At any time, either exactly only one jitter associated queue gate is *open* or several no jitter associated queues gates are *open*;
- if q_i is allocated to a jitter flow f: the gate is closed almost all the time. It is opened when a message f_l is scheduled and remains open during the message transmission duration (τ_{f_l});
- if q_i is allocated to no jitter flow(s): the gate remains always open except when one jitter associated queue is open.

Decision variables. The decision variables, i.e. the variables to which we are trying to find a value, should be those of equation 1: the flow to queue mapping

and the gate control list schedule for all output ports. Instead of computing GCL directly, we introduce an intermediate decision variable $SchedLH$.

Definition 20 ($SchedLH$)**.** *Let $f_l \in \mathbb{F}_j$ a jitter message, $SchedLH[f_l]$ denotes the instant at which the gate $FtQM_{LH_f}(f)$ shall be opened.*

From the variables $SchedLH$, it is possible to reconstruct GCL. Indeed, let us consider a jitter flow f and its last hop $LH_f = (q_0, \ldots, q_7) \in \mathbb{LH}$. f will produce P_{MAF}/P_f events: every time a frame of f is supposed to be transmitted, the gate should be open. More practically, for each f_l, there is an event $e =< s, SchedLH[f_l], \tau_{f_l} >$ where $s =< s_0, \ldots, s_7 >$ with $s_{FtQM_p(f)} = o$ and $s_j = C$ for $j \neq FtQM_p(f)$. Thus GCL is the union of all events associated to all jitter messages f_l where $LH_f = p$. This union is completed with gate opening of queues not allocated to jitter flows on the remaining time (when the jitter associated queue gates are closed). The gate events are generated by a post processing procedure. Since the system we consider is periodic, we only compute schedules on one system period (i.e. P_{MAF}).

Finally, the decision variables for the problem become:

$$\begin{cases} \forall f \in \mathbb{F}, \forall p \in Path_f, FtQM_p(f) \\ \forall f \in \mathbb{F}_j, \forall l < \frac{P_{MAF}}{P_f}, SchedLH[f_l] \end{cases} \tag{3}$$

Network Constraints. Across the network, the Flow to Queue Mapping differs. In last hop ports, while jitter flows are placed into different queues (due to Exclusive Queue Allocation), no jitter flows can share the same queues. In all other ports, flows are allowed to share the same queue.

Remark 4. *(Macrotick) To simplify the formulation of the equations in the rest of the paper, the instants and durations will be written in macroticks (like [8]). For instance, if the macrotick is the necessary duration to transmit a frame of 64 bytes, then for example $Size_f = 128 \implies \tau_{f_l} = 2$.*

Constraint 1 (Exclusive Queue Allocation)**.** *Each jitter flow is associated with one dedicated queue.*

$$\forall f \neq g \in \mathbb{F}_j, LH_f = LH_g \implies FtQM_{LH_f}(f) \neq FtQM_{LH_f}(g)$$

Links are modelled for the solver as two unidirectional links with opposite directions.

Constraint 2 (Link Occupation)**.** *A link can only send a message at a time in one direction i.e. $\forall f_l, g_m \in \mathbb{F}_j$ s.t. $LH_f = LH_g$:*

$$SchedLH[f_l] + \tau_{f_l} < SchedLH[g_m] \text{ or } SchedLH[g_m] + \tau_{g_m} < SchedLH[f_l]$$

Performance Constraints. All jitter flows are subject to deadline and jitter constraints.

Constraint 3 (Ordered Delivery)**.** *For any jitter flow, the i-th message shall be delivered before the (i+k)-th message of that flow, i.e.* $\forall f \in \mathbb{F}_j \forall l, m \in \mathbb{N}, l < m, T_r(f_l) < T_r(f_m)$*. This is translated as* $\forall f \in \mathbb{F}_j \forall l, m \in \mathbb{N}, l < m$*:*

$$SchedLH[f_l] < SchedLH[f_m]$$

Constraint 4 (Deadline)**.** *The delivery instant of a flow is bounded, indeed* $\forall f \in \mathbb{F}, \forall l \in N, Ref(f_l) \leq T_r(f_l) \leq f_l.deadline$*. This is translated as:*

$$\forall f \in \mathbb{F}_j, \forall l \in N, Ref(f_l) \leq SchedLH[f_l] + \tau_{f_l} \leq f_l.deadline$$

Constraint 5 (Jitter)**.** *For any jitter flow, the difference of latency of any two messages is bounded by the flow's jitter constraint i.e.* $\forall f, \in \mathbb{F}_j, \forall i \neq j \in \mathbb{N}, |Lat_{f_i} - Lat_{f_j}| < f.jitter$*. This is translated as* $\forall f, \in \mathbb{F}_j, \forall i \neq j \in \mathbb{N}$*:*

$$|SchedLH[f_i] - Ref(f_i) - (SchedLH[f_j] - Ref(f_j))| < f.jitter$$

Last Hop associated Constraints. In order to compute the last hop schedule, it is necessary to have an upper bound *NetLatBound* on the traversal time of flows until their last hop port.

Remark 5. *In this paper, a bound NetLatBound is estimated with Response Time Analysis because this method was directly implementable as a constraint in the solver. In fact, the solver needs to compute NetLatBound with every new configuration since NetLatBound depends on decision variables. As mentioned earlier, any other methods could be used to estimate that bound as long as it can be either integrated in constraints or coupled with the solver.*

Constraint 6 (Traversal Time Constraint)**.** *The release instant of any message of a jitter flow shall be within the flow's period. This is expressed as* $\forall f \in \mathbb{F}_j, \forall l \in \mathbb{N}$*:*

$$Ref(f_l) \leq SchedLH[f_l] - NetLatBound(f_l) < Ref(f_{l+1})$$

Consider a jitter flow f, and its l-th message f_l. With *Egress TT* configurations, in order to ensure f_l arrives in $FtQM_{LH_f}(f)$ before its schedule, the message must be sent after $Ref(f_l)$ and before $Release(f_l)$.

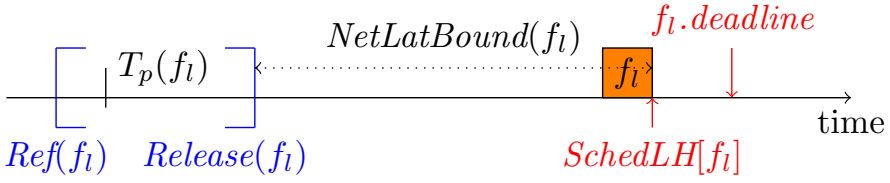


Figure 8: Release instants for jitter flows

We now explain how *NetLatBound* is computed.

Definition 21 (Bound on worst case latency *NetLatBound*)**.** *A bound on the worst case* deposit to last hop emission *latency is computed as* $\forall f \in \mathbb{F}, \forall l \in \mathbb{N}$*:*

$$NetLatBound(f_l) = \sum_{p \in Path_f, p \neq LH_f} \Delta(f_l, p) + \tau_{f_l}$$

where $\Delta(f_l, p)$ *is a bound on the worst case duration for* f_l *at output port* p*.*

In any port, f_l can be delayed, in the worst case, by several other messages. First, f_l can be delayed by all messages with same or higher priority than f_l but also one lower priority frame which arrived in the port before f_l. These delays are known as higher priority blocking $HPB(f_l, p)$, same priority blocking $SPB(f_l, p)$ and lower priority blocking $LPB(f_l, p)$). This delay model is inspired from [28, 2, 29].

Definition 22 (Bound on worst case duration $\Delta(f_l, p)$)**.** $\Delta(f_l, p)$ *is defined as* $\forall f \in \mathbb{F}, \forall l \in \mathbb{N}, \forall p \in Path_f, p \neq LH_f$*:*

$$\Delta(f_l, p) = \frac{HPB(f_l, p) + SPB(f_l, p) + LPB(f_l, p)}{r}$$

It is now necessary to determine which messages will be accounted for in HBP, SPB and LBP. In our system, all messages have a deadline smaller or equal to the end of their period (*implicit deadlines*). Therefore, a finite number of instances (i.e. frames) of each flow may be considered interfering with any message of a defined flow (cf. [27, 4]).

Definition 23 (List of contributing flows [4])**.** *Let* $FlowPort(p)$ *be the set of all the flows whose path includes* p *i.e.* $\forall p \in \mathbb{P}, FlowPort(p) = \{f \in \mathbb{F} | Path_f \cap p \neq \emptyset\}$*. For any message* f_l *of* $f \in FlowPort(p)$*, for every flow* $g \in FlowPort(p) \backslash \{f\}$*, there are at most* $\lceil \frac{P_f}{P_g} \rceil + 1$ *instances of flow* g *taking part in delaying* f_l*. This is illustrated in Fig. 9.*

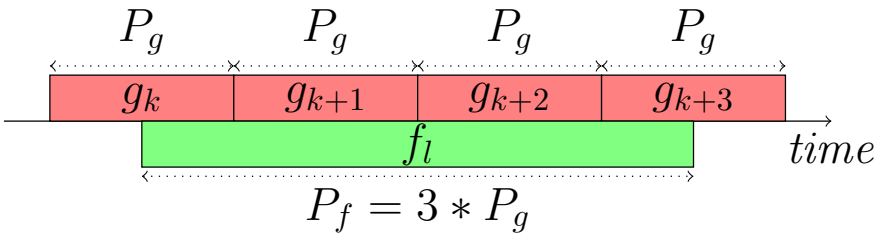


Figure 9: Contributing instances of g for the delay of f_l

Definition 24 (Blocking durations)**.** *We formulate the priority blocking dura-*

tions as follows: $\forall f \in \mathbb{F}, \forall l \in \mathbb{N}, \forall p \in Path_f, p \neq LH_f$*:*

$$
\begin{aligned}
HPB(f_l, p) &= \sum_{g \in FlowPort(p) | FtQM_p(g) > FtQM_p(f)} (\lceil \frac{P_f}{P_g} \rceil + 1) * Size_g \\
SPB(f_l, p) &= \sum_{g \in FlowPort(p) | g \neq f, FtQM_p(g) = FtQM_p(f)} (\lceil \frac{P_f}{P_g} \rceil + 1) * Size_g \\
LPB(f_l, p) &= \max_{g \in FlowPort(p), FtQM_p(g) < FtQM_p(f)} Size_g
\end{aligned}
\tag{4}
$$

6.2 Optimization criteria and post processing

Like in the state of the art (e.g. [9]), we encoded our problem as a set of decision variables and a set of constraints to be solved by a constraint solver. It is now necessary to encode the optimization criteria of equation 2. Therefore, this requires to compute the release instants for both jitter and no jitter flows.

Release(f_l) **for jitter flows.** $SchedLH[f_l]$ occurs exactly later after the worst case duration of f_l compared to $Release(f_l)$. Therefore:

$$\forall f \in \mathbb{F}_j, \forall l \in \mathbb{N}, Release(f_l) = SchedLH[f_l] - NetLatBound(f_l)$$

Release(f_l) **for no jitter flows.** $Release(f_l)$ is computed a posteriori via a post processing. Once the last hop emissions instants for jitter flows have been decided, the scheduling instants of no jitter flows are decided with the remaining port capacity (i.e. when gates for jitter flows are closed).

Remark 6. *Because the release instant for no jitter flows is computed a posteriori, it is necessary to check the correctness of that release instant that is* $\forall f \in \mathbb{F} \backslash \mathbb{F}_j, \forall l \in \mathbb{N}, Release(f_l) \geq Ref(f_l)$.

The last hop gate of no jitter flows is always open (except when some jitter message is being emitted and the output port is its last hop) and several no jitter messages may be in the same queue at the same time. Thus, the release instant is defined as $\forall f \in \mathbb{F} \backslash \mathbb{F}_j, \forall l \in \mathbb{N}$:

$$Release(f_l) = f_l.deadline - NetLatBound(f_l) - \Delta_{LH_f}^{WC+closed}(f_l)$$

where $\Delta_{LH_f}^{WC+closed}(f_l)$ denotes the worst case duration needed to transmit, in port LH_f, in queue $FtQM_{LH_f}(f)$, no jitter message f_l, including time for which the gate of $FtQM_{LH_f}(f)$ is closed.

A bound on the worst case duration $\Delta_{LH_f}^{WC+closed}(f_l)$ is determined with an algorithm not detailed in this paper due to lack of space and as it is quite straightforward.

6.3 Second approach with Size Based Isolation

Let us formulate the constraints for the second implementation of *Egress TT*, that is Size Based Isolation.

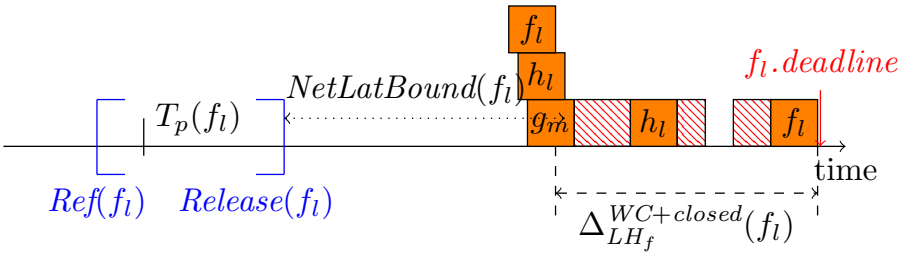


Figure 10: Release instant for no jitter flows

Additional Decision Variable. We add an additional decision variable *Padd* which translates the additional padding used to increase the size of frames.

Definition 25 ($Padd_{f_l}$)**.** *Let f_l a message of f, we define $Padd_{f_l}$ as an additional amount of bytes that is used to increase the size of f_l. In particular, we have:*

$$\forall f \in \mathbb{F}, \forall l \in \mathbb{N}, Size_f \leq Size_f + Padd_{f_l} < MTU_{Ethernet}$$

Remark 7. *The above formulation is generic and allows to take into account systems where periods are not harmonic. When periods are harmonic, it is possible to only compute a padding per flow instead of a padding per frame.*

Constraints We now extend the definition of τ_{f_l},

$$\forall f \in \mathbb{F}, \forall l \in \mathbb{N}, \tau_{f_l} = \frac{Size_f + Padd_{f_l}}{r}$$

Then, we reuse all the constraints from the first approach (i.e 2, 3, 4, 5 and 6) except Constraint 1. In addition we define two new constraints: *Size Based Isolation* and *QueuePerEmitter*.

Definition 26 ($Queue_p(i)$)**.** *Let $Queue_p(i)$ define the set of flows sharing the same last hop port and the same queue i.e. $\forall p \in \mathbb{P}, \forall i \in [0,7], Queue_p(i) = \{f \in \mathbb{F} | LH_f = p \text{ and } FtQM_p(f) = i\}$.*

Definition 27 ($f_l \sharp g_m$)**.** *Let $f_l \sharp g_m$ denote that f_l and g_m can interfere with one another i.e. that they exist in the same queue at the same time. Therefore, $f_l \sharp g_m \implies max(Ref(f_l), Ref(g_m)) < min(f_l.deadline, g_m.deadline)$.*

Constraint 7 (Size Based Isolation)**.** *All interfering messages in a last hop port shall be enqueued and transmitted in increasing message size order on last hop i.e. $\forall f, g \in Queue_p(i), \forall f_l, g_m$ s.t. $f_l \sharp g_m, SchedLH[f_l] < SchedLH[g_m] \implies \tau_{f_l} < \tau_{g_m}$*

In order to be able to control the reception order in the last hop port, we define an additional constraint:

Constraint 8 (Queue Per Emitter)**.** *Any two jitter flows having different source and same destination will be placed into the different queues i.e.* $\forall f, g \in \mathbb{F}_j$ *s.t.* $LH_f = LH_g, Src_f \neq Src_g \implies FtQM_{LH_f}(f) \neq FtQM_{LH_f}(g)$

The newly computed configurations allow a greater number of jitter flows per port; but not without a cost: in addition to the release instant, the emitting applications must follow an order constraint on emission so that their messages arrive in the correct order in the last hop port.

7 Comparison of *Egress TT* and End-to-End TT approaches

The purpose of this section is to evaluate our approach with respect to the state of the art. The comparison will be based on two criteria: *scalability* and *network latency*.

Both approaches were implemented in OPL and the computation was done using CPLEX v12.9.0 running on a Ubuntu computer embedding *Intel Xeon E5-2600 v3 @ 2.6GHz* and 62GiBytes of memory.

Remark 8. *For all the experiments of this paper, the network devices and links are supposed to work at 1Gbit/s.*

7.1 Scalability

In this section, we use the sets of constraints without the optimization criteria for *Egress TT* and our implementation of End-to-End TT with *Frame Isolation*. We compare the results provided by the solver for both approaches. To assess the scalability, we increase the number of switches on the paths and the number of receivers, and we monitor two metrics:

- *Number of constraints* necessary to generate a configuration,
- *Duration* of the computation to find a configuration.

Path size increase. In this first set of experiments, we consider a simple topology with one emitting end-station and one receiving end-station connected with a set of switches from 1 to 10 switches (cf. Fig. 11a). This allows to quantify the computation cost when adding a switch in the path.

Table 1 showcases the set of flows and their constraints. All flows have deadlines equal to their period. This set of flows comes from an industrial case study.

Fig. 11 presents the number of constraints and duration for the computation of one configuration. The number of constraints for both *Egress TT* implementations appears constant whereas it increases with End-to-End TT. This result was expected since, in End-to-End TT configurations, an emission instant has to be constructed for all frames in all hops of the network. In *Egress TT* the size or shape of the path of any flow is taken into account in *NetLatBound*.

Table 1: Set of flows $\mathbb{F}$

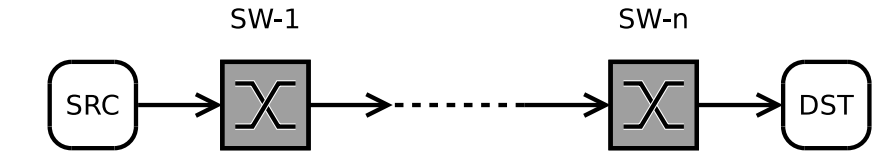
Name	**Period**	$f.\boldsymbol{jitter}$	$Size_f$	Bandwidth
f_1	125ms	NA	64	4Mbit/s
f_2	125ms	NA	512	32Mbit/s
f_3	250ms	NA	64	2Mbit/s
f_4	500ms	NA	1500	24Mbit/s
f_5	125ms	NA	128	8Mbit/s
f_6	125ms	NA	512	32Mbit/s
f_7	250ms	NA	512	16Mbit/s
f_8	125ms	NA	128	4Mbit/s
$f_9 - f_{13}$	125ms	1μs	64	4Mbit/s
f_{14}	125ms	500μs	256	16Mbit/s
f_{15}	125ms	500μs	512	32Mbit/s

A change in the path of any flow in *Egress TT* will only change the value of *NetLatBound* and not add any additional constraint. Thus the resolution time for *Egress TT* remains almost constant and thus much faster than End-to-End TT.

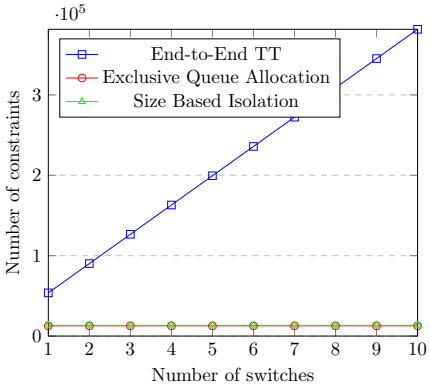
Remark 9. *In the experiment with 6 switches, the measured duration for End-to-End TT does not follow the trend of the other experiments (with different number of switches). We have no justification for this deviation and will investigate it in future works.*

Number of receivers increase. In the second set of experiments, we consider the same topology and increase the number of receiving end-stations from 1 to 6 (cf. Fig. 12a). This helps quantify the computation cost of adding an end-station. Each additional end-station will be receiving 15 flows with characteristics identical to those of Table 1, all emitted from *"Sender"*.

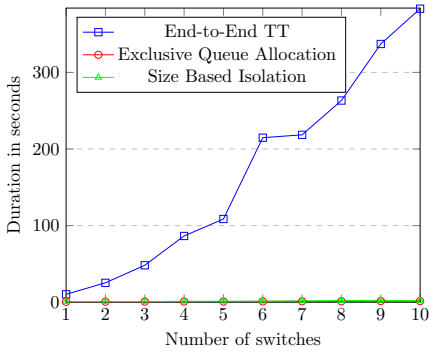
Fig. 14 presents the number of constraints and duration for the computation of one configuration. Again, the computation of a configuration is much quicker with *Egress TT* than End-to-End TT. While a End-to-End TT configuration of a network with 6 receivers will take roughly 75 minutes with our implementation, the *Egress TT* configuration of the same network will only take about 9 seconds with Exclusive Queue Allocation and 18 seconds with Size Based Isolation. Increasing the number of receivers, hence the number of flows, increases the number of constraints per hop for the decision of the emission instants. In *Egress TT* configurations, only the emission instants of the last hop switch in the path of any flow have to be computed. The impact of flows on each other is taken into account in *NetLatBound* and additional constraints are only added on last hops. Therefore the total number of constraints is lower and the computation time is also shorter.



(a) Topology pattern



(b) Number of constraints



(c) Duration in seconds

Figure 11: Path Size Increase experiment

7.2 Network Latency

The next experiment consists in running both End-to-End TT and *Egress TT* algorithms on the same use case (depicted in Fig. 13) while using the optimization criteria and compare the resulting network latency for both configurations. However, since our optimization criteria (brought by Development Effort Requirement 1) is not tackled by the literature, we replaced it by an optimization criteria aiming at minimizing network latency for End-to-End TT.

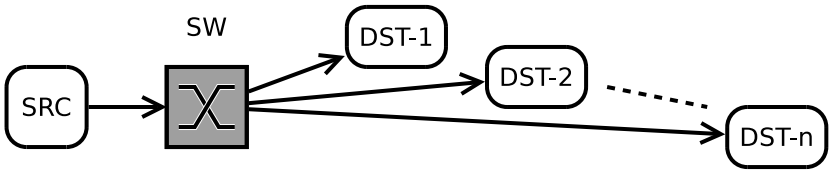
We consider a set of 4 data paths detailed in Table 2. On each data path, the source sends a set of 15 flows to the destination. Flows have the same characteristics as the ones of Table 1. On the following graphs, we will only

Table 2: Set of flows $\mathbb{F}$

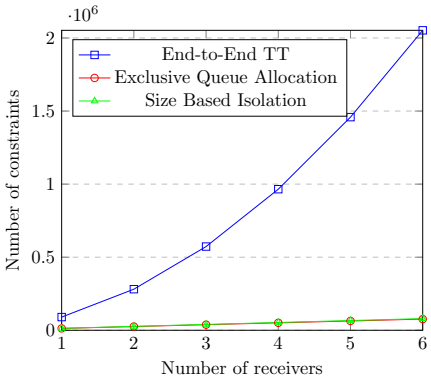
Id	Source	Path	Destination	Flows
①	ES_A	SW_A-SW_B-SW_D	ES_E	$f_1 ... f_{15}$
②	ES_B	SW_A-SW_B	ES_D	$g_1 ... g_{15}$
③	ES_C	SW_C-SW_A-SW_B-SW_D	ES_F	$h_1 ... h_{15}$
④	ES_D	SW_B-SW_D	ES_E	$i_1 ... i_{15}$

depict the average latency from data path ① and ④.

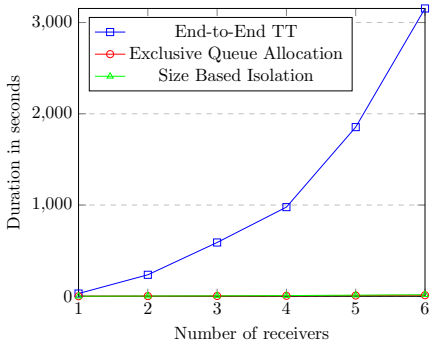
Due to the intrinsic limitation of Exclusive Queue Allocation (maximum of 7/8 jitter flows per last hop port), we have compared Exclusive Queue Allocation



(a) Topology pattern



(b) Number of constraints



(c) Duration in seconds

Figure 12: Number of Receivers Increase

with state of the art End-to-End TT only using data path ①, ② and ③. Fig. 14a shows the results of this first experiment. It took 5 seconds for *Egress TT* and 30 minutes for End-to-End TT to generate a valid configuration.

Then, data path ④ is added and Size Based Isolation and state of the art End-to-End TT are then compared. The experiment lasted 12 seconds for *Egress TT* and 30 minutes for End-to-End TT. The results of this second experiment are shown in Fig. 14b.

Experiments show that *Egress TT* configurations will lead to greater network latencies than End-to-End TT configurations. This increase of network latency is due to the definition of *Egress TT* configurations: a message is delayed by a

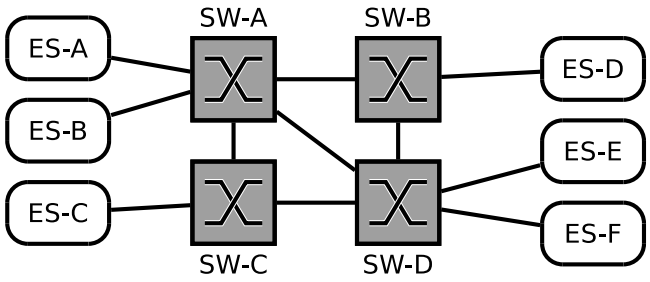
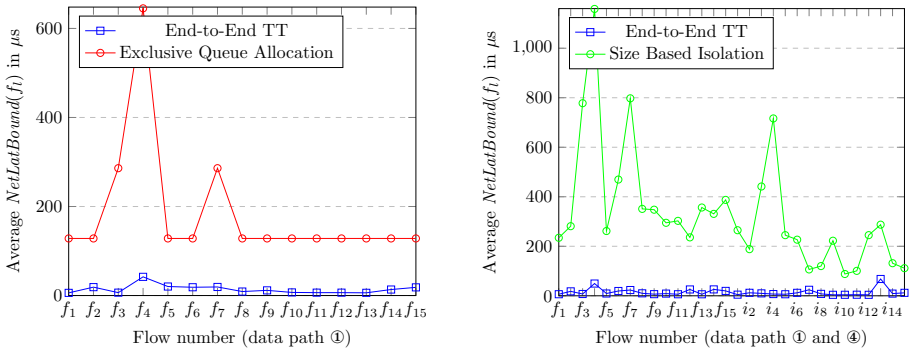


Figure 13: Topology for application impact use case



(a) End-to-End TT vs. Exclusive Queue Allocation

(b) End-to-End TT vs. Size Based Isolation

Figure 14: End-to-End TT vs. *Egress TT* comparisons

bound on its worst case latency so that it can always be delivered at the same time and meet its jitter requirement.

In summary, according to the experimental results above, *Egress TT* configurations reduce the computation effort (and computation time) compared to End-to-End TT configurations at the cost of a greater network latency while having, by design, a lower impact on applications.

It is important to insist on the meaning of these results: this paper compares the scalability and latency of two approaches (End-to-End TT and *Egress TT*) that were not designed for the same purpose. While End-to-End TT aimed at satisfying jitter requirements while minimizing network latency, *Egress TT* configurations aimed at satisfying jitter requirements while minimizing application impact and reducing the computation effort, based on the assumption that minimizing latency is not always required in implicit deadlines systems. Therefore, one solution is not better than the other. Rather, a network system designer will have the ability to choose, according to his needs, between one approach or the other.

7.3 ORION CEV Use Case

Finally, in one last experiment, we evaluate *Egress TT* on a use case adapted from the Orion Crew Exploration Vehicle (CEV) use case (use case and topology described in [31]). It is composed of 100 jitter flows and 86 no jitter flows. Although the model described in Section 2 supports multicast flows, our implementations of End-to-End TT and *Egress TT* that we have used for this paper do not. Therefore, we have duplicated multicast flows into several unicast flows. Thus, the use case is composed of 168 unicast jitter flows and 147 unicast no jitter flows.

Remark 10 (Multicast Support)**.** *There is an adverse effect in doing so: the number of messages in the network is unnecessarily increased, which might re-*

duce the possible configurations. However, it is sufficient to demonstrate the concept of Egress TT configurations. Improving the implementation for multicast support is relatively simple and will be proposed in future works.

Size Based Isolation offers the possibility to put in the same queue several flows with same source and same destination. However, in this use case, there cannot be any two flows with same source and same destination. Therefore, we will only experiment with Exclusive Queue Allocation.

In addition, because of the limitation of *Egress TT* configurations with Exclusive Queue Allocation(i.e. no port can receive more than 8 jitter flows), we can only consider 157 unicast jitter flows and 147 unicast no jitter flows. We compare this reduced use case with End-to-End TT.

Unfortunately, our implementation of End-to-End TT (state of the art), maybe too naive, did not allow us to compute, due to lack of memory resources, a configuration on the full set of flows of the Orion CEV use case like [32] did in their paper. Therefore, we were only able to obtain an End-to-End TT configuration on a reduced set of 60 flows. The generation of the configuration lasted about 4 hours in End-to-End TT and 18s with Exclusive Queue Allocation. The *Egress TT* configuration generation for the full size use case was successful and lasted 4 minutes. This experiment confirms our previous observation on scalability and latency. On the reduced use case, observed network latencies are, in average, 10 times greater in *Egress TT* than in End-to-End TT configurations.

7.4 Limitations of *Egress TT*

Egress TT with Exclusive Queue Allocation will always fail to find configurations when a device is supposed to receive more than 8 jitter flows. Indeed, this comes from the exclusive queue allocation since a queue is dedicated to one jitter flow.

Egress TT with Size Based Isolation will always fail when a device is set to receive flows coming from more that eight different sources. Again, this is due to the Size Based Isolation constraint and our objective to keep the application impact relatively low. In addition, the number of low-jitter flows per queue with Size Based Isolation will be limited by the gate granularity i.e. the smallest duration of a gate event. The maximum number of jitter frames than can be held in a queue at the same time $\chi_p(i)$ is computed with the following formula : $\forall p \in \mathbb{LH}_j, \forall i \in [0,7],\ if\ \exists f \in \mathbb{F}_j \in Queue_p(i), \chi_p(i) = \lceil \frac{Max_{size}}{gcd(d_i)} \rceil$ where $gcd(d_i)$ is the gate granularity (see Def. 10). For instance, with a granularity of $1\mu s$, the smallest open event will be able to transmit 125 bytes (i.e. $\frac{1\mu s}{8} * r$). Therefore considering the maximum frame size is 1518 bytes, this means that a queue can hold up to 13 (i.e. $\lceil \frac{1538}{125} \rceil$) low-jitter frames.

Egress TT will fail when the post processing on no jitter flows fails (i.e. deadlines of no jitter flows cannot be met).

Egress TT will fail when the computation of $NetLatBound(f_l)$ becomes too pessimistic: over-reservation of resources (*Egress TT*) is always less scalable that exact allocation (End-to-End TT).

In the above situations, among others, End-to-End TT will always be a better approach. Nevertheless, we believe that the improved scalability, in particular the shorter configuration time, as well as the lower application impact will still attract industrials towards *Egress TT*.

8 Related Works

The *Egress TT* approach shares similarities with Logical Execution Time (LET), but the aim in LET was to decouple the real timing constraints from scheduling [13], whereas ours is rather to avoid (over)constraining the system with static scheduling when is it not necessary.

The LETT proposal [3] presents the same approach, but the implementation is done in a middleware, whereas our contribution consists in using TSN network devices.

Finding a configuration for gate control lists in TSN networks is a NP-complete problem [6] and a hot topic in the networking community. Most configuration generation methodologies, based on Satisfiability Modulo Theory/Optimization Modulo Theory (SMT/OMT) solvers, root back to TTEthernet networks. We detail them hereafter.

A first approach in the state of the art creates a schedule per frame (or *frame offset*) for either jitter traffic or all traffic on all hops in the network. The pioneering work in [26] introduces a formal TTEthernet network model and an associated set of constraints for schedule generation, some of which we inspired from. The authors also make it clear that the computation of such schedule is expensive and introduce an incremental strategy for configuration generation. Exploiting the constraints of the previous paper, [6] [7] propose to create a schedule for both applications running on end-stations and the underlying TTEthernet network as well as new strategies to support the computation effort. Then, authors have started to consider network based on TSN instead of TTEthernet where the scheduling of frame is slightly different. In order to cope with the potential non-determinism induced by the loss of a frame, [8] adapts the constraints of [26] and introduces two new constraints namely *Flow Isolation* and *Frame Isolation* (cf. 3.2). The previously quoted papers create schedules for jitter traffic without any consideration on the remaining traffic in the network. Therefore, in order to improve the performance of no jitter traffic (i.e. latency requirements) [27], [10] and [14] introduce strategies, a priori or a posteriori, to modify the jitter frame schedule by either spacing the frame offsets or gathering them. [22] also proposes to add space between any two frame offsets but not in a no jitter performance consideration but rather, to leave time for potential retransmission of lost jitter frames.

Most recently, a second approach with configurations based on schedule per group of frames instead of per frame, motivated by TSN Transmission Gates *per queue scheduling* capability, has appeared. [9] applies its TTEthernet schedule generation methodology [7] to TSN networks. The authors introduce new sets

of constraints adapted for group of frames schedules as well as *Stream Isolation*, a fusion of *Frame isolation and Flow isolation* to again cover the loss of a frame. In [25], the same authors use their new constraints to implement a configuration generator and compare their two approaches (single frame offset v.s. group of frames offsets), showing the benefits of group of frames scheduling. More recently [23] proposes a group of frames configuration but chooses not to use exclusive gating like all other configuration generators. Moreover, it considers non-TSN end-stations (i.e. Ethernet) in their system. Based on previous constraints from [25] and new ones, they create a group of frames schedule satisfying temporal constraints for jitter flows and no jitter flows using schedule porosity in an incremental approach.

Another group of papers have chosen to take more variables into account for configuring TSN networks, in particular, several papers (e.g. [12, 11, 18, 20, 21]) deal with joint routing and scheduling configuration generator. This increases the solution space of frame schedules by allowing the route of flows to be modified. To compute these configurations, the authors not only rely on SMT/OMT based solver but also on heuristics. Recently [30] uses an heuristic to face the scalability issue. They can configure networks with 2000 nodes and 10000 flows. We do not detail further these papers since our work is based on ILP solvers and fixed route for all flows.

9 Conclusion

In this paper we have presented *Egress TT* configurations, a new way to configure TSN network which admits a variable travel time for messages in the network and constrains jitter only in the last output port in the path of any flow. *Egress TT* reduces the computation costs of a configuration and maximizes application production contracts (i.e. message emission scheduling flexibility) at the cost of increasing network latency. It also allows to reduce the number of TSN devices to only the very last switch in the path of any flow while the other devices rely on standard Ethernet. Currently, *Egress TT* configurations require Exclusive Queue Allocation or Size Based Isolation to reach the safety requirement of our system. Therefore it implies limitations on the number of flows per last hop port or per queue in a last hop port. While this solution may be sufficient in many use cases, we will aim at increasing the number of jitter flows per last hop port in future work.

References

[1] Aeronautical Radio Incorporated. ARINC Report 664P7-1 Aircraft Data Network, Part 7, Avionics Full-Duplex Switched Ethernet Network. Technical Report ARINC 664P7, 2009.

[2] Philip Axer, Daniel Thiele, Rolf Ernst, and Jonas Diemer. Exploiting shaper context to improve performance bounds of Ethernet AVB networks.

In *Proceedings of the 51st Annual Design Automation Conference*, DAC '14, page 1–6, New York, NY, USA, 2014. Association for Computing Machinery.

[3] Wojciech Baron, Anna Arestova, Christoph Sippl, Kai-Steffen Hielscher, and Reinhard German. LETT: An execution model for distributed real-time systems. In *2021 IEEE 94th Vehicular Technology Conference (VTC2021-Fall)*, pages 1–7. IEEE, 2021.

[4] Henri Bauer, Jean-Luc Scharbarg, and Christian Fraboul. Worst-case end-to-end delay analysis of an avionics afdx network. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '10, page 1220–1224, Leuven, BEL, 2010. European Design and Automation Association.

[5] Marc Boyer and Hugo Daigmorte. Impact on credit freeze before gate closing in cbs and gcl integration into tsn. In *Proceedings of the 27th International Conference on Real-Time Networks and Systems*, RTNS '19, page 80–89, 2019.

[6] Silviu S. Craciunas and Ramon Serna Oliver. SMT-Based task- and network-level static schedule generation for time-triggered networked systems. In *Proceedings of the 22nd International Conference on Real-Time Networks and Systems*, RTNS '14, page 45–54, New York, NY, USA, 2014. Association for Computing Machinery.

[7] Silviu S. Craciunas and Ramon Serna Oliver. Combined task- and network-level scheduling for distributed time-triggered systems. *Real-Time Syst.*, 52(2):161–200, March 2016.

[8] Silviu S. Craciunas, Ramon Serna Oliver, Martin Chmelík, and Wilfried Steiner. Scheduling real-time communication in IEEE 802.1Qbv time sensitive networks. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, RTNS '16, 2016.

[9] Silviu S. Craciunas, Ramon Serna Oliver, and Wilfried Steiner. Formal scheduling constraints for time-sensitive networks, 2017.

[10] Frank Dürr and Naresh Ganesh Nayak. No-wait packet scheduling for ieee time-sensitive networks (tsn). In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, RTNS '16, page 203–212, New York, NY, USA, 2016. Association for Computing Machinery.

[11] Voica Gavrilut, Bahram Zarrin, Paul Pop, and Soheil Samii. Fault-tolerant topology and routing synthesis for ieee time-sensitive networking. In *Proceedings of the 25th International Conference on Real-Time Networks and Systems*, RTNS '17, pages 267–276, New York, NY, USA, 2017. ACM.

[12] V. Gavriluţ, L. Zhao, M. L. Raagaard, and P. Pop. AVB-Aware routing and scheduling of time-triggered traffic for tsn. *IEEE Access*, 6:75229–75243, 2018.

[13] T.A. Henzinger, B. Horowitz, and C.M. Kirsch. Giotto: a time-triggered language for embedded programming. *Proceedings of the IEEE*, 91(1):84–99, 2003.

[14] Bahar Houtan, Mohammad Ashjaei, Masoud Daneshtalab, Mikael Sjödin, and Saad Mubeen. Synthesising schedules to improve qos of best-effort traffic in tsn networks. In *29th International Conference on Real-Time Networks and Systems (RTNS'21)*, April 2021.

[15] IEEE. IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks. Technical Report IEEE 802.1Q, 2008.

[16] IEEE. IEEE 802.1Qbv, Standard for Local and Metropolitan Area Networks-Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks Amendment: Enhancements for Scheduled Traffic. Technical report, 2016.

[17] IEEE. Ieee 802.3br-2016, ieee standard for ethernet, amendment 5: Specification and management parameters for interspersing express traffic and ieee 802.1qbu, ieee standard for frame preemption. Technical report, 2016.

[18] Sune Mølgaard Laursen, Paul Pop, and Wilfried Steiner. Routing optimization of avb streams in tsn networks. *ACM Sigbed Review*, 13(4):43–48, 2016.

[19] Dorin Maxim and Ye-Qiong Song. Delay analysis of avb traffic in time-sensitive networks (tsn). In *Proceedings of the 25th International Conference on Real-Time Networks and Systems*, RTNS '17, page 18–27, New York, NY, USA, 2017. Association for Computing Machinery.

[20] 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, 2018.

[21] Maryam Pahlevan, Nadra Tabassam, and Roman Obermaisser. Heuristic list scheduler for time triggered traffic in time sensitive networks. *ACM SIGBED Review*, 16:15–20, 02 2019.

[22] Francisco Pozo, Guillermo Rodriguez-Navas, and Hans Hansson. Schedule reparability: Enhancing time-triggered network recovery upon link failures. In *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 147–156, 2018.

[23] Niklas Reusch, Luxi Zhao, Silviu S. Craciunas, and Paul Pop. Window-based schedule synthesis for industrial ieee 802.1qbv tsn networks. In *2020 16th IEEE International Conference on Factory Communication Systems (WFCS)*, pages 1–4, 2020.

[24] SAE. AS6802 - Time-Triggered Ethernet. Technical Report SAE AS6802, 2016.

[25] R. Serna Oliver, S. S. Craciunas, and W. Steiner. IEEE 802.1Qbv gate control list synthesis using array theory encoding. In *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 13–24, 2018.

[26] Wilfried Steiner. An evaluation of SMT-based schedule synthesis for time-triggered multi-hop networks. pages 375–384, 11 2010.

[27] Wilfried Steiner. Synthesis of static communication schedules for mixed-criticality systems. In *Proceedings of the 2011 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*, ISORCW '11, page 11–18, USA, 2011. IEEE Computer Society.

[28] Daniel Thiele, Philip Axer, Rolf Ernst, and Jan R. Seyler. Improving formal timing analysis of switched ethernet by exploiting traffic stream correlations. In *Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis*, CODES '14, New York, NY, USA, 2014. Association for Computing Machinery.

[29] Daniel Thiele and Rolf Ernst. Formal worst-case performance analysis of time-sensitive ethernet with frame preemption. In *21st IEEE International Conference on Emerging Technologies and Factory Automation, ETFA'16*, pages 1–9, 2016.

[30] Marek Vlk, Katerina Brejchova, Zdenek Hanzalek, and Siyu Tang. Large-scale periodic scheduling in time-sensitive networks. *Computers and Operations Research*, 137:105512, 2022.

[31] Lin Zhao, Feng He, Ershuai Li, and Jun Lu. Comparison of time sensitive networking (tsn) and ttethernet. In *2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC)*, pages 1–7. IEEE, 2018.

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