Schedule Reparability: Enhancing Time-Triggered Network Recovery upon Link Failures

Francisco Pozo, Guillermo Rodriguez-Navas and Hans Hansson
School of Innovation, Design and Engineering
Mälardalen University
Västerås, Sweden
Email: {francisco.pozo, guillermo.rodriguez-navas, hans.hansson}@mdh.se

***Abstract*—The time-triggered communication paradigm has been shown to satisfy temporal isolation while providing end to end delay guarantees through the synthesis of an offline schedule. However, this paradigm has severe flexibility limitations as any unpredicted change not anticipated by the schedule, such as a component failure, might result in a loss of frames. A typical solution is to use redundancy or replace and update the schedule offline anew. With the ever increase in size of networks and the need to reduce costs, supplementary solutions that enhance the reliability of such networks are also desired. In this paper, we introduce a repair algorithm capable of reacting to unpredicted link failures. The algorithm quickly modifies the schedule such that all frames are transmitted again within their timing guarantees. We found that the success of our algorithm increases significantly with the existence of empty slots spread over the schedule, an opposite approach compared to packing frames, commonly used in the literature. We propose a new ILP formulation that includes a maximization of frame and link intermissions to stretch empty slots over the schedule. Our results show that we can repair with 90% success rate within milliseconds to a valid schedule compared to a few minutes needed to re-schedule the whole network.**

I. Introduction

Time-triggered communication [1] has been successfully adopted in many application domains such as automotive [2] or aerospace [3] where proof of the deterministic behavior is required. The implementation of a static pre-computed schedule where the transmissions of all time-triggered frames are determined and assured by a global synchronization clock protocol [4] is a crucial feature to provide satisfactory proof. Many different protocols have been implemented using the time-triggered paradigm such as TTEthernet [5] and TSN [6].

A well-known drawback of the time-triggered paradigm is the lack of a mechanism to react to unpredicted changes and failures [7], the solution to which is either to re-compute and update the whole network schedule or to implement redundancy. With the increase in size and complexity of time-triggered networks, the time needed to compute their schedules is growing, thus posing serious difficulties to adaptation to changes or failures. For instance, synthesizing the schedule of industrial-size networks usually takes close to one hour [8], and for larger networks it can ascend to several hours [9]. With respect to tolerating communication link failures, solutions based on redundancy incur in extra overhead and introduce additional complexity in the schedule which may not be suitable for large and loaded networks. The current trend towards deploying larger networks, such as the Real-Time Internet-of-Things [10], or cheaper implementations in mass-production products, like autonomous vehicles [7], demands developing novel methods to further enhance the reliability and flexibility of the time-triggered networks.

In this paper, we focus on the specific problem of recovering the predictability of the network when one or more of the communication links are permanently faulty and making it impossible for the frames scheduled over those links to reach their destinations. The proposed solution is called a *repair algorithm* and is a method that localizes which portion of the schedule is affected by the link failure(s) to change (we say it *repairs*) only those parts, instead of re-calculating the whole schedule. The aim of this method is to reduce the number of changes to the minimum such that a new and operational schedule can be in place as soon as possible, reducing the downtime of the network significantly. The rationale of the repair algorithm is simple: whenever two adjacent nodes are disconnected due to a link failure, the first step of the repair algorithm is to find a new path that connects them and then re-allocate on this path only the affected frames, i.e. the ones transmitted over the faulty link, without modifying the rest of the schedule. In case a valid schedule cannot be found like this, the repair algorithm tries a second strategy and changes the transmission times of all frames allocated over the links of the new connecting path.

The evaluation of the repair algorithm shows promising results. The algorithm is able to recover from consecutive link failures in the order of milliseconds, while a whole re-scheduling can take a few minutes. Additionally, we found out that the effectiveness of the algorithm depends on certain properties of the schedule, like the separation between frames and the existence of empty slots spread regularly over time. In particular, it was observed that generating schedules with the frames packed together, a commonly-applied technique known as minimizing the *makespan* of the schedule, is in fact detrimental for our repair algorithm. Based on this, we define a new set of scheduling constraints, based on ILP, that ensure that the generated schedules are easier to repair using our repair algorithm. Such constraints are based on the insertion of frame and link *intermissions* (idle times). Our evaluation shows that the schedules obtained considerably increase the chances of the repair algorithm to be successful even when consecutive link failures occur.

The contributions of this paper are: (i) an efficient and low-cost repair algorithm to recover from link failures; (ii) the introduction of the notion of schedule reparability as a measure of how easily a certain schedule can be transformed into another valid schedule, upon link failure; (iii) an optimization constraint model for obtaining highly reparable schedules using ILP solvers, and its performance evaluation; (iv) an evaluation of the repair success rate when using a highly reparable schedule compared to using a common makespan-minimization schedule.

In Section II we introduce related work in schedule synthesis, optimization, and the inclusion of failures in the scheduling decision procedure. Section III presents the basics of time-triggered networks and Section IV proceeds with the scheduling problem ILP formulation. We propose the concept of schedule reparability in Section V together with the repair algorithm. In Section VI, we extend the ILP constraints to obtain high reparability schedules. We evaluate the performance of our scheduling approach and the repair algorithm in Section VII. We present our conclusions and future work in Section VIII.

II. Related Work

Synthesizing schedules using a combination of SMT solvers and ILP solvers in different protocols such as PROFINET, FlexRay, etc. has been applied to obtain cyclic schedules [11][12][13]. However, as the size and complexity of networks increased with the introduction of Switched-Ethernet real-time protocols, there was a need to combine solvers with incremental approaches [8][14] and a combination of heuristics and solvers for larger networks [9][15] to address the deficient scalability.

The implementation of optimization strategies to obtain demanded attributes from the schedules has proven successful applying both meta-heuristics and ILP Solvers. Serna et al. [14] employed ILP solvers to reduce the number of gates needed in Time-Sensitive Network switches. A tabu-search meta-heuristic was adopted to minimize costs in mixed-critically networks including the mapping of tasks into processors to the network scheduling problem by Tamas et al. [16]. There exist extensive literature in minimization of the schedule make-span [17][18][19], especially in mixed-critically networks. In the absence of any schedule attribute to prioritize, many authors choose to minimize the make-span to pack all time-triggered frames at the start of the schedule trying to avoid interference with frames from other criticality levels. Our work shows that this strategy is detrimental for reparability and we will present an alternative optimization model.

Different approaches have been proposed to tolerate transient and permanent link failures. Pop et al. include the re-execution of frames to cope with arbitrary transient link failures [20]. But this approach overloads the schedule with the transmission of duplicated frames, which could even prevent scheduling in highly congested networks. The same authors eliminated frame re-execution with the formulation of quasistatic schedules [21], where a set of schedules for different link failures scenarios is designed, successfully reduced the schedule overload. The limitation to a pre-defined link failure scenario reduces the applicability of this techniques, as it does not provide tolerance to any link failure, or combination of link failures, outside the selected scenario. Scheduling along with network redundancy was also proposed by Wisniewski et al. [22]. Our approach is complementary to these solutions using redundancy in networks, particularly for those in which the cost of a totally redundant solution can be prohibitive.

Other research efforts investigated how to react to unpredictable changes during run-time. Zhang et al. proposed a cloud-based approach with a server containing alternative possible schedules that can be launched after network changes [23]. In the case that no schedule meeting the requirements is available, a new one would be synthesized. Even though this approach reacts to any change, note that the whole schedule may need to be computed, which requires several seconds even for small networks. Our approach aspires to repair industrial-size networks with a response time milliseconds range. Avni et al. implement a combined offline and online technique to tolerate link failures generating a k-resistant schedule together with a simple recovery protocol [24][22], although it introduced the possibility of some frames to be dropped after recovery, which our approach avoids. Avni et al.'s work also presents scalability issues because they rely on very computationally demanding algorithms [25].

The introduction of slack in the schedule was studied in time-triggered distributed systems to recover from intermittent faults by Kandasamy et al. [26], where run-time recovery policies could be employed as a consequence of the schedule slack. Our approach differs in the possibility to also modify part of the schedule while maintaining a low repair time.

III. Preliminaries

A. Time-Triggered Networks

We define a multi-hop network as a directed graph $G = (V, L)$, where the vertices V represent a switch or an end system and the edges L represent the links that connect vertices including an associated capacity C_l measured in Bytes per second. In time-triggered networks, information can be exchanged between end systems through a frame $f \in F$, where F is the set of all frames in the network. A frame can only be initiated by an end system *sender* but can be dispatched to one or more end systems *receivers*. A direct connection is not allowed between two end systems. An end system can only be connected to switches, which can be connected to both end systems and other switches.

Frames are transferred from a sender to a receiver through a path called *data flow path* p that is a sequence of links $(v_x, v_y) \in L$ representing a connection from sender v_s to receiver v_r such as:

$$p = [(v_s, v_x), ..., (v_y, v_r)] \tag{1}$$

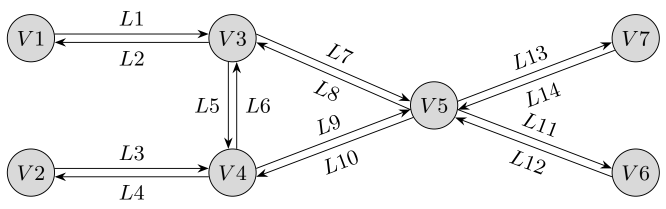


Fig. 1. Network Example

TABLE I
TRAFFIC MODEL FOR FIGURE 1

Frame ID	Sender	Receivers	Tree Path	Period	Deadline
1	V1	V6	L1-L7-L11	8	8
2	V1	V6	L1-L7-L11	8	8
3	V2	V6	L3-L9-L11	8	8
4	V2	V6-V7	L3-L9-L11-L13	8	8

There may exist multiple possible paths to connect v_s and v_r. In this paper, the frame is given the shortest path to connect both end systems, but any other path can also be given. We represent the associations of a frame that has multiple receivers as the union of all data flow paths denoted as a *tree path TP*.

To illustrate tree paths, consider the network in Figure 1 that contains four end systems $(V1, V2, V6, V7)$, three switches $(V3, V4, V5)$ and a total of fourteen directional links. In the traffic presented in Table I we can perceive four different frames with tree paths assigned by the shortest path to connect the sender with the receivers. In the case of f_4, as it contains multiple receivers, the tree path TP_4 is assigned as the union of a path from $V1$ to $V6$ and a path $V1$ to $V7$: $[L3, L9, L11] \cup [L3, L9, L11]$ that can also be annotated as $TP_4 = [(v_2, v_4), (v_4, v_5), (v_5, v_6)] \cup [(v_2, v_4), (v_4, v_5), (v_5, v_7)] = [(v_2, v_4), (v_4, v_5), (v_5, v_6), (v_5, v_7)]$.

We can define a frame as a tuple:

$$f = \langle T_f, D_f, L_f, TP_f \rangle \tag{2}$$

where T_f is the period, D_f is the deadline, L_f is the size in Bytes and TP_f is the tree path.

B. Scheduling Problem

A time-triggered network requires a cyclic schedule that specifies the transmission times of all frames over each of the links within the *hyper-period*, such that all reach their destination on time. The schedule hyper-period T_F is obtained as the minimum common multiple of all the frame periods: $T_F = LCM(T_f)$ for all $f \in F$.

Every node keeps the schedule of their outcoming and incoming links to identify when frames should be received and when to send them to other nodes. Frames that have a period smaller than the hyper-period require being transmitted several times within the schedule. Each such transmission is called a *frame instance*. The number of instances within an hyper-period can be calculated as $N_f = \frac{T_F}{T_f}$.

A common practice to reduce the scheduling complexity is to discretize the problem to the integer domain with the inclusion of schedule granularity of a *time slot* [27]. In the rest of this paper, we will consider the time slot as one nanosecond to attempt to capture as much details as possible while avoiding having to deal with continuous time.

We can then define the scheduling problem as finding an assignment for the transmissions times of all frames such as satisfies all the network and traffic constraints as they are presented in Section IV:

$$\Phi_f : [1, N_f] \times L \rightarrow \mathbb{N}^+ \cup \{*\} \tag{3}$$

where the transmission time called *Offset* $\Phi_f(i, l) = t$ designates that the i-th frame instance of frame f will start its transmission over link l at time $t \in \mathbb{N}$. If $\Phi_f(i, l) = *$, the i-th frame instance is not transmitted over link l. For brevity, to indicate that a frame is transmitted over link l at time t, we will say that the frame is given an offset t over link l. This assignment is what will be called the schedule. Note that the solution is not unique, since there may be many different schedules satisfying the constraints.

C. Fault Model

Our fault model only includes faults that have permanent or long impact on the system schedulability, as these are the faults that can be tolerated with network rescheduling. Specifically, we consider only *permanent link failures* and transient link failures that are long enough to cause the loss of a large number of frames. Note that thanks to the global time notion provided by the synchronization protocol and the schedule stored in the network nodes, switches and end systems can detect if any frame they should have been received is missing or if it is not correctly synchronized. Whenever multiple frames that are expected to be received in a link are missed, the node recognizes such link as suffering a permanent failure, or a long enough transient failure, that requires triggering the repair algorithm.

IV. SCHEDULING CONSTRAINTS

In this section, we formally define the ILP formulation constraints for Φ:

a) Frame Range Constraints: Every first frame instance can only be transmitted in a range defined by the interval $(0, D_f]$ that assures that the frame satisfies its deadline requirement:

$$\begin{aligned} &\forall f \in F, \forall (v_x, v_y) \in TP_f : \\ &0 < \Phi_f(1, (v_x, v_y)) \leq D_f \end{aligned} \tag{4}$$

Remember that since frames are strictly periodic, it is not necessary to define a range for subsequent frame instances. Instead, we only need to ensure that such frame instances are strictly separated by the adequate distance, which is defined as a multiple of the frame period, starting from the first frame instance:

$$\begin{aligned} &\forall f \in F, \forall (v_x, v_y) \in TP_f, \forall i = 2, ..., N_f : \\ &\Phi_f(i, (v_x, v_y)) = \Phi_f(1, (v_x, v_y)) + (T_f \times (i - 1)) \end{aligned} \tag{5}$$

b) Avoid Collision Constraints: Two frames cannot be transmitted over the same link simultaneously. We model this constraint considering every possible pair of frames sharing the same link on their tree paths. However, we also need to model which of the pair of frames will be transmitted before and which will be transmitted after. The ILP formulation does not allow us to naturally describe such interaction. We are bound to create two new binary variables a, b where only a or b can be active, hence $a+b=1$ as to allow only one frame at a time to be transmitted on the same link:

$$\begin{aligned}
&\forall f_i, f_j \in F, i \neq j, \forall (v_x, v_y) \in TP_{f_i}, TP_{f_j},\\
&\forall d = 1, ..., N_{f_i}, \forall e = 1, ..., N_{f_j}:\\
&a = 1 \rightarrow \Phi_{f_i}(d, (v_x, v_y)) + \frac{N_{f_i}}{C_{(v_x, v_y)}} \leq \Phi_{f_j}(e, (v_x, v_y)),\\
&b = 1 \rightarrow \Phi_{f_j}(e, (v_x, v_y)) + \frac{N_{f_j}}{C_{(v_x, v_y)}} \leq \Phi_{f_i}(d, (v_x, v_y)) \qquad (6)
\end{aligned}$$

where if $a=1$ indicates that the offset of f_i over link (v_x, v_y) happens before the offset of f_j in the same link. On the other hand, if $b=1$ then the offset of frame f_j happens before. The frames offsets that need to be carried over the link are calculated as $\frac{N_f}{C_{(v_x, v_y)}}$.

To further reduce the formulation complexity, we will only append the avoid collision constraint if both frame instances can collide according to their available range, given by $(T_f \times (i-1), T_f \times (i-1) + D_f], i \in N_f$. If they cannot collide, we do not need to model such impossible collisions, reducing the number of constraints significantly.

c) Path Dependent Constraints: For a frame to successfully reach each of its receivers, it needs to follow the data flow paths in an orderly manner. Therefore, a switch cannot relay a frame if it has not received and processed it before. We also utilize this constraint to specify the time that the switch needs for processing the frame, which is measured from the time the frame has been received until the time it is ready to be transmitted.

$$\begin{aligned}
&\forall f \in F, \forall p \in TP_f, \forall (v_x, v_y), (v_y, v_z) \in p:\\
&\Phi_f(1, [v_y, v_z]) - \Phi_f(1, [v_x, v_y]) \geq \mathit{hopdelay} \qquad (7)
\end{aligned}$$

where *hopdelay* specifies the switch processing time. Notice that it is only needed to add the constraint for the first frame instance because succeeding frame instances will inherit it as a result of the strict periodicity.

d) End-to-End Delay Constraints: Time-triggered frames often require to be received after a short delay measured from the instant in which the sender started the transmission. We design such function restricting the time difference between the start and the end of each receiver:

$$\begin{aligned}
&\forall f \in F, \forall p \in TP_f, (v_s, v_x), (v_y, v_r) \in p:\\
&\Phi_f(1, (v_y, v_r) - \Phi_f(1, (v_s, v_x) \leq \mathit{endtoend} \qquad (8)
\end{aligned}$$

where *endtoend* identifies the maximum end-to-end delay allowed between the transmission on the first link (v_s, v_x) and the transmission on the last link in the paths of any receivers (v_y, v_r). By the same principle as the path constraints, we only need to enforce it for the first frame instance.

V. Reparability

The principal limitation of the time-triggered paradigm is the inability to adapt to unpredicted changes, which causes the uploaded schedule not to guarantee the transmission of all frames within their timing requirements. If the network does not contain a backup schedule predicting that specific change, the schedule needs to be synthesized again from scratch, which is computationally and timely expensive. In this section, we present a method that generates a valid schedule with just a limited number of changes, avoiding full re-scheduling.

A. Reparability Concept

Our repair algorithm is based on the notion that a schedule S can be transformed into another schedule S' by means of modifications. Before describing the algorithm, we need to differentiate between *schedulability* and *reparability* in the presence of faulty links.

Given a network N, a set of frames F and a set of faulty links $L_F \subset L$, we say that N is Schedulable upon L_F failures if there exist a schedule that satisfies all constraints without using any link of L_F. This property gives us some intuition about how the repair will happen, because it indicates which faulty links a schedule can tolerate. If, let us say, S uses a certain link l that S' does not use, then, upon failure of l, S' would be a valid repair of S.

To express such a relation between schedules, we define the following notion. Given a network N, configured with schedule S, a set of faulty links $L_F \subset L$ and a repair algorithm A, we say that network N is reparable by A upon L_F failures, if either S does not use any link of L_F or A can transform S into a new schedule S' that does not transmit over L_F. Note that a network that is not schedulable, cannot be repaired; but the reciprocal does not hold. However, in the trivial case in which A implies a full re-scheduling of the network, then schedulability and reparability are equivalent (i.e. if a system is schedulable upon L_F failures, then it is reparable).

For all other cases, and to evaluate the quality of a repair algorithm, we should measure the ability of the algorithm to find a valid schedule when a valid schedule exists, and relate it to the number of faulty links. To calculate that, we define the notion of Schedule Reparability $SR_n(A, N, S) = [0, 1] \in \mathbb{R}$, with $n \in \mathbb{N}$ being the number of faulty links. For instance, a value $SR_n(A, N, S) = 0.5$ implies that in any scenario having exactly n faulty links and in which N is schedulable, algorithm A is able to repair S (on average) only one out of every two times. In Section V-C, it will be shown that SR_n does not depend exclusively on the algorithm A, but also on the characteristics of the initial schedule S.

$SR_n(A, N, S)$ is calculated by dividing all the occurrences where A repairs the schedule by all the occurrences where the network is schedulable, with $|L_F| = n$. A reparability of 1 implies that given the initial schedule S and repair algorithm A, it is possible to repair acany arbitrary set of n link failures, provided that the network is schedulable upon such failures. Note that if for all L_F of size n there is no schedule, then $SR_n(A, N, S) = 1$ irrespective of A and S. We state this for completeness, though it is a property that will not be evaluated.

In the following subsections, we will present the implementation of a computationally efficient repair algorithm and will discuss how to find a so-called highly reparable initial schedule.

B. Low-Cost Repair Algorithm

We introduce a repair algorithm A that seeks to repair S with low complexity, which in this paper involves reducing the number of frame offset modifications. The procedure starts with a link failure. The general idea is to reconnect both end nodes of the faulty link with another available path. If such path exists, we then start the algorithm, which is composed of two phases (from less complex to more complex) for the schedule repair problem:

1) The first phase obtains all the frame offsets that were transmitted in the faulty link. We create new frame offsets for every link that connects both end nodes of the faulty link. Then we try to allocate such new frame offsets within their timing requirements.
2) If no solution was found in the first phase, we try again in the second phase, but this time we allow for all the frame offsets in the new path to be re-allocated too within the link they are transmitted. This phase is more computationally expensive but has a higher probability to find a valid frame offset allocation. If both phases fail, we conclude that the repair has failed.

The pseudo-code of the repair algorithm can be examined in Listing 1. Given faulty link, in line 2 we try to find if there exists an alternative path that connects both nodes. If no such path exists, we cannot repair the schedule. If it exists, we iterate over all frames to check which among them include a transmission over the faulty link, and save them in a list of affected frames. In the same loop, we also check if any frame has a transmission already in the links of the newly obtained path, to save them to the list of immutable offsets for that link. Once all frames have been inspected, the function *calculate_available_ranges* in line 13 checks when the predecessor and successor transmission times in their frame path are allocated to set a permissible range of the frame offsets without breaking any constraint. Once all this information is extracted, we can start phase 1 by executing the same scheduling algorithm presented in Section IV; recognizing that the affected frames need to be in the obtained ranges, and the fixed offsets cannot be modified from the initial schedule.

If phase 1 fails, we start phase 2 also calculating the available ranges of the fixed offsets as done before with the affected frames. We continue by allowing the transmission times of the fixed offsets to be re-adjusted by adding them to the list of affected frames. Finally, we execute the scheduling algorithm again. The purpose of phase 2 is to allow the links in the new obtained path to be fully re-scheduled. If any of both phases returns a schedule, we can update the current schedule with the values obtained. However, if both phases fail, the repair algorithm is unable to find a new schedule. In the case that the network has more than one failure at the same time, we perform the repair algorithm sequentially, with one failure at a time.

Notice that the presented repair algorithm seeks simplicity and rapid execution so as to support the proof of the reparability concept. Other algorithms can be implemented in which the search for more efficient alternative paths may increase the chances for repair. We leave that improvement for future work.

Listing 1. repair Algorithm Pseudo-code

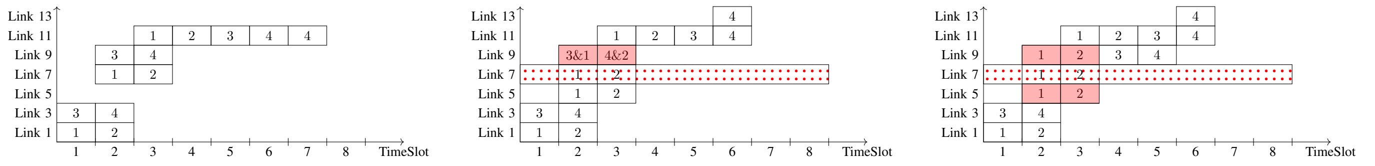
```
1  function repair_Algorithm (link $(v_x, v_y)$)
2    newpath ← find_shortest_path($v_x, v_y$)
3    if not newpath then
4      return false
5    for $f \in F, (v_w, v_z) \in TP_f$ do
6      if $(v_x, v_y) == (v_w, v_z)$ then
7        for $(v_t, v_v) \in$ newpath do
8          affected_frames ← $\Phi_f(i, (v_t, v_v))$
9      for $(v_t, v_v) \in$ newpath do
10       if $(v_w, v_z) == (v_w, v_z)$ then
11         fixed_offsets ← $\Phi_f(i, (v_w, v_z))$
12   %% Phase 1
13   ranges ← calculate_available_ranges
          (affected_frames)
14   if schedule (ranges, affected_frames,
          fixed_offsets) failed then
15     %% Phase 2
16     ranges ← calculate_available_ranges
          (fixed_offsets)
17     affected_frames ← fixed_offsets
18     if schedule (ranges, affected_frames) failed then
19       return false
20   update schedule
```

Let us illustrate how the algorithm operates with an example. If we recollect the network in Figure 1 with the traffic in Table I, we can execute a scheduler that tries to minimize the make-span to obtain the schedule shown in Figure 2a. Given a failure of link $L7$, the repair algorithm starts by finding that there is an alternative path, $L5$-$L9$, that connects nodes $V3$ and $V5$. Furthermore, it observes that f_1 and f_2 are affected and can be allocated in the ranges $[2, 2]$ and $[3, 3]$ respectively. However, when we perform phase 1 we cannot find a schedule as, even if we can designate their frame offsets in $L5$, we still would need to transmit at the same time frames f_1 and f_3, and frames f_2 and f_4, breaking the avoid collision constraint, as illustrated in Figure 2b.

When performing phase 2, we are also unable to find a valid schedule as observed in 2c. Even though frames f_3 and f_4 were able to be reallocated due to the $L5$ and $L9$ re-scheduling, frames f_1 and f_2 still violate the path dependent constraint when trying to transmit both frames at the same time (as shown in red). The fundamental issue of the initial schedule is that, as frames are packed as much as possible due to the make-span minimization, in the event that we need to extend a link on the frame path to a larger one, we typically fail since slots requited are already allocated.

C. Initial Schedule

As manifested in the previous example, the repair algorithm is not the only factor for reparability; the characteristics of



(a) Obtained schedule with a minimization solver

(b) Obtained schedule with a minimization solver after link 7 failure in phase 1

(c) Obtained schedule with a minimization solver after link 7 failure in phase 2

Fig. 2. Example of impossibility to repair an initial poor reparability schedule

the initial schedule S are also relevant. The example gives us some insight: to be reparable, a schedule requires shifting to other schedules in which transmissions do not occur on certain (faulty) links.

We observed that it is crucial for affected frame offsets reallocated to links in a new path to find an extensive range of time slots at any schedule position. To do so, the transmission of the frame offsets over its original paths have to be as distant as possible to each other. This *Frame Distance* is a critical feature: if the new path is much longer, it will demand fitting all the new frame offsets in-between without adjusting any other frame offsets in the path.

Let us clarify this notion with the same network used before. If we maximize as much as possible the distances between frames we produce a schedule such as the one in Figure 3a, where we utilize the whole schedule hyper-period. Despite the existence of more slack between frames, we still cannot find a schedule when applying the repair algorithm in phase 1, as shown in Figure 3b. Frame f_2 still has no time left to be transmitted on $L9$ due to the interference of frames already allocated on the same link: f_3 and f_5. However, when applying phase 2 of the repair algorithm, we can reallocate the aforementioned frames and obtain a valid schedule as shown in Figure 3c.

Even though we were able to repair the schedule, we would like to avoid applying phase 2 as it is more computationally expensive than phase 1. We saw in Figure 3b that because all frames offsets were allocated consecutively over the same link, we needed to reallocate them and broaden for new frame offsets with limited transmission range. As an approach to avoid performing phase 2 habitually, we also propose to maximize the distances between frame offsets in the same link, which we call *Link Porosity*. If the links have good porosity, new frame offsets regularly will find some available time slots, reducing the time to repair a schedule, particularly for larger networks as it will be shown in the evaluation.

Figure 4a shows a schedule for the same example maximizing both frames distances and link porosity. We can appreciate that frames in the same link are separated when possible, but bear in mind that frame distances have higher priority than link porosity, since non-existent links porosity can be solved by phase 2 in most cases. In Figure 4b we can see how the repair algorithm can solve the schedule in phase 1, only needing to reallocate four offsets instead of six offsets when applying phase 2.

VI. Schedule Optimization

To find a schedule that maximizes frames distances and link porosity we need to introduce optimization to our ILP scheduling model. However, maximizing every frame distance and link porosity is costly as, even for small networks, there exist a considerable amount of new variables to optimize, which negatively impacts the scheduler scalability. We simplify the problem by modelling frame distances with an individual *frame intermission* for every frame. Link porosity is also changed by an individual *link intermission* for every link. Using intermissions reduces the number of variables to optimize to the number of frames plus the number of links, and also simplifies the implementation on the current scheduling model. Frame intermissions can be added to the path dependent constraint, while link intermissions can be included to the avoid collision constraint.

a) Optimization Function: The optimization function maximizes the cost as the summation of all the frames and links intermissions:

$$\text{maximize} \quad \sum_{f=1}^{|F|}(wf * I_f) + \sum_{l=1}^{|L|}(wl * I_l) \tag{9}$$

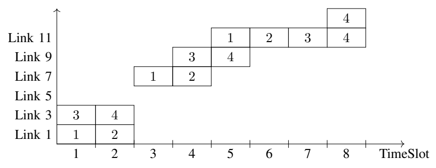
where wf is an user-defined weight influencing all the frame intermissions and wl all the link intermissions. Different values of such weights will impact the preference to maximize frames or links intermissions.

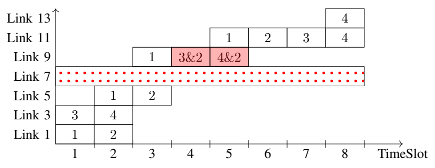
b) Path Dependent Constraint: We incorporate frame intermission as an additional value added, different for every frame, to the switch processing time to the original path dependent formulation:

$$\begin{aligned} &\forall f \in F, \forall p \in TP_f, \forall (v_x, v_y), (v_y, v_z) \in p : \\ &\Phi_f(1, [v_y, v_z]) - \Phi_f(1, [v_x, v_y]) \geq hopdelay + I_f \end{aligned} \tag{10}$$

where I_f designates the frame intermissions of the frame for which the constraint is being added.

c) Avoid Collision Constraint: We model link intermission for every link similarly to the frame intermission adding


(a) Obtained schedule maximizing frame distance


(b) Obtained schedule maximizing frame distance after link 7 failure in phase 1

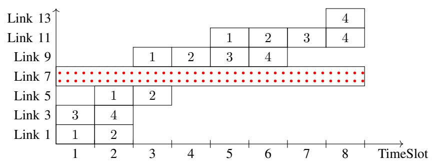
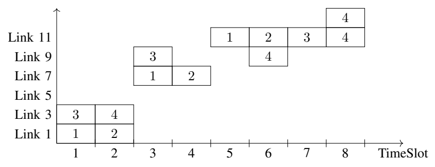

(c) Obtained schedule maximizing frame distance after link 7 failure in phase 2

Fig. 3. Example of application of repair algorithm when frame distances are maximized


(a) Obtained schedule maximizing both frame distance and link porosity

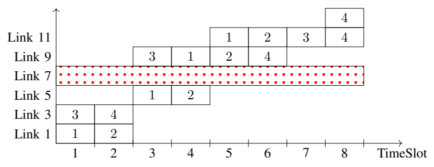

(b) Obtained schedule maximizing both frame distance and link porosity after link 7 failure in phase 1

Fig. 4. Example of application of repair algorithm when frame distances and link porosity are maximized

a different variable for every link to the frame size:

$$\forall f_i, f_j \in F, i \neq j, \forall (v_x, v_y) \in TP_{f_i}, TP_{f_j},$$
$$\forall d = 1, ..., N_{f_i}, \forall e = 1, ..., N_{f_j} :$$
$$a = 1 \rightarrow \Phi_{f_i}(d, (v_x, v_y)) + \frac{N_{f_i}}{C_{(v_x, v_y)}} + I_{(v_x, v_y)} \leq \Phi_{f_j}(e, (v_x, v_y)),$$
$$b = 1 \rightarrow \Phi_{f_j}(e, (v_x, v_y)) + \frac{N_{f_j}}{C_{(v_x, v_y)}} + I_{(v_x, v_y)} \leq \Phi_{f_i}(d, (v_x, v_y)) \tag{11}$$

where $I_{(v_x, v_y)}$ designates the link intermissions of the link for which the collision is being avoided.

VII. Evaluation

We have implemented a scheduler prototype that implements all the model and optimization constraints presented in this paper. We selected the ILP Solver Gurobi v.7.5.2 with its Python API, but any ILP Solver could be employed. The experiments were run on a MacBook Pro with macOS High Sierra, 2.9 GHz CPU Intel Core i7 and 16 GB of RAM.

A. Description

To evaluate the performance of our scheduling and repair algorithm we created two synthetic networks topologies that are schedulable despite any single faulty link. The first network, as illustrated in Figure 5, is a *small network* consisting of 3 switches, 6 end systems and 28 links where the initial shortest path is 2. The second network in Figure 6 is a *larger network*, to study how longer paths may impact both scheduling and reparability performance. It consists of 8 switches, 8 end systems and 54 links with the initial shortest path of 5. Links in the network are configured with two different capacities: *50MB/s* when connecting an end system with a switch and *100MB/s* when connecting two switches. We assume switches need 100 *ns* from receiving a frame until it is ready for transmission.

The traffic is classified into three types depending on the number of receivers:

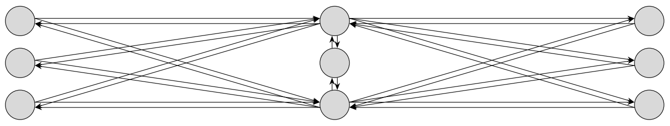


Fig. 5. Small Network Topology

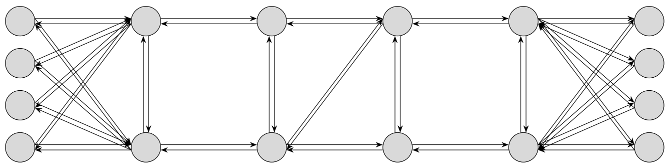


Fig. 6. Larger Network Topology

- Single: one sender and one receiver are selected randomly.
- Multi: one sender is selected randomly, and an arbitrary number of receivers are selected randomly.
- Broadcast: one sender is selected randomly, and the remaining end systems are set as receivers.

To analyze how a different number of frame offsets affects our evaluation while maintaining the number of frames, we selected two different frame distributions. Low Distribution (LD) where we have 70%, 20% and 10% of single, multi and broadcast frames respectively, and High Distribution (HD) where we have 10%, 40% and 50% of single, multi and broadcast frames respectively.

Every frame size is set to the maximum allowed by the Ethernet protocol, while periods are (10, 20, 40) *ms* at same percentages, obtaining an hyper-period of 40 *ms*. As the time slot size is set to 1 *ns*, every link will contain 4×10^7 time slots. The frame deadline is always equal to the period. We decided to set the end to end delay to a 1/4 of the frame period to limit the frame transmission expanding to the whole frame period to give our approach disadvantages. We found that, in our experiments, using a frame intermission weight $wf = 5$ and a link intermission weight $wl = 0.2$ yielded a good

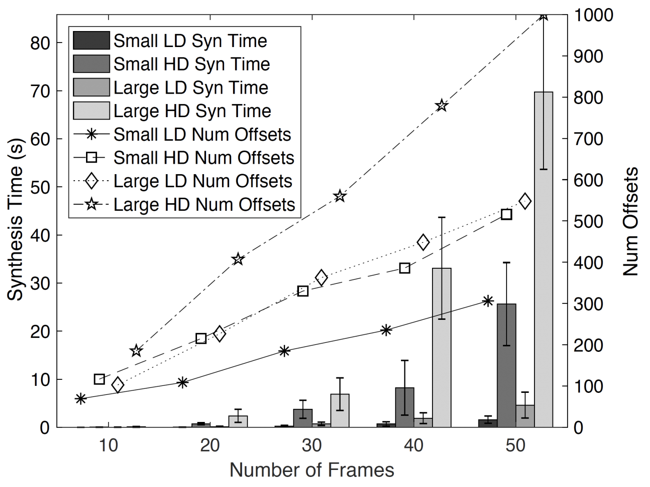


Fig. 7. Synthesis time to obtain a schedule for the both networks with different traffic distributions and different number of frames

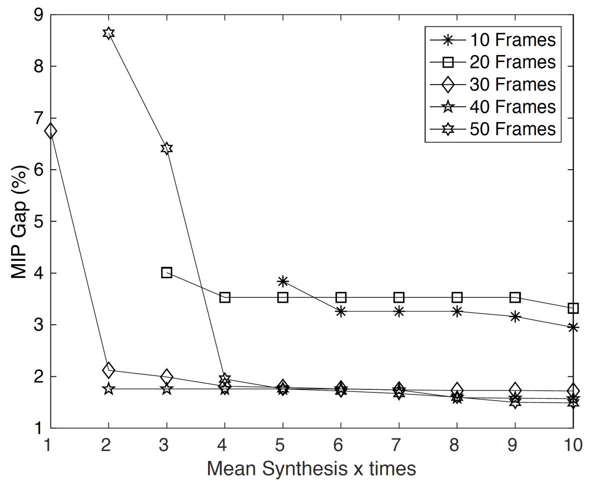


Fig. 8. Obtained MIP Gap for the small network and different number of frames with high distribution over time

prioritization of frame intermissions while still not neglecting link intermissions.

B. Scheduling Results

We first seek to evaluate the synthesis time needed for ILP Solver to obtain any valid schedule for both synthetic networks, without implementing any optimizations. For every network, we generate the traffic randomly 10 times, and each traffic was executed 10 times also, obtaining a total of 100 time measurements.

In Figure 7 we can observe the results for the small network and the larger network. As the larger network has longer paths, it also increases the number of frame offsets, which at the same time increases the synthesis time. However, the distribution has a much higher impact, where HD also increases the synthesis time as the tree path contains more offsets. Note that this substantial increase can be explained as the schedule becomes more constrained with less available time slots and therefore is harder to find for the ILP solver. It is also apparent that the scalability with the HD distribution starts to be problematic if we increase the number of frames. The scalability problem can be solved using an incremental algorithm which we do not show here due to space limitations [8].

To evaluate the performance adding the frame and link intermissions together with the optimization function, we allow the ILP Solver to be running 10 times more than the mean synthesis time obtained for each execution. We prefer to synthesize the schedule with higher possible reparability because this process is performed offline, and then we can allocate a larger amount of time to obtaining the schedule.

In Figure 8 and 9 we can see the obtained MIP Gap from the solver over the executing time for both network with different number of frames highly distributed. The MIP gap is the relative difference in lower and upper objective bound collected by the solver. A lower MIP Gap indicates a better maximization of the intermissions and being closer

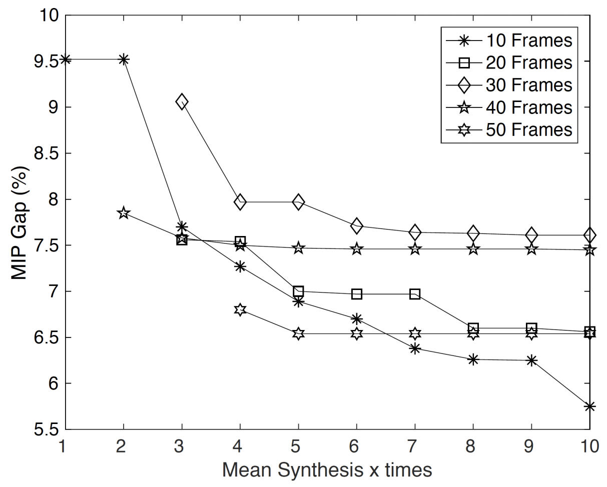


Fig. 9. Obtained MIP Gap for the larger network and different number of frames with high distribution over time

to the optimal values. We can observe that the solver only requires between two and four times the synthesis time for the small network and four to six times for the larger network to stabilize. The MIP for the small network is lower as there are fewer links in the network and it is easier for the solver to optimize. The reason why there is no MIP Gap at the start in some networks can be explained as the inherit randomness of the ILP solver may not find any valid solution in some occasions when the allowed time is too small.

C. Reparability Results

We evaluate now the reparability obtained applying the repair algorithm described in section V-B. We compare our initial schedule maximizing frames distances and links porosity against conventional schedules minimizing the make-span. We choose to evaluate both networks only with an HD traffic and 50 frames, which is the most adverse scenario for our method.

TABLE II
SCHEDULABILITY FOR THE SMALL AND LARGER NETWORKS WITH DIFFERENT NUMBER OF LINK FAILURES

Num Link Failures	Small Network	Larger Network
1	1.0	1.0
2	0.968	0.986
3	0.904	0.957

Note that higher number of frames and higher distribution are more challenging for repair.

We first show in Table II the schedulability for the small network and larger network, respectively. We can notice that, as indicated before, the schedulability of both networks is 1 when only one link failure occurs. However, it starts to decrease as the number of link failures increase as all possible paths connecting end systems start to become unusable. We did not get results for more than three link failures as there exist more than 100.000 possible link failure combinations.

To compare the reparability, we allow 10 times the mean synthesis time for both maximizing the frame and link intermissions, and for minimizing the make-span. We also include optimizations to the repair algorithm to get high reparability repaired schedules too. To minimize the make-span, we only have to exchange the optimization function to:

$$\text{minimize} \quad \sum_{f=1}^{|F|} \sum_{l=1}^{|L|} \Phi_f(i,l) \tag{12}$$

We can observe in Figure 10 the reparability for all combination of one, two and three link failures where our approach is superior, obtaining perfect reparability for one failure and approximately 90% for more failures. An unexpected result is that minimizing the make-span still yields a surprisingly high reparability. It can be explained for the small network because it is the scenario in which minimizing has more chances to succeed, for two main reasons. The first cause is that many links do not transport any frames due to the selection of the shortest paths, and therefore their failures do not affect the schedule. The second cause is related to the path length. Even when minimizing the make-span, the phase 2 of the repair algorithm still succeeds because the frame offset of the path' last link available range is not constrained by the subsequent link in the path. There is a high probability for many frames not to be too constrained as all frames in the initial schedule have a path length of only two. However, for more than one failure, the reparability starts to decrease rapidly because path lengths also increase. This is also much slower than our approach, as many repairs are done by phase 2, with an average repair time close to one second, compared to tens of *ms*.

In Figure 11, with the presence of longer paths, our approach performs better with a reparability above 95%, while minimizing the make-span deteriorates notably for more than one link failure. For the considered experiments, our approach seems to have a more positive impact on larger networks, which is encouraging.

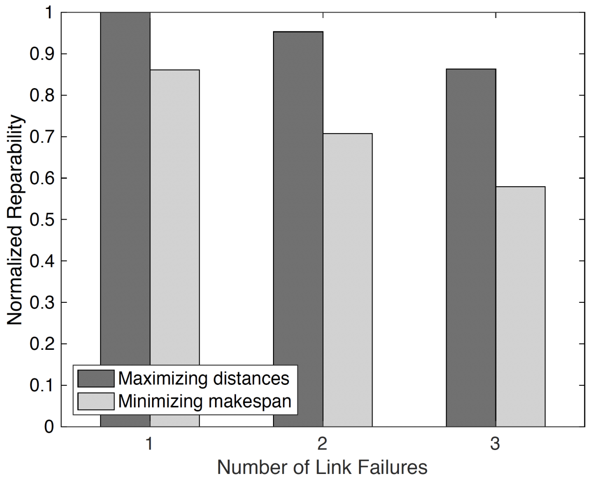


Fig. 10. Comparison of the reparability for the small HD network with 50 frames using a high reparability schedule against minimizing make-span

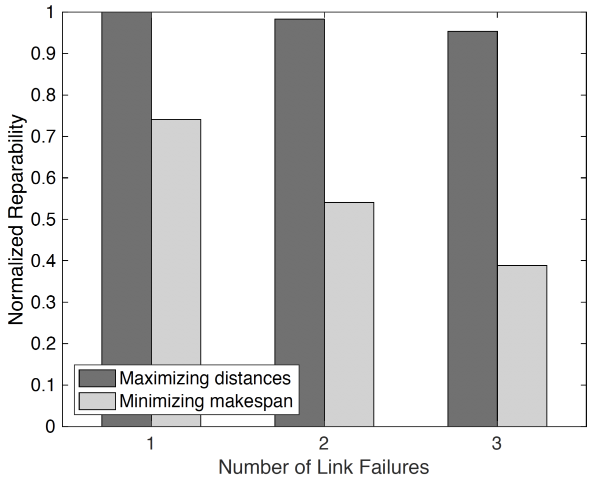


Fig. 11. Comparison of the reparability for the larger HD network with 50 frames using a high reparability schedule against minimizing make-span

VIII. CONCLUSIONS

Time-triggered networks offer high predictability with low jitter assurance provided by the pre-computation of an offline schedule that all nodes in the network follow. However, it comes at the cost of low flexibility as any unpredicted change demands re-scheduling, which is computationally expensive. We proposed the concept of schedule reparability as a desirable characteristic to recover from an unpredicted change adjusting only a small schedule section. We demonstrate that maximizing the distances between frames and adding link porosity helps to recover from link failures within an average time of tens *ms*, using a low-complexity repair algorithm. We reached a probability of success superior to 90% for the networks evaluated.

Our on-going work direction is to schedule larger networks while maintaining high reparability. We consider that employing an incremental approach or a combination of heuristics and ILP Solvers might produce the best results concerning scalability. Despite the fact that we foresee that the optimization on the schedules might deteriorate, particularly link porosity is difficult to maintain, a suitable trade-off between scalability and reparability should be further studied.

Having achieved schedule repair delays in the tens of *ms* range, the possibility to create a protocol to repair link failures during run-time without losing many frame transmissions has been opened. We would like to develop a protocol by which nodes, without knowledge of the network, can cooperate to recover to a valid schedule after one or multiple link failures have been detected.

Acknowledgment

The research leading to these results continues the funding received from the People Programme (Marie Curie Actions) of the European Union's Seventh Framework Programme FP7/2007-2013/ under REA grant agreement n°607727.

References

[1] H. Kopetz and G. Bauer, "The Time-Triggered architecture," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 112–126, 2003.

[2] L. L. Bello, "The Case for Ethernet in Automotive Communications," *ACM SIGBED Review*, vol. 8, no. 4, pp. 7–15, 2011.

[3] K. Bisson and T. Troshynski, "Switched Ethernet Testing for Avionics Applications," *IEEE Aerospace and Electronic Systems Magazine*, vol. 19, no. 5, pp. 31–35, 2004.

[4] W. Steiner and B. Dutertre, "SMT-based Formal Verification of a TTEthernet Synchronization Function," in *International Workshop on Formal Methods for Industrial Critical Systems*. Springer, 2010, Conference Proceedings, pp. 148–163.

[5] W. Steiner, "TTEthernet Specification," *TTTech Computertechnik AG, Nov*, vol. 39, p. 40, 2008.

[6] "Intstitute of Electrical and Electronics Engineers, Inc. 802.1Qbv - Enhancements for Scheduled Traffic." [Online]. Available: http://www.ieee802.org/1/pages/802.1bv.html

[7] C. Katrakazas, M. Quddus, W.-H. Chen, and L. Deka, "Real-Time Motion Planning Methods for Autonomous on-road Driving: State-of-the-art and Future Research Directions," *Transportation Research Part C: Emerging Technologies*, vol. 60, pp. 416–442, 2015.

[8] W. Steiner, "An Evaluation of SMT-based Schedule Synthesis for Time-Triggered Multi-hop Networks," in *Real-Time Systems Symposium (RTSS), 2010 IEEE 31st*. IEEE, Conference Proceedings, pp. 375–384.

[9] F. Pozo, G. Rodriguez-Navas, W. Steiner, and H. Hansson, "Period-Aware Segmented Synthesis of Schedules for Multi-Hop Time-Triggered Networks," in *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2016 IEEE 22nd International Conference on*. IEEE, Conference Proceedings, pp. 170–175.

[10] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog Computing and its role in the Internet of Things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, Conference Proceedings, pp. 13–16.

[11] Z. Hanzálek, P. Burget, and P. Sucha, "Profinet IO IRT Message Scheduling with Temporal Constraints," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 3, pp. 369–380, 2010.

[12] H. Zeng, W. Zheng, M. Di Natale, A. Ghosal, P. Giusto, and A. Sangiovanni-Vincentelli, "Scheduling the Flexray Bus using Optimization Techniques," in *Proceedings of the 46th Annual Design Automation Conference*. ACM, 2009, Conference Proceedings, pp. 874–877.

[13] J. Huang, J. O. Blech, A. Raabe, C. Buckl, and A. Knoll, "Static Scheduling of a Time-Triggered Network-on-Chip based on SMT Solving," in *Design, Automation and Test in Europe Conference and Exhibition (DATE)*. IEEE, 2012, Conference Proceedings, pp. 509–514.

[14] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, "Scheduling Real-Time Communication in IEEE 802.1 Qbv Time Sensitive Networks," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*. ACM, 2016, Conference Proceedings, pp. 183–192.

[15] F. Pozo, W. Steiner, G. Rodriguez-Navas, and H. Hansson, "A Decomposition Approach for SMT-based Schedule Synthesis for Time-Triggered Networks," in *20th IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2016, Conference Proceedings, pp. 1–8.

[16] D. Tamas–Selicean and P. Pop, "Design Optimization of Mixed-Criticality Real-Time Applications on Cost-constrained Partitioned Architectures," in *32nd IEEE Conference on Real-Time Systems Symposium (RTSS)*. IEEE, 2011, Conference Proceedings, pp. 24–33.

[17] T. Carle and D. Potop-Butucaru, "Predicate-aware, Makespan-Preserving Software Pipelining of Scheduling Tables," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 11, no. 1, p. 12, 2014.

[18] L. Wisniewski, M. Schumacher, J. Jasperneite, and C. Diedrich, "Increasing Flexibility of Time Triggered Ethernet based Systems by Optimal Greedy Scheduling Approach," in *20th IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2015, Conference Proceedings, pp. 1–6.

[19] C. Schöler, R. Krenz-Bååth, A. Murshed, and R. Obermaisser, "Optimal SAT-based Scheduler for Time-Triggered Networks-on-a-Chip," in *10th IEEE International Symposium on Industrial Embedded Systems (SIES)*. IEEE, 2015, Conference Proceedings, pp. 1–6.

[20] P. Pop, K. H. Poulsen, V. Izosimov, and P. Eles, "Scheduling and Voltage Scaling for Energy/Reliability Trade-offs in Fault-Tolerant Time-Triggered Embedded Systems," in *Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis*. ACM, 2007, Conference Proceedings, pp. 233–238.

[21] V. Izosimov, P. Pop, P. Eles, and Z. Peng, "Scheduling of Fault-Tolerant Embedded Systems with Soft and Hard Timing Constraints," in *Proceedings of the conference on Design, automation and test in Europe*. ACM, 2007, Conference Proceedings, pp. 915–920.

[22] L. Wisniewski, V. Wendt, J. Jasperneite, and C. Diedrich, "Scheduling of Profinet IRT Communication in Redundant Network Topologies," in *IEEE World Conference on Factory Communication Systems (WFCS)*. IEEE, 2016, Conference Proceedings, pp. 1–4.

[23] L. Zhang, D. Roy, P. Mundhenk, and S. Chakraborty, "Schedule Management Framework for Cloud-Based Future Automotive Software Systems," in *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2016 IEEE 22nd International Conference on*. IEEE, Conference Proceedings, pp. 12–21.

[24] G. Avni, S. Guha, and G. Rodriguez-Navas, "Synthesizing Time-Triggered Schedules for Switched Networks with Faulty Links," in *International Conference on Embedded Software (EMSOFT)*. IEEE, 2015, Conference Proceedings, pp. 1–10.

[25] G. Avni, S. Goel, T. A. Henzinger, and G. Rodriguez-Navas, "Computing Scores of Forwarding Schemes in Switched Networks with Probabilistic Faults," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2017, pp. 169–187.

[26] N. Kandasamy, J. P. Hayes, and B. T. Murray, "Transparent Recovery from Intermittent Faults in Time-Triggered Distributed Systems," *IEEE Transactions on Computers*, vol. 52, no. 2, pp. 113–125, 2003.

[27] A. K. Mok and W. Wang, "Window-Constrained Real-Time Periodic Task Scheduling," in *22nd Proceedings on Real-Time Systems Symposium (RTSS)*. IEEE, 2001, pp. 15–24.