Migrating Legacy Ethernet-Based Traffic with Spatial Redundancy to TSN networks

Mateu Jover*, Manuel Barranco*, Inés Álvarez†, Julián Proenza*
* Universitat de les Illes Balears, Spain, † Mälardalen University (MDU), Sweden
mateu.jover@gmail.com, manuel.barranco@uib.es, ines.alvarez.vadillo@mdu.se, julian.proenza@uib.es

***Abstract*—Distributed Control Systems (DCSs) for emerging industrial control applications impose new communication requirements that cannot be satisfied by current Industrial Ethernet protocols. As a result, industry is pushing the *Time-Sensitive Networking* (TSN) standards as the de-facto Ethernet-based link-layer to fulfill these requirements. Adequate roadmaps are needed to support a smooth transition from Industrial-Ethernet-based legacy systems to TSN-based ones. In this context some works propose mechanisms to *migrate*, i.e. map, route and schedule, legacy traffic to TSN. However none of them considers traffic including streams with spatial redundancy requirements and, thus, they cannot be used to migrate legacy highly-reliable DCSs. The present work extends a previous toolchain to migrate, for the first time, legacy critical traffic that includes spatially redundant streams. Particularly, since redundancy is costly, this work proposes and compares two routing methods that consider one redundant stream per traffic.**

***Index Terms*—Ethernet, TSN, Spatial Redundancy, Migration**

I. Introduction

Novel industrial control applications are emerging in contexts like the Industry 4.0, smart cities or space transportation. Due to their nature, many of them rely on *Distributed Control Systems* (DCSs), whose network is fundamental to guarantee their proper operation. Besides the classical requirements imposed on these networks, e.g. hard real-time and dependable communication, new DCSs raise a series of additional communication needs like real-time flexibility, manageability, ease of reconfiguration, TCP/IP support, network convergence, adaptivity and advanced yet cost-effective fault-tolerance.

Ethernet has largely been used for data communications due to its large bandwidth, low cost, widespread know-how and TCP/IP seamless integration. Thus, a myriad of Industrial Ethernet protocols have been proposed to take advantage of these properties in control applications. However, each one of these protocols is tailored for specific domains and, thus, provides just a partial solution. Therefore, both industry and academia are intensively pushing the *Time-Sensitive Networking* (TSN) standards [1] to fulfill the mentioned requirements.

Although TSN is expected to be the de-facto link-layer standards of the whole industry, it is fundamental to ease the transition and thus interoperability between legacy systems that rely on Industrial Ethernet protocols and new TSN-based systems. A clear example are brownfield grid installations, which demand adequate transition roadmaps towards TSN [2].

This smooth transition asks for tools to migrate the traffic of legacy DCSs, i.e. legacy traffic, to TSN. Some works have been already proposed in this regard. The most advanced one so far is the toolchain proposed in [3], which allows migrating to TSN (i.e., mapping to TSN traffic classes, routing and scheduling) any type of Industrial-Ethernet-based legacy traffic. Nevertheless all the existing works focus on the real-time aspects of the traffic to be migrated, neglecting the traffic reliability requirements.

Reliability stands for the probability of providing uninterrupted service throughout a given mission time, and it is of utmost importance in many emerging control applications, e.g. autonomous driving, re-usable space shuttles, or intelligent energy distribution. To provide highly reliable communication, and thus support highly reliable DCSs, TSN proposes using spatial redundancy, which consists in replicating each critical *stream* (communication flow) and, then, route each replica in parallel through a different physical path.

The present work extends the toolchain proposed in [3], to provide a new toolchain called *Redundancy aware TSN migration toolchain* (RaTSN) that is able to migrate for, the first time, critical traffic of legacy highly-reliable DCSs to TSN. This extension consists in two methods, named *Redundant Only Routing Method* (RORM) and *Genetic Algorithm Routing Method* (GARM), to route this traffic. Since redundancy is costly, both methods duplicate only one of the traffic streams and, then, we compare them to assess their suitability.

Section II outlines basic TSN aspects relevant to this work. Section III discusses the mentioned related work and the toolchain of [3]. Section IV explains RaTSN's rationale, whereas Sections V and VI provide more details on RORM and GARM respectively. Then Sections VII and VIII discuss a series of experiments to compare RORM and GARM. Finally, Section IX concludes the paper.

II. Relevant TSN Standards

The TSN TG [1] has proposed a series of technical standards to provide Ethernet with real-time guarantees, network management capabilities and fault-tolerance mechanisms. Furthermore, TSN allows integrating traffic with different characteristics over a single network. In particular, TSN defines three traffic classes: *Scheduled* traffic (ST traffic), *Audio-Video Bridging* traffic (AVB traffic) and *Best Effort* traffic (BE traffic) [4]. ST traffic is usually time-triggered traffic with hard RT requirements, i.e. traffic transmitted periodically that must be delivered within a bounded time (prior to a deadline) to guarantee the correct operation of the system. AVB traffic is usually event-triggered traffic with soft RT requirements, i.e. traffic transmitted aperiodically that should be delivered within a bounded time to prevent the degradation of the system. Finally, BE traffic is traffic with no RT requirements.

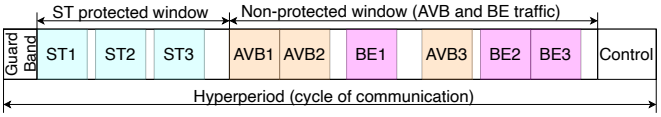


Fig. 1: Example of the structure of a TSN hyperperiod.

In order to provide real-time guarantees to ST and AVB traffic, TSN divides the communication time into cycles called *hyperperiods*. Fig. 1 shows an example of hyperperiod which is divided in four different intervals: a guard band, a ST protected window, a non-protected window and a control interval. The guard band prevents frames from different hyperperiods to overlap causing interference on the transmission of the ST streams. The ST protected window is a reserved interval where frames of ST streams are forwarded according to a Time-Division Multiple Access (TDMA) schedule that guarantees the timing requirements of the streams. The non-protected window is used to forward AVB and BE streams. As depicted in the example hyperperiod in Fig. 1, AVB and BE could interleave as they are event-triggered, so the transmission of frames depends on the instants the bridges receive them. Finally, the control interval completes the hyperperiod to send TSN control frames among the network devices, e.g. control frames for clock synchronization.

III. Related and previous work

Some recent works propose mechanisms that can be used to migrate legacy traffic to TSN. The first one, proposed in [5], is not aimed at migrating traffic to TSN, but it provides a method to monitor and extract the traffic properties of a legacy network that, then, can be used to migrate such traffic to TSN. A second work that does propose a method for migrating legacy traffic to TSN is [6]. However, the number of legacy traffic classes it considers is very limited. A third interesting one is [7], which proposes a new functionality for bridges to map legacy traffic from PROFINET [8] to TSN. However, again, it does not consider the broad range of traffic classes used in legacy systems that rely on existing Industrial Ethernet protocols.

Unlike the aforementioned works, the toolchain [3] allows migrating any type of Ethernet-based legacy traffic to TSN. Unfortunately, as the works mentioned above, it does not consider redundant streams and, thus, it cannot be used to migrate the traffic of legacy highly-reliable DCS, which do include this kind of streams, e.g. substation automation control systems. In any case, as said in Section I, [3] is so far the most advanced proposal to migrate legacy traffic to TSN; hence, it is the starting point from which the present work builds on. We will refer to the toolchain of [3] as the *original toolchain* from hereon.

Fig. 2 depicts the basic workflow of both the original toolchain and the new migrating toolchain proposed in the present work, i.e. RaTSN. The yellow boxes represent the elements of RaTSN that were not present in the original toolchain; thus, at this point, let us focus only on the white ones to understand how this original toolchain works.

The first tool of the original toolchain is called LETRA [3], which is a mapping tool that takes the legacy traffic as input and, then, classifies (maps) it into the different TSN traffic classes (ST, AVB and BE). The legacy traffic is characterized according to a given *traffic model*, i.e. according to a set of key parameters like periods, offsets, deadlines, etc. The traffic mapped by LETRA constitutes the input of two scheduling tools. First, the *ST Scheduler* [9] takes the traffic mapped as ST, which, as already explained in Section II, is the time-triggered hard real-time traffic. The ST Scheduler tries to produce a feasible routing and TDMA scheduling for that traffic, i.e. a routing and a TDMA scheduling that fulfill that traffic hard real-time requirements. Second, as regards the AVB Analyzer [10], it has two inputs, namely: (1) the routing and TDMA schedule produced by the ST scheduler, and (2) the traffic mapped as AVB or BE by LETRA. As AVB traffic has soft real-time requirements and the BE traffic has not any, the AVB Analyzer just checks if the AVB and BE traffic can be allocated using the bandwidth left by the ST traffic.

IV. RaTSN design rationale

We designed the *Redundancy aware TSN migration toolchain* (RaTSN) as an extension of the original toolchain without changing any of its modules. We consider the modules that constitute RaTSN as blackboxes. This allows replacing any of them by modules proposed by other experts, as well as to keep compatibility with their potential upgrade. Note that the AVB analyzer is depicted in grey in Fig. 2 since it does not form part of RaTSN. This is because AVB and BE traffics are not critical and, thus, none of their streams are replicated.

The elements we introduce in RaTSN are highlighted as colored boxes. The first one is the *Redundant Traffic Model*. Note that legacy systems that have traffic with spatial redundancy requirements rely on topologies with certain level of redundancy, so that each one of the replicas of a redundant stream can be transmitted through different routes in parallel. The traffic model of the original toolchain includes parameters to specify the topology, and allows it to be redundant; but it does not include any parameter to specify the spatial redundancy requirements of the traffic. *Redundant Traffic Model* is an extension to the original traffic model that allows to do so. Concretely, it allows specifying in the box *Stream red. spec.* (besides the legacy traffic) which streams should be replicated and how many replicas should each stream have.

The second element we introduce is a new module to replicate the redundant streams and then route the traffic including the stream replicas, called *Replication and Routing tool* (ReRo). The ST traffic that LETRA outputs becomes the input of ReRo together with the spatial redundancy requirements from *Stream red. spec*. ReRo takes the ST traffic and generates the necessary stream replicas. Then, it routes the original ST traffic together with the generated replicas through the specified topology. Finally, ReRo asks the ST scheduler to schedule the *Routed ST Traffic with replicas*. If the ST scheduler succeeds in scheduling the traffic through the indicated routes, the traffic is successfully migrated. Otherwise, ReRo

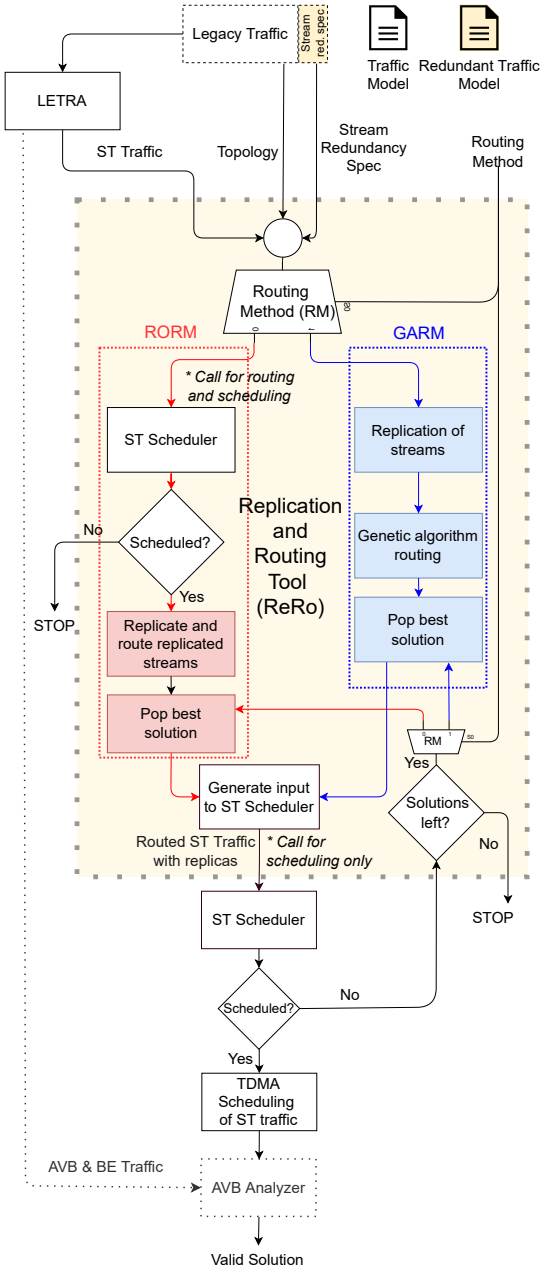


Fig. 2: Block diagram of RaTSN.

recalculates the routes and asks the ST scheduler to schedule again; this step is repeated until a schedulable solution is found or ReRo desists.

As said in Section I, we propose and compare two routing methods within RaTSN, which focus on replicating one stream once. Fig. 2 depicts the workflow of these two methods within ReRo, as well as the input *Routing Method* (RM) to specify which one to use. The first one, called *Redundant Only Routing Method* (RORM), is depicted in red in the left side. RORM's motivation is to take advantage from the routing features of the ST scheduler [9] so as to obtain a feasible routing for the non-replicated ST traffic, i.e. for the ST traffic that still does not include the stream replica to be generated. This routing is a valuable partial solution since, although it still does not include the stream replica, it is already known to be feasible. As Fig. 2 shows, an instance of the ST scheduler is placed as the first module of RORM; meaning that RORM firstly invokes the ST scheduler to obtain that partial solution. If the ST scheduler cannot find a feasible routing then RORM desists ("STOP"). Otherwise RORM replicates the redundant stream and, starting from the partial solution, proposes a complete routing solution that includes the generated replica.

Starting from a partial but already feasible solution may reduce the computation time required by RORM to find the route for the replica. However this makes the performance of the routing method dependent on the performance of the routing features of the ST scheduler. Moreover, the partial solution provided by the ST scheduler does not take into account all the traffic, since it does not consider the stream replica. Thus, the partial solution may lead to a sub-optimal complete solution, e.g. a complete solution that uses more bandwidth than would be actually needed. This may limit ReRo's ability to tackle more complex traffic, as it may exhaust the bandwidth before other routing methods, e.g. when the number of streams increases.

To overcome these potential limitations, the second routing method we propose (depicted in blue in the right side of Fig. 2) considers all the traffic from the very beginning. We refer to it as the *Genetic Algorithm Routing Method* (GARM). Note that considering all the traffic from the very beginning increases the number of routing combinations that the routing method should consider. Thus, we decided to base GARM on a genetic algorithm, since genetic algorithms are known to suit huge combinatorial problems, such as routing and scheduling ones [11]–[13].

Although RORM and GARM follow a different strategy, both of them try to maximize the reliability of the redundant traffic by applying the following criteria.

1) The route of a given stream and the route of its replica must share as few bridges as possible.
2) Each route must use as few bridges as possible.
3) Each stream should interfere with as few streams as possible.

We consider criterion 1 as the most important of the three, since it aims at maximizing the fault independence between the original stream and its replica. The objective of criterion 2 is twofold. First, it aims at reducing the number of bridges (and links) whose failure can make a route unusable. Second, it is intended to ease the scheduling by reducing the number of resources (bridges and links) to be allocated to the stream (or replica) routed through that route. Finally, criterion 3 also aims at easing the scheduling, but by reducing the amount of resources shared among different streams.

In the following Sections we further explain the details of RORM and GARM.

V. Redundant Only Routing Method

As said before, RORM first invokes the ST scheduler to route and schedule the ST streams without considering any replica. Then, RORM generates the stream replica and

routes it. The pseudocode of Algorithm 1 shows how RORM calculates the possible routes for the stream replica and selects the route it consider as the most adequate.

Algorithm 1 Replica route selection in RORM

```
1: if First time then
2:     ordered_routes ← Empty list
3:     available ← all_routes[talker, listener]
4:     available ← available − {original}
5:     max_b ← num bridges original stream
6:     b ← 0
7:     while b ≤ max_b do
8:         routes ← group_by_b(available, b, original)
9:         Score routes according to criteria 2 and 3
10:        Sort routes by score
11:        Append routes to ordered_routes
12:        Increment b
13:    end while
14:    First time ← False
15: end if
16: replica_route ← best in ordered_routes          ▷ Pop route
17: return replica_route
```

The first part (lines 2-4) obtains the set of all possible routes for the stream replica in such a way that none of these routes is the route already assigned by the scheduler to the original stream. This set is stored in *available*.

The second part of the algorithm (lines 5-13) aims at storing in *ordered_routes* the routes of *available* sorted according to criteria 1, 2 and 3. Variable *max_b* represents the maximum number of bridges that can be shared between the route of the original stream and the route of its replica, and it is initialised to the number of bridges of the original route. Variable *b* represents the number of shared bridges allowed in the current iteration of the loop. By iterating from the least overlapped routes ($b = 0$) to the most overlapped ones ($b =$ *max_b*), the algorithm orders the routes and stores them in *ordered_routes* according to criterion 1, i.e. the higher the position of a route in *ordered_routes*, the lower the number of bridges it shares with the original stream's route.

Each iteration takes the routes that share *b* bridges and further sorts them according to criteria 2 and 3. Specifically, line 8 picks up from *available* the subset of routes that share *b* bridges with the route of the original stream. This subset is stored in variable *routes*. Line 9 uses criteria 2 and 3 to score each one of these routes, and line 10 sorts them within *routes* according to their score. Finally, line 11 appends the ordered content of *routes* to *ordered_routes*.

Finally in lines 16-17 the algorithm pops the best route from *ordered_routes*, and passes it to the ST scheduler. If the ST scheduler cannot schedule the traffic using this route for the replica, the next invocation to the algorithm pops the following best route. RORM is invoked until the traffic is schedulable or *ordered_routes* is empty.

Note that line 9 scores each route by using Eq. 1.

$$C_{n,m} = w\frac{S_m - S_{n,min}}{S_{n,max} - S_{n,min}} + (1-w)\sum_{\substack{i=1 \\ i\neq n}}^{N}\frac{|e_m \cap e_i|}{|e_m \cup e_i|} \quad (1)$$

Where,

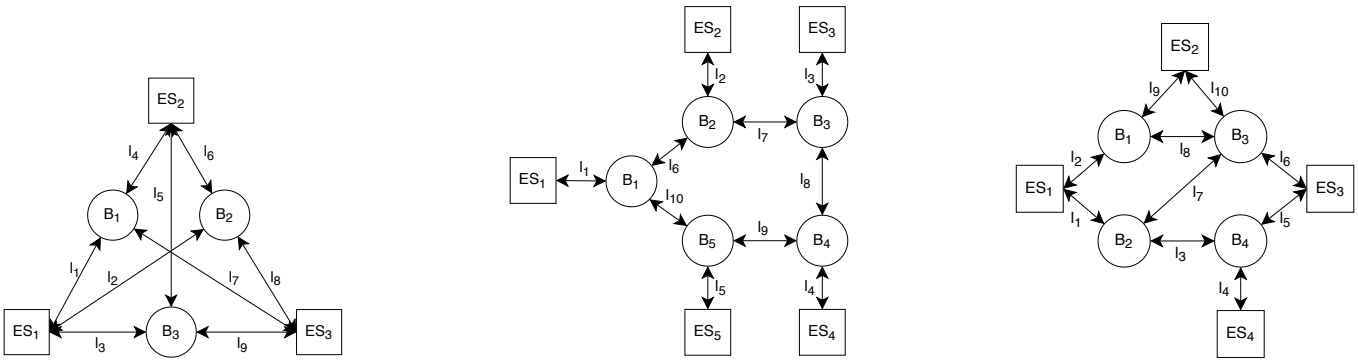
- $C_{n,m}$ is the cost of routing the replica of stream n through route m. The lower the value of $C_{n,m}$ the more desirable is to route through that route m.
- S_m is the number of bridges of route m.
- $S_{n,min}$, $S_{n,max}$ are the minimum and maximum number of bridges of all the possible routes connecting the talker and listener of stream n.
- e_m is the set of links of route m.
- e_i is the set of links assigned to stream i.
- N is the number of streams in the ST traffic.
- w is a parameter that weights the two terms, $w \in [0, 1]$.

In particular the second term of Eq. 1 represents criterion 3 and quantifies the overlapping between the route m of the replica and the routes of the streams that are not the original one, n. For this it uses the Jaccard index (intersection over union) which here quantifies the overlapping as a similarity in terms of links.

VI. Genetic Algorithm Routing Method

As explained in Section IV, GARM is based on a Genetic Algorithm (GA). GAs search over a *population* of possible solutions, called *individuals*. More precisely, GAs generate and choose, through several generations, the individuals that better maximize a given *fitness function* so as to eventually obtain a good solution. Each individual is represented by a *chromosome*. In GARM each individual is a routing proposal for the whole ST traffic, which includes the stream replica, and the chromosome is a list of the routes of this proposal.

GAs initialize the population as random as possible to better explore the search space. In GARM each individual is initialized as a list of routes, one per stream (and stream replica), such that each route is randomly selected from among all the possible routes for that stream (or replica). In each generation GAs select a subset of individuals that will be used as the parents to generate the *offspring*. GARM implements parents selection with a method called *tournament selection* [14]. After selecting the parents, GARM generates a set of descendants. The chromosome of each descendant is generated from two parents, so that the route of each stream (and the replica) of a descendant is inherited with equal probability from one of its two parents. To avoid local minimums GARM sometimes randomly mutates (changes) some routs of each descendant and, additionally, introduces some random individuals. Once the offspring is generated, GARM uses the fitness function to filter the descendants and select the individuals for the next generation. For simplicity GARM selects as many descendants as the initial population, and substitutes the whole generation for the offspring. GARM finishes the search when it reaches a certain number of generations.



(a) Star topology with three bridges. (b) Ring topology with five bridges. (c) Mesh topology with four bridges.

Fig. 3: Topologies used for the experimental evaluation of RORM and GARM.

The fitness function quantifies the desired characteristics of each individual by means of a series of constraints. In GARM the constraints are computed as routing costs that reflect the criteria specified in Section IV as follows.

Eq. 2 computes a constraint that reflects criterion 1 as a cost based on the number of shared bridges between the original stream and its replica.

$$F_1(ind) = |S_{orig} \cap S_{rep}|^{1.5} \tag{2}$$

Where,

- F_1 is the cost of a given individual (*ind*) according to criterion 1.
- S_{orig} is the set of bridges of the route assigned to the original stream.
- S_{rep} is the set of bridges of the route assigned to the stream replica.
- We use the 1.5 exponent for the number of elements in the intersection to increase the cost of overlapped routes in a parabolic manner that helps the algorithm to converge.

Eq. 3 computes a constraint to reflect criteria 2 and 3.

$$F_{2,3}(ind) = \sum_{i=1}^{N} C_{i,m} \tag{3}$$

Where,

- $F_{2,3}$ is the cost of a given individual (*ind*) according to criteria 2 and 3.
- N is the number of ST streams including the replica.
- $C_{i,m}$ is the cost of routing i through m using Eq. 1.

Recall that Eq. 1 computes the cost of a given route of the stream replica. However Eq. 3 uses Eq. 1 to compute the route cost not only of the replica but also of each one of the streams and, then, adds them all together. Thus it is necessary to reinterpret N and $C_{n,m}$ of Eq. 1. Now N is the total number of ST streams including the replica; whereas $C_{n,m}$ is the cost of routing stream n through route m.

Finally the fitness function is computed, according to Eq. 4, as a weighted addition of the two costs just specified above. The weights, i.e. w_1 and $w_{2,3}$, allow to favor one cost over the other, which allows to adjust the fitness function.

$$Cost(ind) = w_1 \cdot F_1(ind) + w_{2,3} \cdot F_{2,3}(ind) \tag{4}$$

Note that since in our case the fitness function, i.e. $Cost(ind)$, is calculated as a cost, maximizing the fitness function actually means to minimize this cost.

VII. Experimental setup

As already said, in this work we compare the two routing methods of RaTSN. To that, we execute RORM and GARM to route the traffic of a set of networks. To generate these networks we used the tool from [3] called *network generator*. Specifically, we generate a variety of networks that differ in terms of the network topology and the *traffic complexity*. The *traffic complexity* is characterized by the number of ST streams and the size of the ST protected window.

We consider these to be relevant aspects to evaluate because of several reasons. On the one hand, the network topology determines the routes connecting each pair of end-stations and, thus, the routes that the routing methods can assign to the different streams and the replica. On the other hand, we expect that the number of ST streams and the size of the ST protected window have a significant impact on the routing and the schedulability. A higher number of ST streams does not only make it more difficult to route them, but increases the complexity of the scheduling since the higher the number of streams the lower the resources available for each one of them. Moreover, the size of the ST protected window limits the amount of bandwidth available for the ST streams.

We considered three full-duplex topologies inspired by topologies commonly used in legacy systems. Fig. 3 illustrates the three topologies. We must note that a higher number of links implies more resources available for routing (and scheduling). Thus, to properly study the impact of the topology on our results, we designed those three topologies so that each one of them includes approximately the same number of links.

Fig. 3a depicts the first topology, which is a replicated star with three central bridges and three end-stations. Each end-station is attached to each bridge, resulting in three routes that do not share bridges between any pair of end-stations. We should also note that in this topology every route has the same

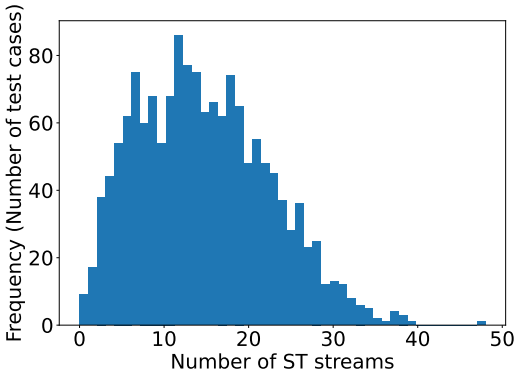


Fig. 4: Distribution of test cases by ST streams.

amount of bridges. Fig. 3b depicts the second topology, which is a ring with five bridges and five end-stations. Each bridge is connected to two other bridges, and it is directly connected to just one end-station. Note that in this topology there are only two possible routes to connect each pair of end-stations, and that these routes share at least two bridges. This topology resembles ring topologies that are used in legacy systems based on Industrial Ethernet protocols such as AeroRing[15]. Finally, Fig. 3c illustrates the third topology, which is a mesh with four bridges and four end-stations. This mesh topology offers several routes for connecting each pair of end-stations, and some of these routes share one or more bridges.

As regards the traffic complexity, the total number of streams ranges from 4 to 75; the number of ST streams ranges from 1 to 48; and the size of the ST protected window ranges from 10 to 90. These ranges can be considered as representative of many legacy industrial networks, according to [3]. Taking into account these ranges, we have generated 510 sets of traffic for each topology, resulting in a total of 1530 networks or test cases. Note that the network generator [3] randomly generates each one of the 510 traffic instances considered per topology. Specifically, for each size of the ST protected window it generates the same number of traffic instances, so that this size is uniformly distributed along the 510 instances. This allows not to prioritize any ST protected window size over others in the comparison.

Concerning the number of ST streams the network generator produces in each test case, note that to schedule a higher number of ST streams it is necessary to have a sufficiently large ST protected window. Thus, the network generator intrinsically establishes a relationship between the ranges of these two magnitudes. Specifically, the network generator produces test cases that, on average, keep a proportional relationship between the number of ST streams and the size of the window. In any case, the number of streams and ST streams per instance are randomly generated following a normal distribution. Fig. 4 shows the frequency with which a given number of ST streams appear in the test cases. This frequency, expressed as the number of test cases that use a given number of ST streams, follows the shape of a gaussian bell, which corroborates the non-biased character of the traffic in terms of the number of ST streams.

		Scheduled solutions				**Valid solutions**			
Overlapping degree		0	1	2	>2	0	1	2	>2
RORM	Star	354	0	0	0	354	0	0	0
	Ring	0	0	210	0	0	0	210	0
	Mesh	5	184	74	30	5	184	74	30
GARM	Star	382	0	0	0	382	0	0	0
	Ring	0	0	257	0	0	0	106	0
	Mesh	117	118	47	30	117	17	0	0

TABLE I: Scheduled & valid solutions and overlapping degree
!

VIII. Experimental results

In this Section we compare the two routing methods from different points of view. First we compare their *routing capability* depending on the topology. We consider that the routing capability includes two aspects, namely the number of test cases (networks) for which the routing method is able to find a *valid solution*; and the *overlapping degree* of each one of the solutions. On the one hand, we consider that a solution is valid if it fulfills the two following requirements: it is schedulable and the route of the original stream and of its replica are not exactly the same. Note that a replica adds no spatial redundancy, and thus no fault tolerance, if its route is the same as the route of its original stream. Thus the route of a given stream and of its replica must differ from each other by at least one bridge or link. On the other hand, note that the higher the difference between two routes, the more fault-independent they are. We assess the fault-independence of the routes each solution proposes for the original stream and its replica by means of the overlapping degree, which counts the number of bridges they share.

Second, we compare the percentage of valid solutions RORM and GARM provide depending on the *traffic complexity*, i.e. on the number of ST streams and the size of the ST protected window, which as already said affects the difficulty for routing and scheduling. Then, we compare both methods in terms of the bandwidth overhead of their valid solutions. Finally, we analyze the execution time of RaTSN depending on which one of the two routing methods it uses.

Note that in some analyses we compare RORM and GARM with a routing method called Base. This method simply represents the case in which no stream is replicated, i.e. the case of the original toolchain in which the traffic mapped by LETRA as ST is scheduled and routed by the ST scheduler.

A. Routing capability vs topology

We compare the routing capabilities of RORM and GARM for each topology. Table I shows the number of networks they were able to schedule and the number of those for which they were able to find a valid solution. Results are grouped in rows by topology, as well as in columns by the overlapping degree between the route of the stream and of its replica.

As regards the replicated star, all the solutions provided by RORM and GARM have an overlapping degree equal to 0. Also, all the networks that are scheduled are valid. These results indicate that both methods are able to fully exploit the fact that this topology provides three independent stars, so as to route the stream and its replica through completely

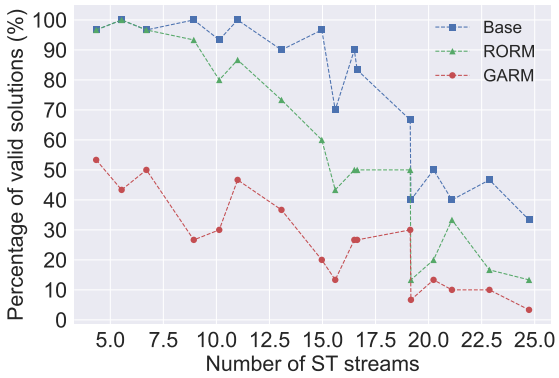


Fig. 5: Percentage of valid solutions vs number of ST streams.

independent routes. Table I also shows that, for the replicated star, GARM provides more valid solutions (382 out of 510) than RORM (354 out of 510). This indicates that when we have independent routes, as it happens in this star, GARM is superior because it provides more routing solutions that are schedulable. This superiority can be explained by the fact that RORM is burdened by the partial routing solution initially provided by the ST scheduler. This partial solution does not include the stream replica and, thus, leans RORM towards providing sub-optimal solutions in terms of schedulability. This result suggests that it is interesting to investigate how to provide schedulers that can schedule and route considering the stream replica from the very beginning. This interest has also been identified in other works, e.g. [16].

Concerning the ring, Table I shows that the overlapping degree of all solutions is two. It is not possible to provide a lower overlapping degree because, as explained in Section VII, the only two routes that the ring provides for connecting each pair of nodes share at least two bridges. In any case, the results show that the overlapping degree is never greater than two.

As regard the number of solutions, note that Table I indicates that GARM provides more schedulable routings than RORM (257 vs 210) also in the ring. This corroborates the superiority of GARM in terms of schedulability, which was explained above for the replicated star. However, the number of valid solutions of GARM is reduced down to 106. This difference between the number of scheduled and valid solutions for GARM indicates that its fitness function is not fully adequate for routing in the ring topology. More specifically, note in Eq. 4 that the first term of the function rewards lower overlapping degrees between a stream and its replica, whereas one of the aspects the second term rewards are shorter routes. Thus, in cases in which the two routes connecting two end-stations are such that one of them is much larger than the other, the fitness function may incorrectly lead GARM to select the same route for the stream and its replica. Note that the costs and the fitness functions were defined for any topology, and their parameters were set by trial and error for good results in general. Thus, for certain topologies it is needed to adjust the fitness function or define an ad-hoc one

Conversely to the ring, the mesh (Figure 9) does not limit the minimum overlapping degree; but the mesh does not guarantee non-overlapping routes as the replicated star does. For the mesh, RORM provides a slightly higher number (293) of scheduled networks than GARM (285), and, as happens in the ring, RORM ends up providing more valid solutions than GARM, i.e 293 vs 134. In any case, for the mesh, GARM tends to provide solutions with less overlapping degree than RORM and, thus, solutions that a priori attain a higher reliability. In particular, the average number of shared bridges between the routes of a stream and its replica in the valid solutions for the mesh is 1.44 and 0.12 for RORM and GARM.

B. Percentage of valid solutions vs traffic complexity

The percentage of valid solutions obtained by RORM and GARM depending on the traffic complexity are similar for all topologies. Thus, for the sake of succinctness, Fig. 5 shows the comparison for the case of the mesh topology only. We must recall that the network generator produces test cases that on average keep a proportional relationship between the number of ST streams and the size of the ST window. For clarity, the x-axis of Fig. 5 represents the traffic complexity in terms of the number of ST streams.

Fig. 5 confirms that the traffic complexity negatively impacts the percentage of valid solutions of RORM and GARM (and of the Base case). It also shows that although the percentage of valid solutions is greater in RORM, it decreases at a higher rate than in GARM. This suggests that the traffic complexity is more of a limiting factor for RORM than for GARM; thereby reinforcing the interest of adjusting the fitness function (or other aspects like the chromosome), so as to exploit the potential benefits that GARM, may provide when addressing more complex problems.

C. Bandwidth overhead

We consider the bandwidth of a solution as the average bandwidth the ST scheduler allocates to the ST traffic at the links of the selected routes. Then, we define the *bandwidth overhead* of a valid solution i, given by RORM or GARM, as an increment $\Delta BW_i = BW_i - BW_{base}$; where BW_i is the bandwidth of the valid solution; and BW_{base} is the bandwidth of the solution given by the Base method, which does not consider redundant streams. In other words, the bandwidth overhead is the average bandwidth increment with respect to the case in which no stream is replicated.

Fig. 6 depicts the bandwidth overhead distribution of RORM and GARM. All RORM valid solutions have a $\Delta BW_i > 0$, i.e. they always increase the bandwidth consumption. This was expected, as RORM keeps the routes proposed by the ST scheduler and then adds the route of the stream replica. Conversely, around half of valid solutions of GARM have a $\Delta BW_i < 0$. This means that GARM allows finding routes that consume less bandwidth than those proposed by the ST scheduler, even though the ST scheduler does not consider the stream replica, i.e. GARM allows reducing the average bandwidth consumption of the ST traffic with respect to the

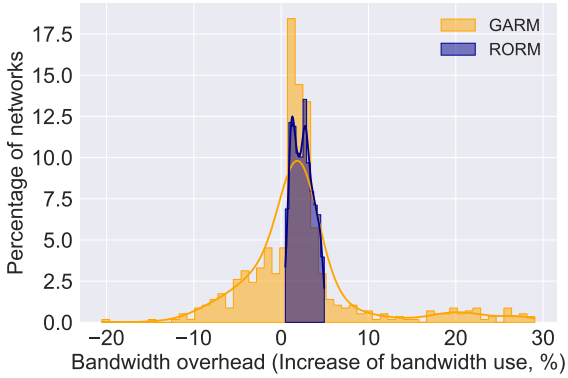


Fig. 6: Bandwidth overhead distribution.

original toolchain (and RORM). This is because GARM considers more routing combinations than the ST scheduler and, thus, GARM explores a larger search space than the original toolchain (and consequently than RORM). The downside of GARM is that it can yield a greater maximum overhead than RORM, i.e. around 29% vs 4%. In any case, the number of GARM's solutions that are worse than RORM's ones is low.

D. Execution time of RaTSN

Finally we measured the execution time of RaTSN for migrating each one of the 1530 test cases when using RORM and GARM. The execution time of each test case includes the execution of LETRA, the actions carried out by RORM (or GARM depending on the used method) within the ReRo tool, as well as each call to the ST scheduler that RORM (or GARM) carry out outside the ReRo tool (Fig. 2) until they find an schedulable solution or they desist.

Results are similar for all topologies. Although GARM is slightly better than RORM in terms of the maximum execution time, i.e. 38103 and 45064 s, and standard deviation, i.e. 5883 vs 6767 s; the average execution time of GARM doubles the one of RORM, i.e. 17808 vs 8120 s. This supports RORM as a method that can take advantage from a tool, already refined by the community, to carry out part of the process.

IX. Conclusions

Industry and academia are pushing TSN as the de-facto Ethernet-based link-layer to comprehensively fulfill the requirements of novel control applications in emerging areas such as Industry 4.0 or smart cities. In this context it becomes fundamental to ease the transition between legacy and TSN-based systems. In this regard some works have been proposed to partially migrate traffic from proprietary protocols to TSN. In particular, they focus on the real-time aspects of the traffic, while neglecting existing reliability requirements.

In this work we extend the toolchain of [3] to migrate, i.e. map, route and schedule, for the first time, critical hard-real time traffic of legacy highly-reliable DCSs to TSN. This extension basically consists in two methods, RORM and GARM, to route critical traffic that includes a redundant stream that is spatially duplicated. RORM focuses on taking advantage of the routing capabilities of a scheduler, proposed by the community and used in [3], to start from a partial routing solution that, although not including the stream replica, is known to be feasible. GARM is based on a genetic algorithm to route all the traffic, including the stream replica, from scratch. We compare these methods for three topologies. Generally speaking, RORM obtains a higher number of valid solutions than GARM and, in average, its execution time is noticeably lower. However, GARM tends to provide solutions with lower overlapping between the route of the stream replica and of its original stream, which in principle benefits reliability; and introduces less bandwidth overhead. Particularly, we observed that it may be necessary to use an ad-hoc fitness function to benefit from GARM's lower overlapping in some topologies.

Acknowledgements

Grant PID2021-124348OB-I00 funded by MCIN/AEI/ 10.13039/501100011033 / ERDF, EU. This work was supported in part by the Swedish Governmental Agency for Innovation Systems (VINNOVA) through the DESTINE project.

References

[1] Time-Sensitive Networking (TSN) Task Group. [Online]. Available: https://1.ieee802.org/tsn/

[2] IEC, "Communication networks and systems for power utility automation – part 90-13: Deterministic networking technologies," IEC, Geneva, Switzerland, Tech. Rep. 61850-90-13:2021-02(en), 2021.

[3] D. Bujosa Mateu, M. Ashjaei, A. V. Papadopoulos, J. Proenza, and T. Nolte, "LETRA: Mapping Legacy Ethernet-Based Traffic into TSN Traffic Classes," in *26th IEEE ETFA*, 2021, pp. 1–8.

[4] "IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks," *IEEE Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014)*, pp. 1–1993, 2018.

[5] M. Gutiérrez, W. Steiner, R. Dobrin, and S. Punnekkat, "Learning the parameters of periodic traffic based on network measurements," in *2015 IEEE International Workshop on Measurements and Networking (M&N)*, 2015, pp. 1–6.

[6] V. Gavriluţ and P. Pop, "Traffic-Type Assignment for TSN-Based Mixed-Criticality Cyber-Physical Systems," *ACM Trans. Cyber-Phys. Syst.*, vol. 4, no. 2, jan 2020.

[7] S. Schriegel and J. Jasperneite, "A Migration Strategy for Profinet Toward Ethernet TSN-Based Field-Level Communication: An Approach to Accelerate the Adoption of Converged IT/OT Communication," *IEEE Industrial Electronics Magazine*, vol. 15, no. 4, pp. 43–53, 2021.

[8] IEC, "Digital Data Communications for Measurement and Control – Fieldbus for Use in Industrial Control Systems (All Parts)," IEC, Tech. Rep. 61158/61784, 2014.

[9] N. Reusch, L. Zhao, S. S. Craciunas, and P. Pop, "Window-Based Schedule Synthesis for Industrial IEEE 802.1Qbv TSN Networks," in *16th IEEE WFCS*, 2020, pp. 1–4.

[10] M. Ashjaei, G. Patti, M. Behnam, T. Nolte, G. Alderisi, and L. Lo Bello, "Schedulability analysis of ethernet audio video bridging networks with scheduled traffic support," *RTSs*, vol. 53, no. 4, pp. 526–577, 2017.

[11] M. Pahlevan and R. Obermaisser, "Genetic Algorithm for Scheduling Time-Triggered Traffic in Time-Sensitive Networks," in *23rd IEEE ETFA*, vol. 1, 2018, pp. 337–344.

[12] N. G. Nayak, F. Dürr, and K. Rothermel, "Incremental Flow Scheduling and Routing in Time-Sensitive Software-Defined Networks," *IEEE Trans. on Industrial Inf.*, vol. 14, no. 5, pp. 2066–2075, 2018.

[13] S. Majidi and R. Obermaisser, "Evolutionary Algorithm for Incremental Scheduling in Systems of Systems with Real-Time Requirements," in *47th IEEE IECON*, 2021, pp. 1–6.

[14] A. Eiben and J. Smith, *Introduction to Evolutionary Computing*, 2nd ed., G. Rozenberg, Ed. Heidelberg, Germany: Springer-Verlag, 2015.

[15] A. Amari, A. Mifdaoui, F. Frances, J. Lacan, D. Rambaud, and L. Urbain, "AeroRing: Avionics Full Duplex Ethernet Ring with High Availability and QoS Management," in *ERTS 2016*, Jan. 2016.

[16] Z. Feng, Q. Deng, M. Cai, and J. Li, "Efficient reservation-based fault-tolerant scheduling for ieee 802.1qbv time-sensitive networking," *Journal of Systems Architecture*, vol. 123, p. 102381, 2022.