A Unified Model for Integrated Modular Architecture-TSN based Systems

Matthias Houssin, Oana Hotescu, Frédéric Boniol

A Unified Model for Integrated Modular Architecture-TSN based Systems

Matthias Houssin
ONERA, ISAE-SUPAERO
Toulouse, France
matthias.houssin@isae-supaero.fr

Oana Hotescu
ISAE-SUPAERO
Toulouse, France
oana.hotescu@isae-supaero.fr

Frédéric Boniol
ONERA
Toulouse, France
frederic.boniol@onera.fr

***Abstract*—A recent trend in embedded industry is to mix Integrated Modular Architectures (IMA) with Time Sensitive Networking (TSN). IMA principles allow resource sharing between numerous software functions in a deterministic way. On the other side TSN is considered a promising communication solution for distributed architectures. However, TSN has not been fully exploited with IMA, while it would meet the increasing communication needs due to the explosion of the number of on-board software functions. In order to better understand the challenges of designing IMA-TSN systems, this paper proposes a joint model which unifies task and message management and allows evaluation of end-to-end latency properties. The model is illustrated by simulation on an avionic case study.**

***Index Terms*—Time-Sensitive Networking, Integrated Modular Architecture, system-network**

I. Introduction

Cyber-physical systems are composed of computing modules linked through one or several communication networks. The computing part of many of these systems in the automotive and avionic industry often follows the Integrated Modular Architecture (IMA) paradigm allowing resource sharing and time partitioning between software functions. Many frameworks have been proposed to standardized IMA principles, such as AUTomotive Open System ARchitecture (AUTOSAR) [1] in the automotive domain and ARINC653 [2] in the avionic domain. The deployment of these IMA standards has opened the possibility to significantly increase the missions and the complexity of on-board systems by allowing more software functions to be executed on fewer computing modules.

On the communication side, several industrial communication protocols exist on the market, depending on the application area, *e.g.,* FlexRay [3], TTEthernet [4] for automotive, EtherCAT [5], PROFINET [6] for industrial automation. For avionics, the choice has been set for years on ARINC 664-P7 Avionic Full-DupleX switched Ethernet (AFDX) [7].

In the past few years, IEEE 802.1Q Time-Sensitive Networking (TSN) [8] standard providing high data rate and low latency has emerged in industrial fields such as Industry 4.0, automotive and aerospace. TSN enhances real-time transmissions thanks to accurate time synchronization, stream reservation and scheduling of time-sensitive data, providing high reliability and security mechanisms.

The use of TSN in IMA introduces indeed new communication possibilities, however it requires re-thinking several aspects related to the configuration of processing elements to comply with the specificity of a particular application domain. In avionics for instance, the IMA modules are not synchronized with each other when using AFDX and synchronizing them with TSN may introduce new challenges related to the certification of the avionic systems [2]. High data rate and the extended number of traffic classes in TSN introduce the opportunity of new traffic types meeting the needs of increasing system software complexity. So, safe-critical and non-critical (*e.g.,* video, audio, etc.) applications that need to share the network bandwidth with different quality of service requirements have to be considered when deciding allocation and scheduling at both network and system level. Thus, adopting TSN in integrated architectures may imply optimisation of the system and network configuration (*i.e.,* service policies, routing, and task allocation), but also re-evaluating the use or not of some features such as synchronization, run-time reallocation or reliability enhancements.

In this paper, we propose a generic model unifying IMA and TSN network aspects. The main goals of this model are first, to enable studying how TSN and IMA can be configured together and secondly, to allow end-to-end timing analysis. Our main contribution consists in formalizing different scheduling policies for both IMA and TSN which is the basis for transversal configuration and analysis of IMA-TSN systems.

This paper is organized as follows. First, we present a background on IMA and TSN in Section II. Then, we introduce our unified model in Section III. Our model is illustrated by simulation in Section IV. Section V surveys related work. Finally, Section VI concludes the paper and gives leads for future work.

II. Background

A. IMA principles

The concept of Integrated Modular Architecture has been proposed in embedded domains involving more and more safety-critical functions. To reduce the weight and the number of the computing resources, the IMA principle relies on two main ideas: (1) resource and (2) temporal partitioning.

Resource partitioning means that for each software function is allocated a set of spatial resources (CPU core, memory area, DMA channel, etc.) in a static manner, that is to say that the resource integrator has the task of assigning the

maximum allowed resources to each function while respecting space segregation between them. Low-level mechanisms (at operating system level) provide protection for function data against any modification from the other functions.

Temporal partitioning means that the scheduling of software functions on each CPU core is defined off-line by a periodic sequence of slots statically organized in a time-frame named the MAjor time Frame (MAF). Each function is allocated a time slot for execution. At the end of this time slot, if not completed, the function is suspended and execution is given to the next function (according to the time-table). Thus, each function periodically executes at fixed times.

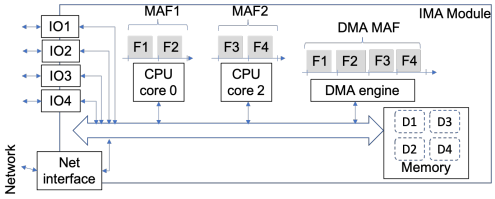


Fig. 1: Example of IMA computing module

To illustrate the IMA principle, let us consider the architecture depicted in Figure 1. It describes an IMA module composed of two CPU cores, a single memory, four IO ports and one DMA engine. The module hosts four functions: F_1 and F_2 on core 1 managed by MAF1, F_3 and F_4 on core 2 managed by MAF2. The memory is segregated into four parts. Each function is allocated one IO port. And finally, the single DMA engine is shared by the four functions according the local DMA MAF (data transfert of F_1 are done during the time slot associated to F_1, etc.).

The main advantage of IMA is to allow resource sharing while minimizing interference between functions for which reason such architectures are now widely used in automotive, space and avionic domains.

B. TSN

TSN is a set of standards based on switched Ethernet [8] which has been designed by the IEEE802.1Q task group to propose deterministic and reliable communication services. A TSN architecture is composed of a set of end systems and switches interconnected by communication links.

1) End system: In TSN, end systems communicate with each other through monitored Virtual Links (VLs). VL definition enforces a maximum emission rate of messages to be injected in the network. Generally, this rate matches the generation rate of periodic flows (Time Triggered traffic) or rate constrained flows (Event Triggered traffic). To enforce this constraint, end systems implement shaping policies.

2) TSN Switch: Frames arriving at switch are received through the ingress ports, then redirected by the switch fabric to an egress port according to their destination. The egress port classify the frames depending on their traffic class and dispatches them on the corresponding queue. A selection strategy is implemented to decide the order of transmission on the channel. Frames simultaneously enqueued suffer a queuing delay. Figure 2 illustrates the main functions of a TSN switch.

3) Service policies: TSN switches implements in egress ports a scheduling strategy combining a Credit-Based Shaper (CBS) [9] with a Time Aware Shaper (TAS) [10] and Strict Priority Queuing (SPQ) to decide frame selection on 8 priority levels. Our model handles TAS and SPQ while CBS will be left for future work. The TAS uses a Gate Control Lists (GCL). This policy considers 8 gates (one associated to each traffic class) which can be either open or close. So, the transmission of messages from a class may be allowed or not. The opening and closing of each gate is determined by a pre-established list such as illustrated on Figure 2. SPQ, a priority-based selective algorithm resolves the potential competition between the different traffic classes.

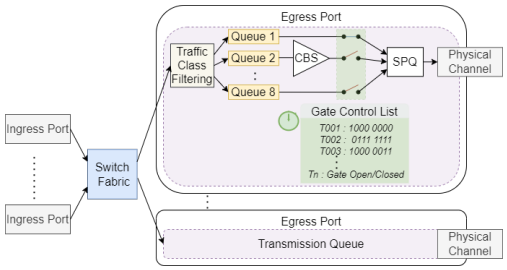


Fig. 2: TSN switch

III. IMA-TSN MODEL

Several models already handle unified task and messages, such as the one in [11] based on Steiner's model [12]. Our model extends it to TSN and IMA specificities.

The model is organized in 4 layers. First, the hardware elements are introduced at the platform layer (subsection III-A). The software elements (tasks and messages) are abstracted as generic flows at the system layer (subsection III-B). We introduce then at the configuration layer the scheduling strategies associated with each hardware element (CPU core, DMA, TNS port) (subsection III-D). Finally, the latency properties to be evaluated are defined over the whole model in subsection III-E. Notations used in the following are sum up in Table I.

A. Platform model

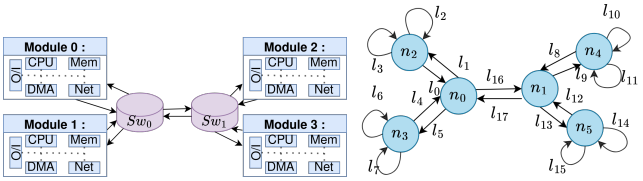
The first layer of the model describes the hardware view of the platform. Following the approach developed in [11], an IMA-TSN platform can be seen as a set of abstract *nodes* modeling in a unified way the computing modules and switches of the platform, connected through a set of abstract *links* modeling the processing and the networking components of the platform.

Definition III-A.1 (Platform)**.** Let P be an IMA-TSN platform, P is defined by $P :=< Nodes, Links >$ where

— *Nodes* is the finite alphabet of nodes of P (the IMA modules and the TSN switches),

— and *Links* is the finite alphabet of links in P (denoting the CPUs, the DMAs, and then network links).

Example III-A.1. Let us consider the platform depicted in Figure 3a. This platform is composed of four modules and two switches. Figure 3b describes the abstract model of the platform in which nodes n_2, n_3, n_4, n_5 represent the four modules in the real platform and nodes n_0 and n_1 represent the switches. Nodes are interconnected by network links such as l_0 and l_1, while the CPU and DMA components of modules become looping links on nodes such as l_2 and l_3 on node n_2.



(a) Real platform (b) Modeled platform

Fig. 3: Platform example

Definition III-A.2 (Node)**.** Let P be a platform. Let $n \in P.Nodes$ denote either a switch or a module. A node $n :=< offset, drift, d_{proc} >$ is characterized by three timing attributes:

- *offset* $\in \mathbb{Q}$ is the clock shift w.r.t. a reference clock;
- *drift* $\in \mathbb{Q}$ is the clock deviation w.r.t. a reference clock;
- $d_{proc} \in \mathbb{Q}$ is the processing delay due to addressing, packetizing or switching on module or switch n.

Definition III-A.3 (Relative offset)**.** Let $n_i, n_j \in P.Nodes$. We define the relative offset between n_i and n_j by: $\Delta_{offset}(n_i, n_j) = n_j.offset - n_i.offset$

Definition III-A.4 (Link)**.** Let P be a platform. Let $l \in P.Links$, where l denotes a processing or a networking element associated to a node or from a node to another one. Processing elements are CPU or DMA. Links are defined by $l :=< src, dst, d_p, q, type >$ where

- *src* $\in$ *Nodes* is the source node of the link;
- *dst* $\in$ *Nodes* is the destination node of the link;
- $d_p \in \mathbb{Q}$ is the link propagation delay;
- $q \in \mathbb{Q}$ is the link data-rate;
- *type* $\in$ {*CPU, DMA, Net*} indicates if the link represents a CPU, a DMA or a network link.

B. System Model

The second layer of the model involves the tasks and the messages hosted by the platform.

Definition III-B.1 (Task model)**.** Following work in [11], [13], a task hosted by a IMA module is a software object defined as $\tau :=< T, C, D, offset, prio >$ where:

- $T \in \mathbb{Q}$ is the task period, $C \in \mathbb{Q}$ is the worst-case execution time (WCET), and $D \in \mathbb{Q}$ is the deadline;
- *offset* $\in \mathbb{Q}$ is the task's offset w.r.t. the module's starting time;
- *prio* $\in \mathbb{N}$ is the task's priority used by the task scheduler.

Definition III-B.2 (Message)**.** A message m is an object sent through the network by a task τ. $m :=< len, prio >$ is characterized by two attributes: *len* $\in \mathbb{N}$ is the message length (in bytes) and *prio* $\in \mathbb{N}$ is the priority of the message.

Node: offset, drift, processing delay	$n.offset$, $n.drift$, $n.d_{proc}$
Link: propagation delay, data rate, type	$l.d_p$, $l.q$, $l.type$
Task: period, WCET, deadline, offset, priority	$\tau.T$, $\tau.C$, $\tau.D$, $\tau.offset$, $\tau.prio$
Message: length, priority	$m.len$, $m.prio$
Flow tree: links, set of directions, root link	ft, L, $next$, $root(ft)$
Flow: task, message, flow tree	$f.\tau$, $f.m$, $f.ft$
Flow instance, set of instances of a flow	f_k^l, $I(f)$
Set of all instances of all flows	$I(Flows)$
Flow workload, flow priority	$wl(f,l)$, $prio(f,l)$
System: platform, set of flows	$Sys.P$, $Sys.Flows$
State of a system at time t	$state(Sys,t)$
State predicates	$arr(f_k^l,t)$, $trans(f_k^l,t)$, $del(f_k^l,t)$
Scheduling strategy, policy	SS, pol
Functional chain, Data	FC,δ

TABLE I: Definition of symbols and notations

Definition III-B.3 (Flow-Tree)**.** Let be a platform P. A flow-tree ft in P is a tree of links. $ft := G(L, next)$ where $L \subset P.Links$, and $next \subset L \times L$ is a successor relation, such that *next* defines a tree of links. To start and the end links in a flow-tree must be processing elements (CPU or DMA). The start link of ft is denoted $root(ft)$.

Example III-B.1. Let us consider again the platform depicted in Figure 3. Let us suppose that a task τ hosted by *Module 1* sends a message m to *Module 0* and *Module 3*. The flow-tree is the following (where *next* is depicted by "$\rightarrow$"):

$$l_6 \rightarrow l_7 \rightarrow l_4 \rightarrow \begin{array}{l} \nearrow \; l_1 \rightarrow l_3 \\ \rightarrow \; l_{16} \rightarrow l_9 \rightarrow l_{11} \end{array}$$

It starts from l_6 (the CPU link of *Module 1*), it then involves link l_7 (DMA link of *Module 1*), and then link l_4 (communication link from *Module 1* to *Sw 0*). Then the flow-tree forks to *Module 0* (link l_1) and to *Module 3* (link l_{16}).

Definition III-B.4 (Flow)**.** A flow f is defined as $f :=< \tau, m, ft >$. It denotes the periodic action of a task τ sending a message m through the path ft. For platform P, let us denote:

- *Flows* the set of flows hosted by P;
- f_k^l is the k-th instance of f considered on link l. It may be either an instance of task or message;
- $I(Flows) = \{f_k^l | f \in Flows, l \in f.ft.L, k \in \mathbb{N}\}$;
- $I(f)$ the set of all instances of the flow f.

Let be P a platform, $l \in P.Links$ and $f \in Flows$. A link l in the flow tree $f.ft$ is occupied by f during a certain amount of time. This duration is the workload wl defined as:

Definition III-B.5 (Workload)**.**

$$wl(f,l) = \begin{cases} f.\tau.C & \text{if } (l \in f.ft.L) \wedge (l.type = CPU) \\ \frac{f.m.len}{l.q} & \text{if } (l \in f.ft.L) \wedge (l.type \in \{DMA, Net\}) \end{cases}$$

If the link is a CPU link, the task corresponding to flow f is executed, so the workload is given by its WCET. In case of network and DMA links, the workload is the time required by f to be sent on l. It depends on the $f.m.len$ and $l.q$[1].

[1]This supposes that the execution time through the DMA is linear with the length of the flow to transmit.

A priority has been associated for a task, respectively for a message. In order to unify these concepts, we extend the definition of the priority for a flow f on a link l. Let be a platform P, $l \in$ *P.Links* and $f \in$ *Flows*, the priority of f on l is defined as follows:

Definition III-B.6 (Priority)**.**

$$prio(f,l) = \begin{cases} f.\tau.prio & \text{if } l.type = CPU \\ f.m.prio & \text{if } l.type \in \{DMA, Net\} \end{cases}$$

Definition III-B.7 (System)**.** A system $Sys :=< P, Flows >$ is a platform P hosting a set of flows *Flows*.

C. State of a system

This section models the state of the system at time t through the states of its flows instances f_k on all the links l at time t: $state(Sys,t) = \prod_f \prod_l \prod_k (state(f_k^l, t))$. During its transfer through a link l, f_k^l goes through multiple states. To describe these states, we introduce three predicates: *arrival*, *transmission* and *delivery*. The state of the k-th occurrence of f on link l at a given time t is denoted by:

$$state(f_k^l, t) = \left(\ arr(f_k^l, t),\ trans(f_k^l, t),\ del(f_k^l, t) \ \right)$$

Transmission predicate, $trans(f_k^l, t) \in \{0, 1\}$, is true if the flow instance embodies a message being sent or a task being executed. The transmission selection follows a scheduling strategy. The scheduling strategy will be discussed later in Subsection III-D. The transmission predicate lasts the time needed to process the task or the message (*i.e* the workload). During transmission, only one flow instance can be sent at a time as formulated in the following equation:

Constraint III-C.1 (Exclusive transmission)**.**

$$\forall t \in \mathbb{R}^+, \forall f_k^l \in I(Flows) : trans(f_k^l, t) = 1$$
$$\Rightarrow \forall {f'}_{k'}^{l} \in I(Flows) \setminus \{f_k^l\}, trans({f'}_{k'}^{l}, t) = 0$$

Delivery predicate, $del(f_k^l, t) \in \{0, 1\}$, is true if the flow instance embodies a message being written in an ingress port or the result of an execution being written in memory. When the transmission starts, the flow instance f_k arrives on the next link after a propagation delay, so begins the delivery step.

Definition III-C.1 (Delivery)**.**

$$\forall t \in \mathbb{R}^+, \forall f_k^l \in I(Flows) :$$
$$delivery(f_k^l, t) = trans(f_k^l, t - l.d_p)$$

Both delivery and transmission end after the workload:

$$Let\ t, t_1 \in \mathbb{R} | \forall t < t_1, trans(f_k^l, t) = 0 \wedge trans(f_k^l, t_1) = 1$$
$$\forall t' \in \mathbb{R}, trans(f_k^l, t') = 1 \Rightarrow t' \in [t_1, t_1 + wl(f,l)]$$

Arrival predicate, $arr(f_k^l, t) \in \{0, 1\}$, is true if the flow instance embodies a message in a buffer or a task in ready state as represented in Figures 4a and 4b. When a flow instance f_k is arrived on link l, it was necessarily under delivery on the previous link l'. The delivery must have lasted a duration equal to the workload $wl(f, l')$ in addition to the processing delay $l'.dst.d_{proc}$ before f_k reaches link l. The flow instance remains in arrival during the queuing delay d_q as represented in Figure 4c. The arrival predicate becomes equal to 0 during transmission. The following equation defines the arrival predicate in relation to the delivery one.

Definition III-C.2 (Arrival)**.**

$$\forall t \in \mathbb{R}^+, \forall f_k^l \in I(Flows), \forall l' \in f.ft.L \mid (l', l) \in f.ft.next :$$
$$arr(f_k^l, t) = 1 \iff$$
$$\forall t' \in [t_1 - wl(f, l'), t_1] : delivery(f_k^{l'}, t') = 1$$
$$For\ t_1 = t - l'.dst.d_{proc} - \Delta_{offset}(l'.src, l'.dst)$$

D. Configured system

The constraints defined in Subsection III-C enables the model to describe all the state transitions allowed except the beginning time of the transmission of a flow instance.

The characteristics of the transmission are determined by the state of the system and the scheduling strategy of the link where the flow instance is stored at this time. In our model, we consider the main scheduling policies proposed by TSN such as the TAS-GCL, FIFO and Strict Priority Queuing (SPQ).

Definition III-D.1 (Scheduling policy)**.** The scheduling policy is defined by a rule and, possibly, parameters according to the rule $pol :=< rule, param, prio >$. This rule is a function defined for any f_k^l at a time t: $pol.rule(f_k^l, t, pol.param) = 1$ iff the rule elects the flow instance.

The rules are respected if some constraints are verified. Generic constraints (denoted GC1 and GC2) and rules specific behaviors are formalized as follows.

GC1: The selection of a flow instance by a policy is exclusive: only one flow instance can be elected at a time.

$$\forall t \in \mathbb{R}^+, \forall f_k^l \in I(Flows) : pol.rule(f_k^l, t, pol.param) = 1 \Rightarrow$$
$$\forall {f'}_{k'}^{l} \in I(Flows) \setminus \{f_k^l\}, pol.rule({f'}_{k'}^{l}, t, pol.param) = 0$$

GC2: A policy can not elect a flow instance that is not currently arrived on the link.

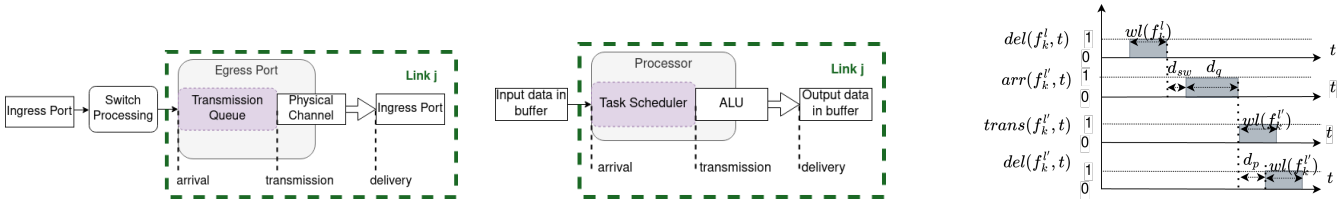
$$\forall t \in \mathbb{R}^+, \forall f_k^l \in I(Flows) :$$
$$pol.rule(f_k^l, t, pol.param) \leq arr(f_k^l, t)$$

Next, we formalize the behavior of usual TSN service policies. Following specific constraints are true only under the assumption that the generic constraints stated before are respected.

FIFO: Following equation expresses the behavior of a FIFO policy. If f_k^l is arrived before ${f'}_{k'}^{l}$ then there is a time t' when f_k^l is arrived and ${f'}_{k'}^{l}$ is not.

$$pol.FIFO(f_k^l, t) = 1 \iff [\forall {f'}_{k'}^{l} \in I(Flows) \setminus \{f_k^l\},$$
$$\exists t' \leq t \mid arr({f'}_{k'}^{l}, t') = 0 \wedge arr(f_k^l, t') = 1]$$

GCL: For GCL, the selection depends on a predefined schedule encoded in $pol.param$. The schedule is the aggregation of sending windows. Windows are defined by their offsets, their length and their period. The schedule is periodic with a period equal to the hyperperiod of the windows periods. We define



(a) Network link (b) Processing element link (c) Chronograph

Fig. 4: Link transit for a flow instance.

this schedule as: $\mathbb{1}_{pol.param}(t)$. The only condition for a flow instance to be elected by GCL is:

$$pol.GCL(f_k^l, t, pol.\mathbb{1}_{pol.param}) = 1 \iff$$
$$[arr(f_k^l, t) = 1 \wedge \forall t' \in [t; t + wl(f_k^l)] : \mathbb{1}_{pol.param}(t') = 1]$$

SPQ: Finally, SPQ or Fixed Priority (FP) can be summarized as "the most important comes first". This policy chooses between multiple flow classes of different priorities. Following equation implies that a flow instance f_k elected by SPQ may return to not elected if a higher priority flow instance arrives.

$$\forall f_k^l \in I(Flows), pol.SPQ(f_k^l, t) = 1 \iff$$
$$[\forall {f'}^l_{k'} \in I(Flows) \setminus \{f_k^l\},$$
$$arr({f'}^l_{k'}, t) = 1 \iff (prio(f) > prio(f'))]$$

Definition III-D.2 (Scheduling strategy)**.** Let be a platform P and $l \in P.Links$. The scheduling strategy of l defines the transmission order of flow instances on l according to policies. It is defined as a relation $SS(l) := \{(pol_{i_1}, pol_{j_1}), (pol_{i_2}, pol_{j_2}), \ldots\}$ denoting a graph tree of policies. For a given link l, we denote $pred^l(pol) = \{pol' | (pol', pol) \in SS(l)\}$ the precedence relation between policies in $SS(l)$.

An example scheduling strategy can be represented by:

$$\begin{array}{rcl} pol_1 \rightarrow pol_9 & \searrow & \\ \ldots & & pol_{19} \\ pol_8 \rightarrow pol_{17} \rightarrow pol_{18} & \nearrow & \end{array}$$

Each network link, CPU link and DMA link is associated with a scheduling strategy as illustrated on Figure 4.

GC2*: To run a scheduling strategy $SS(l)$, GC2 is replaced by the following generic constraint for policies which have a predecessor in $SS(l)$. This constraint means that to get the clearance from a policy pol, a flow instance must get a clearance from a policy among $pred^l(pol)$.

$$\forall t \in \mathbb{R}^+, \forall f_k^l \in I(Flows) : pol.rule(f_k^l, t, pol.param)(f_k^l, t)$$
$$\leq \sum_{pol' \in pred^l(pol)} pol'.rule(f_k^l, t, pol.param)(f_k^l, t)$$

Also in the rules behaviors presented before, the instances are now compared with instances elected by precedent policies:

$$pol.FIFO(f_k^l, t) = 1 \iff [\forall {f'}^l_{k'} \in I(Flows) \setminus \{f_k^l\},$$
$$\exists pol' \in pred(pol) \exists t' \leq t \mid pol'.rule({f'}^l_{k'}, t', pol'.param) = 0$$
$$\wedge\, pol'.rule({f'}^l_{k'}, t', pol'.param) = 1]$$

A policy with no predecessor is applied to a flow instance only if the flow priority matches the policy's priority:

$$Let\ pred^l(prio) = \emptyset : pol.FIFO(f_k^l, t) = 1 \iff$$
$$[\forall {f'}^l_{k'} \in I(Flows) \setminus \{f_k^l\} | prio(f', l) = pol.prio, \exists t' \leq t$$
$$\mid arr({f'}^l_{k'}, t') = 0 \wedge arr(f_k^l, t') = 1]$$

Example III-D.1. An example of a scheduling strategy composed of policies involved in TSN switch egress ports is given in Figure 2. This strategy can be represented by:

$$\begin{array}{rcl} FIFO_1 \rightarrow FIFO_9 \rightarrow GCL_{18} & \searrow & \\ FIFO_2 \rightarrow CBS_{10} \rightarrow GCL_{19} & \searrow & \\ \ldots & & SPQ_{28} \\ FIFO_8 \rightarrow FIFO_{17} \rightarrow GCL_{27} & \nearrow & \end{array}$$

E. Analysis

In this subsection, first we introduce concepts related to the execution of functions that will then serve to formalize system properties such as the functional delays on which analysis of real-time systems is usually based.

According to IMA specification [2], avionic or aircraft functions are capability provided by the hardware and software of the aircraft. It includes flight control, autopilot, fuel management, braking etc. In this work, we focus on the software part, which can be described as chains of applications.

Definition III-E.1 (Functional chain)**.** A functional chain FC is a directed acyclic graph (DAG) of flows. We note $(f, f') \in FC$ where $f, f' \in Flows$ to mean that f precedes f' in FC, that is, the data transported by f is transmitted to f'.

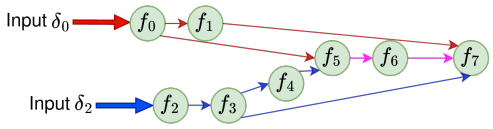


Fig. 5: Example of functional chain

Definition III-E.2 (Data)**.** A data $\delta \in Data$ is related to a source flow $f = \delta.srcFlow$. A new flow instance f_k corresponds to a new refresh of the value of δ, denoted δ_k.

Example III-E.1. An example of functional chain is given in Figure 5. In this functional chain, f_0 and f_2 are the source flows of the chain, *i.e.*, $f_0 = \delta_0.srcFlow$, $f_2 = \delta_2.srcFlow$.

Definition III-E.3 (Instance carrying a data)**.** To associate a data instance with a flow instance, we introduce the following function $carry_{FC} : I(Flows) \times Data \mapsto \{0,1\}$. For f, a flow in FC, if f_k carries δ_j then $carry_{FC}(f_k^l, \delta_j)$ is equal to 1 else it equals 0.

Data transmission can be achieved under two modes: queuing and sampling. For the sake of simplicity, we only present the sampling mode. However, both modes are taken into account in the model. A data instance δ_j is carried by a flow instance ${f'}_{k'}^{l'}$ with $l' = root(f'.ft)$ iff there exists a flow instance f_k^l verifying the conditions: (1) f precedes f', (2) f_k^l carries δ_j, (3) f_k^l is the last flow instance of f to be delivered on link l before the transmission of ${f'}_{k'}^{l'}$.

Definition III-E.4 (Data transmission (sampling mode))**.** The previous conditions are translated in the following equation:

Let t_1, $l' = root(f')$ and $l \in f.ft | l.dst = l'.src$ such as

$$\forall t < t_1, trans({f'}_{k'}^{l'}, t) = 0 \wedge trans({f'}_{k'}^{l'}, t_1) = 1 \wedge$$
$$(f, f') \in FC \wedge \Big[\exists t < t_1, (\forall t' \in]t - wl(f,l); t], del(f_k^l, t') = 1)$$
$$\wedge [\forall k'' \in \mathbb{N}, \forall t_2 < t_1, (\exists t' \in]t_2 - wl(f,l); t_2], del(f_{k''}^l, t') = 0)$$
$$\vee (del(f_k^l, t_2) = 1 \Rightarrow t_2 < t)]\Big] \wedge (carry_{FC}(f_{k''}^l, \delta_j) = 1)$$
$$\iff carry_{FC}({f'}_{k'}^{l'}, \delta_j) = 1$$

Real-time systems require a full control of the time between the arrival of a data and the corresponding output. This time is called "functional delay". Consider a functional chain FC and a data δ. Let f be a flow in FC and the source flow of δ, and f' a flow in FC depending on δ. A functional delay $FD_{FC,\delta_k,{f'}^{l'}}$ is the time between the arrival of the k-th instance of f and the delivery of the matching instance to f'. An instance k' of f' matches to instance k of f under the condition that ${f'}_{k'}$ carries the same data as f_k.

Definition III-E.5 (Matching instances)**.** Let $match \in \{0,1\}$ represent the matching condition ${f'}_{k'}$ carries δ_k at time t:

$$match(FC, {f'}_{k'}^{l'}, \delta_k, t) = carry_{FC}({f'}_{k'}^{l'}, \delta_k) \times del({f'}_{k'}^{l'}, t)$$

Definition III-E.6 (Functional delay)**.**

$$FD_{FC,\delta_k,{f'}_{k'}^{l'}} = \max_{match({f'}_{k'}^{l'},\delta_k)}(t) - \min_{arr(f_k^{root(f.ft)},t)=1}(t)$$

Among the functional delays, there are different specific delays. For example, [14] defines the *Age Delay* and *Reaction Delay*. According to [14], *Age Delay* is defined as "the time elapsed between the arrival of data at the input and the latest availability of the corresponding data at the output". In the current model, the *Age Delay* can be defined as:

$$AgeDelay(FC, \delta) = \max_{k,{f'}_{k'}^{l'}}(FD_{FC,\delta_k,{f'}_{k'}^{l'}}) \quad (1)$$

The *Reaction Delay* is defined in [14] as "the earliest availability of the data at the first instance of the output corresponding to the data that just missed the read access at the input". In the current model, *Reaction Delay* is:

$$ReactionDelay(FC, \delta) = f.\tau.T + \min_{k,{f'}_{k'}^{l'}}(FD_{FC,\delta_k,{f'}_{k'}^{l'}}) \quad (2)$$

IV. Case study

A. Experimental setup

1) Platform: To illustrate our model, we consider the platform in Figure 3 supporting three functional chains involving the flows defined in Table II: FC_0 = $\{(a,b),(a,c),(b,c),(c,d)\}$, FC_1 = $\{(e,f)\}$, FC_2 = $\{(g,h)\}$.

Flow	Task				Message		flow tree
	T	C	D	prio	len	prio	
a	8	2	8	3	200	7	$l_2 l_3 l_0 l_{16} l_9 l_{11}$
b	8	2	8	3	200	7	l_{10}
c	4	2	4	4	200	7	$l_{10} l_{11} l_8 l_{17} l_5 l_7$
d	2	1	2	3	-	-	l_6
e	8	2	8	2	200	6	$l_{10} l_{11} l_8 l_{17} l_5 l_7$
f	4	1	4	2	-	-	l_6
g	1	0.5	8	0	1500	0	$l_{14} l_{15} l_{12} l_{17} l_1 l_3$
h	1	0.5	8	0	-	-	l_2

TABLE II: Flows parameters

The first chain is representative of an avionic chain. Consider a is an ADIRS function (computing the Mach number of the aircraft). a runs on module M_0. It sends it to the automatic pilot (b, on M_2) and to the flight control function (c, on M_2). b computes and sends the flight objective to c. Finally, c computes and sends the angles to apply to the flight surfaces to d (the flight surface control laws, on M_3). To master the age delay of such a chain is safety-critical. The second and third chain are composed of only two tasks (e on M_2 and f on M_1 for FC_1, and g on M_3 and h on M_0 for FC_2).

2) Configuration of the scenarii: We study three configurations. i)In the simplest one, the flow instances are en-queued in different FIFO queues depending on their priority. The output of these FIFO queues are submitted to the selection of SPQ. ii)The second configuration considers a partitioned schedule on CPU links and the SPQ schedulers on the network links. For CPU links, flow instances are first sorted among FIFO queues depending on their priorities. A set of priority levels represents a partition. A SPQ policy is used to select an instance among the ones in the output of the queues of the partition. Finally, a GCL policy realizes the time-segregation between the different partitions, to do so the windows of each partition must not overlap. iii)The last configuration is using the partitioned schedule for CPU links. Network links are configured as follows: FIFO queues sort the flow instances by priority, a GCL policy per priority enables or disables a flow instance from a priority to be sent, and a SPQ policy selects the instance with the highest priority among the enabled instances.

B. Simulation results

We used an in-house simulator[2] to compute the behavior of the system considering each configuration. The windows

[2]https://github.com/houssima/IMA-TSN-Simulation

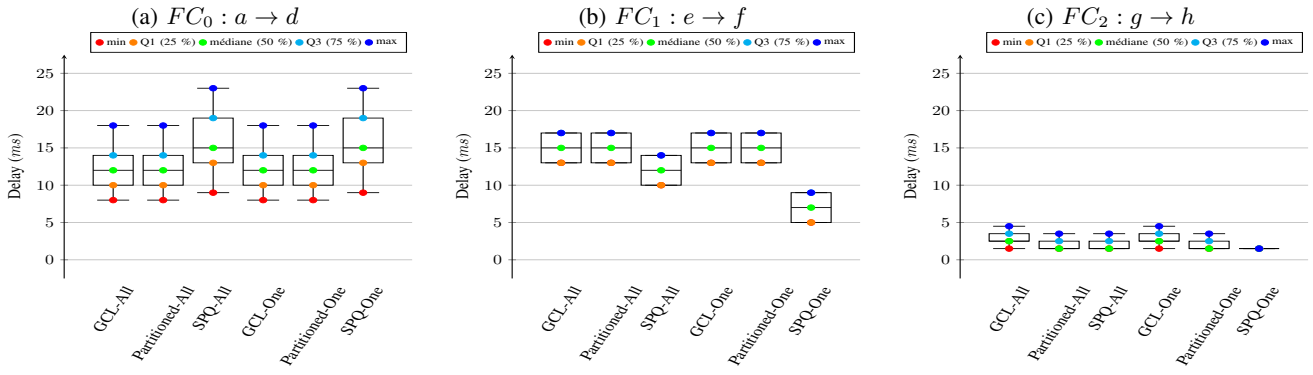


Fig. 6: Functional delays for scenarios considering GCL, partitioned configuration and SPQ

for the GCL and partitioned scenario have been designed in a naive way, hence, they are not optimal.

The simulation results show the distribution of the functional delay for each configuration scenario in Figure 6. For each configuration, two cases are considered: functional chains alone (denoted $-One$) or sharing the architecture with other functional chains (denoted $-All$). It can be noticed that when there is interaction between chains, with SPQ FC_0 is not impacted as it has the highest priority, but FC_1 and FC_2 suffer an increase in their delays. A favorable impact is noticed for FC_0 when using windows to force the flows to wait for each other, as the GCL and the partitioned configuration have lower functional delays than the SPQ scenario. Based on these results and the Equations 1 and 2, the age and reaction delay of each chain in each scenario can be easily retrieved. For example, the maximal delay of FC_0 in $GCL - All$ scenario is $18ms$, then its age delay is $18ms$. The lowest delay is $8ms$, and the period of the entry flow is $8ms$, hence the reaction delay is $16ms$. Notice that the sum of all execution times of FC_0 is $7ms$, and the network traversal time is around 130μs. Better reaction delay and lower age delay may be expected with tighter windows.

V. Related Work

The model presented in this paper considers IMA-TSN-based systems and is oriented towards temporal and spatial allocation of tasks and messages respectively on the IMA modules and inside the TSN network.

Several models have been proposed in the literature, modeling either IMA task allocation or network scheduling. In the avionics domain, works in [13], [15], [16] focus on end-to-end timing requirements of IMA functional chains and propose task models allowing allocation of avionics applications on modules and partitions. From the network point of view, these models consider the impact of the avionics de-facto AFDX network latency on the applications execution. Worst-case bounds on the latency are usually computed with analytical methods such as Network Calculus [17], Trajectory Approach [18] or model-checking based approaches [19].

In the automotive domain, [14] proposed a system-network model suited for generic real-time network protocol. The task model used in [14] is based on the model introduced in [20] for multi/many core platform, which considers a single module.

In the context of TTEthernet, *Steiner* [12] proposes a first model allowing to formulate constraints to avoid interaction between frames using the same communication links. Static scheduling of time-triggered messages is obtained by solving the optimization constraint problem with a Satisfiability Modulo Theories (SMT) solver. Other work in [21] extends the original model designed for TTEthernet for avionics requirements by integrating scheduling policies in the SMT optimization program, but does not consider task execution. Based on the Steiner's model, *Craciunas et al.* [11] explore the idea of combining task and TTEthernet network in the same model. This study raised the problem of the high complexity of scheduling a large network.

All these previous works assume TTEthernet network constraints in the proposed system-network model. To the best of our knowledge, only few recent works consider IMA and TSN together. The work in [22] uses the model in [11] to compute system and network schedules for an automotive TSN use case with simulated annealing and genetic algorithms. In the context of avionics, recent work in [23] optimizes the end-to-end delay at partition level by unifying avionic task execution and TSN network message transmission constraints.

Our paper intends to unify task and message modeling as well as the works in [11], [22], [23], and [14]. Such models consider the system as a whole by introducing functional latency, which is also done in other works [20], [24]. The functional latency depends on precedencies in the functional chains, and excepting [24], all these previous works consider single-dependency between tasks. Our model extends the concept of data proposed by [20] to have both unified task and messages and multiple dependencies in the functional chain. Also, these works focus on a single traffic type, while in our model we consider mixed traffic types. Modular architecture and mixed-criticality systems implies the definition of partitions or at least time segregation. This aspect addressed in [20], [24] is integrated in our model as well. Task preemption is an important feature in mono-processors RTOS, while frame preemption at switches comes with an important cost in terms

of implementation complexity and overhead [25]. To address preemption, the approaches proposed in [11] and [23] are based on substitutes which are very costly for upcoming optimisation strategies while in [22] task preemption is applied with EDF during optimisation. Given these implementation costs, in our model we do not address preemption yet. IMA architecture is usually based on mono-processor platforms, as modeled in [11], [24] and [14]. However, the industrial world is expecting the use of multi-processor in critical systems [22], [23]. Our model is compatible with both mono-processor and multi-processors platforms. In the works mentioned above, the impact of scheduling policies, especially the combination of policies, which is one of the main challenges of TSN, is poorly discussed. Such analysis is only performed with network calculus approaches such as [26], [27]. However, network calculus is a pessimistic analysis, which gives an upper bound of the worst case. Our work computes exact worst latencies under simplifying hypothesis. Even without hypothesis considered, our approach can approximate the realised worst cases. Most of the last models in the related work presented here are intended for optimisation of the allocation and scheduling. Our model is intended for both optimisation and simulation such as the one in [11].

VI. Conclusion and discussion

One of the benefits of the approach presented above is to take into account in the same model both TSN and IMA configuration. A second benefit is the modularity of the model, allowing integration of other scheduling policies (CBS, RR for TSN elements and non-preemptive EDF for IMA elements).

However, several questions still arise. The first one is about the validity of the model, that is, how to guarantee that the encoded behaviour is correct w.r.t. the IMA and TSN standards. This issue is out of the scope of the paper. Two solutions are possible. First, to test the model w.r.t. real execution on a real platform through a set of covering scenarii. The second solution is to test the model w.r.t. a second model which has been previously qualified. To that purpose a perspective could be to adapt the IMA-TSN model into a OMNeT++ simulator which offers a detailed TSN framework (synchronization, reliability and network management protocols).

A second perspective will investigate an optimization loop, based on meta-heuristic (such as genetic algorithm) in order to jointly optimize TSN and IMA configurations for a given platform w.r.t. age and reaction delays of functional chains.

References

[1] S. Fürst, J. Mössinger, S. Bunzel, T. Weber, F. Kirschke-Biller, P. Heitkämper, G. Kinkelin, K. Nishikawa, and K. Lange, "AUTOSAR–A Worldwide Standard is on the Road," in *14th International VDI Congress Electronic Systems for Vehicles, Baden-Baden*. Citeseer, 2009.

[2] Aeronautical Radio Inc. ARINC Specification 653 P1-3, "Avionics application software standard interface: Required services," 2013.

[3] R. Makowitz and C. Temple, "Flexray-a communication network for automotive control systems," in *2006 IEEE International Workshop on Factory Communication Systems*. IEEE, 2006, pp. 207–212.

[4] SAE(Society of Automotive Engineers), "SAE AS6802: Time-Triggered Ethernet," 2016.

[5] D. Jansen and H. Buttner, "Real-time Ethernet: the EtherCAT solution," *Computing and Control Engineering*, vol. 15, no. 1, pp. 16–21, 2004.

[6] J. Feld, "Profinet-scalable factory communication for all applications," in *IEEE International Workshop on Factory Communication Systems, 2004. Proceedings.* IEEE, 2004, pp. 33–38.

[7] Aeronautical Radio Inc. ARINC specification 664 P7-1, "Aircraft Data Network, Part 7: Avionics Full Duplex Switched Ethernet (AFDX) Network," 2009.

[8] IEEE, "802.1Q - IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks," 2018.

[9] ——, "IEEE Standard for Local and metropolitan area networks– Virtual Bridged Local Area Networks Amendment 12: Forwarding and Queuing Enhancements for Time-Sensitive Streams," 2009.

[10] ——, "IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic," 2015.

[11] S. S. Craciunas and R. S. Oliver, "Combined task- and network-level scheduling for distributed time-triggered systems," *Real-Time Systems*, vol. 52, no. 2, pp. 161–200, Mar 2016.

[12] W. Steiner, "An evaluation of SMT-based schedule synthesis for time-triggered multi-hop networks," in *2010 31st IEEE Real-Time Systems Symposium*. IEEE, 2010, pp. 375–384.

[13] N. Badache, K. Jaffres-Runser, J.-L. Scharbarg, and C. Fraboul, "Managing temporal allocation in integrated modular avionics," in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*. IEEE, 2014, pp. 1–8.

[14] M. Ashjaei, N. Khalilzad, S. Mubeen, M. Behnam, I. Sander, L. Almeida, and T. Nolte, "Designing end-to-end resource reservations in predictable distributed embedded systems," *Real-Time Systems*, vol. 53, pp. 1–41, 11 2017.

[15] M. Lauer, J. Ermont, F. Boniol, and C. Pagetti, "Worst case temporal consistency in integrated modular avionics systems," in *2011 IEEE 13th International Symposium on High-Assurance Systems Engineering*, 2011, pp. 212–219.

[16] A. Al Sheikh, "Resource allocation in hard real-time avionic systems: scheduling and routing problems," Ph.D. dissertation, Toulouse, INSA, 2011.

[17] F. Frances, C. Fraboul, and J. Grieu, "Using network calculus to optimize the AFDX network," 2006.

[18] H. Bauer, J.-L. Scharbarg, and C. Fraboul, "Improving the worst-case delay analysis of an afdx network using an optimized trajectory approach," *IEEE Transactions on Industrial informatics*, vol. 6, no. 4, pp. 521–533, 2010.

[19] 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)*. IEEE, 2010, pp. 1–4.

[20] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, "End-to-end timing analysis of cause-effect chains in automotive embedded systems," *Journal of Systems Architecture*, vol. 80, pp. 104–113, 2017.

[21] S. Beji, S. Hamadou, J. Mullins, and A. Gherbi, "Iterative integration of ttethernet network flows," *International Journal of Critical Computer-Based Systems*, vol. 9, 01 2018.

[22] S. D. McLean, E. A. Juul Hansen, P. Pop, and S. S. Craciunas, "Configuring ADAS platforms for automotive applications using meta-heuristics," *Frontiers in Robotics and AI*, p. 353, 2022.

[23] X. Zhou, F. He, L. Zhao, and E. Li, "Hybrid Scheduling of Tasks and Messages for TSN-Based Avionics Systems," *IEEE Transactions on Industrial Informatics*, 2023.

[24] M. Lauer, J. Mullins, and M. Yeddes, "Cost optimization strategy for iterative integration of multi-critical functions in ima and ttethernet architecture," in *2013 IEEE 37th Annual Computer Software and Applications Conference Workshops*. IEEE, 2013, pp. 139–144.

[25] A. Pruski, M. A. Ojewale, V. Gavrilut, P. M. Yomsi, M. S. Berger, and L. Almeida, "Implementation cost comparison of TSN traffic control mechanisms," in *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2021, pp. 01–08.

[26] P.-J. Chaine, M. Boyer, C. Pagetti, and F. Wartel, "Egress-TT Configurations for TSN Networks," in *Proceedings of the 30th International Conference on Real-Time Networks and Systems*, 2022, pp. 58–69.

[27] L. Maile, K.-S. Hielscher, and R. German, "Network calculus results for TSN: An introduction," in *2020 Information Communication Technologies Conference (ICTC)*. IEEE, 2020, pp. 131–140.