

Self-Healing Protocol: Repairing Schedules Online after Link Failures in Time-Triggered Networks

Abstract—The time-triggered paradigm is not adaptive, a static schedule determines the time-triggered communication and, then, any unpredicted change, like a link failure, might result in the loss of frames. Using spatial redundancy or recomputing a new schedule for replacement achieves fault tolerance only in moderate-size networks. With the increase in size and complexity of cyber-physical systems, more scalable and cost-efficient mechanisms are needed in order to complement conventional solutions. We propose a distributed Self-Healing Protocol that instead of recomputing the whole schedule, repairs the existent schedule at runtime. The basis of our protocol is the collaboration of nodes in the network to individually adjust their local schedules for rerouting the frames affected by link failures. Our protocol exhibits a high success rate compared to full rescheduling, as well as remarkable scalability; it repairs the schedule in milliseconds, whereas rescheduling requires minutes.

Index Terms—Real-Time, Time-Triggered Network, Scheduling, Fault-Tolerance.

I. INTRODUCTION

THE time-triggered (TT) paradigm [1] is frequently applied in networks demanding low latency, high determinism, and high reliability. This behavior is accomplished with the synthesis of a precomputed static schedule that contains the transmission times of all the synchronous (time-triggered) frames over all the links in the network. The obtained schedule is then distributed to all the network nodes, which can follow it consistently thanks to a shared notion of time, guaranteed by a synchronization clock protocol [2]. Communication protocols such as FlexRay [3], TTEthernet [4] and TSN [5] implement this paradigm.

One major disadvantage of the time-triggered concept is its insufficient ability to adapt to changes, particularly to link failures. This problem is exacerbated when time-triggered communication is used in large-scale systems because the number of links increases significantly [6], [7]. There have been some proposals to enhance time-triggered schedules with tolerance to link failures. One family of solutions are based on rescheduling the whole network [8]. Rescheduling will always provide a new schedule, if it exists, that meets all the new requirements. However, it introduces a high response time, not only because of the time required to (re)synthesize the new schedule, but also because of the time needed to distribute it to all the nodes. On the other side, alternative solutions try to minimize response time, by precomputing schedules that tolerate potential failures. Precomputing has an almost instant response time, but it is still static in nature: if a failure that was not anticipated occurs, a new schedule cannot be provided. These solutions have both been successfully implemented in small networks, but they present significant impracticalities for larger networks. Rescheduling a large network incurs an excessive response time, in the order of minutes or even hours

[9]. Precomputing becomes very complex computationally for large scale systems, and the number of possible changes is too high to be able to save the precomputed schedules into memory; moreover, predicting the potential failures is also challenging. In this article, we seek to implement a middle ground solution between response time and success rate. I.e., given a failure in any link of the network, our aim is to obtain a new valid (and high-quality) schedule very quickly, such that end-to-end communication can be reestablished as soon as possible.

For that purpose, we revise the notion of schedule reparability, recently introduced in [10], and present a runtime Self-Healing Protocol (SHP) that is able to, upon link failure, detect which portion of the network schedule needs to be modified and orchestrates a distributed algorithm that ensures that all the relevant nodes perform the required updates in a consistent manner. More specifically, the SHP tries, when a link has a failure, to reallocate and adjust all frame instances transmitted over such a link to a new path that still connects the *receiver* and *sender* nodes. The protocol is separated into two phases: 1) discovering the new path that connects the sender node to the receiver while preparing information needed for the second phase and, 2) using the information obtained in the previous phase, modify the local schedules by allocating the frames instances transmitted in the faulty link into the newly found path, while satisfying all the frame and network constraints. Our SHP shows equivalent success rate compared to using a repair algorithm with full system knowledge. It also exhibits great scalability due to the localized nature of the schedule modifications, with very low repair times for large networks. In most of the evaluated cases, the SHP requires less than a hundred milliseconds to patch the schedule and in the order of hundreds of milliseconds to optimize the schedule to maintain a high success rate also for consecutive link failures.

Even though the SHP aims to solve some of the limitations for large networks of the previously mentioned approaches, it is conceived as an enhancement and not a replacement. Due to its best effort and localized nature when repairing from a failure, it does not reach the reliability guarantees of static redundancy or the success rate of full rescheduling. However, SHP provides a fast and online response to failures in network segments where static redundancy might not viable, at reduced cost. Moreover, it can also be applied on top of static redundancy to recover the same redundancy levels after multiple failures and prevent fast redundancy attrition.

Section II introduces relevant background on time-triggered networks and scheduling. We present the fault model and the Self-Healing Protocol rationale in Section III. In Section IV we explain the Notification and Preparation phase and in Section V, the Schedule Update phase. We evaluate the repair success

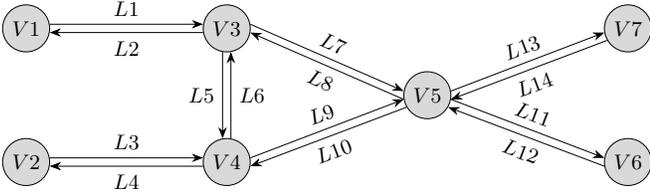


Fig. 1. Network Example

and the performance in Section VI. In section VII we compare our work with different state-of-the-art approaches to recover from link failures. We conclude in Section VIII and give some insight for future work.

II. PROBLEM STATEMENT

A. System model

We describe a multi-hop network as a directed graph $G = (V, L)$, where the vertices V represent nodes and the edges L represent links that connect nodes, each link designated with a capacity in Bytes per second C_l . We categorize nodes in *end systems* (V_e), which can send and receive information over links through frames f , and *switches* (V_s) that relay frames obtained from an end system or a switch to another end system or switch; ($V = V_e \cup V_s$) and ($V_e \cap V_s = \emptyset$). Consequently, we can connect switches to any other node, but end systems can only be connected to switches.

Every frame in the network from the set of all frames $f \in F$ starts at an end system sender that initiates the transmission to one or multiple end system receivers, following a sequence of links $(v_x, v_y) \in L$ called *data flow path* p :

$$p = [(v_s, v_x), \dots, (v_y, v_r)] : v_s, v_r \in V_e, v_x, v_y \in V_s \quad (1)$$

where v_s is the sender and v_r is the receiver. In the case that the frame has multiple receivers, we can describe the multiple paths as the union of all paths creating a *tree path* TP . Lastly, we can specify a frame as a tuple $f = \langle T_f, D_f, L_f, TP_f \rangle$ where T_f is the period, D_f is the deadline, L_f is the size in Bytes and TP_f is the previously mentioned tree path.

Let us illustrate a time-triggered network example with its topology and its traffic. In Figure 1, we can examine the network topology consisting of four end systems (v_1, v_2, v_6, v_7), three switches (v_3, v_4, v_5) interconnected by fourteen links (l_1, \dots, l_{14}). Note that in the figures we use uppercase to improve readability. In Table I we also introduce an example of traffic with four frames. For the frame with multiple receivers, f_4 , we assign the tree path TP_4 as the union of the paths from v_1 to v_6 and from v_1 to v_7 , therefore $TP_4 = [(v_2, v_4), (v_4, v_5), (v_5, v_6)] \cup [(v_2, v_4), (v_4, v_5), (v_5, v_7)] = [(v_2, v_4), (v_4, v_5), (v_5, v_6), (v_5, v_7)]$.

B. Time-Triggered Scheduling

The schedule dictates the transmission instants of all frames as they traverse the links in their respective tree paths. A correct schedule guarantees that all frames reach their receivers by their deadline, satisfying their individual end-to-end delay

TABLE I
TRAFFIC MODEL FOR FIGURE 1

Frame	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

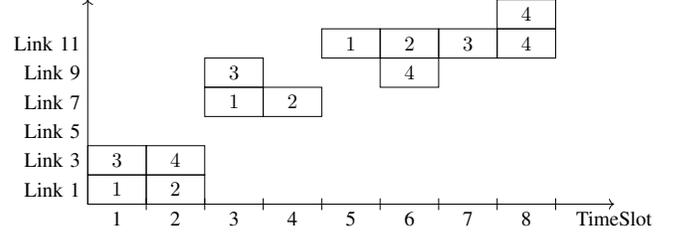


Fig. 2. Time-Triggered Schedule, highly reparable according to [10]

constraints. Then, the schedule synthesis problem is defined as finding a correct assignment over this function:

$$\Phi_f : [1, N_f] \times L \rightarrow \mathbb{N}^+ \cup \{*\} \quad (2)$$

where the *Offset* $\Phi_f(i, l) = t$ defines the frame f transmission time t (relative to the start of the hyperperiod) of the i -th frame instance over the link l if and only if $l \in TP_f$. In the case $l \notin TP_f$, we assign the offset as $\Phi_f(i, l) = *$. Every node stores the schedule for all its incoming and outgoing links, to identify when it should receive and/or transmit a frame. Because of the traffic periodicity, the schedule size can be limited to the minimum common multiple of all the frame periods, or *hyper-period* $T_F = LCM(T_f), \forall f \in F$. If a frame has a small period compared to T_F and needs to be transmitted multiple times within the hyper-period, we define each transmission as a *frame instance*. The number of instances is calculated as $N_f = \frac{T_F}{T_f}$.

For interested readers, the specifications for the network and traffic constraints can be found in [11]. Higher reparability schedules are achieved by maximizing the distances between frame transmissions [10]. For this maximization to work properly, they removed the constraint that upper bounded the time a frame can stay in a switch. However, the end to end delay constraint, which upper bounds the time for the frame to reach its destination, does indirectly limit the time that frames can effectively stay in a switch. This increases the buffer memory needed for the switches, but in the cases studied in Section VI, we observe that it does not surpass 30KB in the worse case. We show an example of a valid schedule of the previously presented network and traffic typically obtained by an ILP Solver in Figure 2.

III. OUR RATIONALE FOR SELF-HEALING

Our approach for self-healing considers permanent link failures and transient link failures lasting enough to cause a considerable loss of frames. We also assume the possibility of simultaneous link failures where, e.g., a cut in a physical link causes both link directions to be faulty. Nodes can detect a link failure with the aid of their local schedules and the

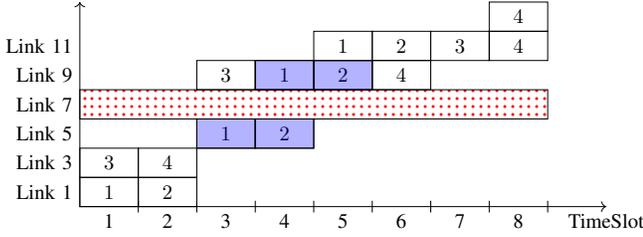


Fig. 3. Time-Triggered Schedule obtained after applying the Self-Healing Protocol to repair the failure of link 7

global notion of time shared by all the nodes in the network. When a link has a failure and stops transferring frames, the receiving node will notice that frames stated in its schedule to be received are missing. After a specified number of lost frames is detected, the node will trigger the SHP to try to repair the schedule.

Let us illustrate the rationale of the SHP with an example on the previously introduced network, assuming that l_7 suffers a permanent link failure. As seen in Figure 2, f_1 and f_2 will not be received by node v_5 when l_7 fails, and therefore the frames cannot be relayed to their end system destinations. Node v_5 detects that frames are not being received and activates the protocol. All nodes in the network collaborate in the first phase to find an alternative path, $[l_5, l_9]$. In the second phase, a group of nodes, called the SHP group, has to find transmission times for both frames in the new path while satisfying all the constraints. For example, f_1 needs to be transmitted in the range $1 < t < 5$, which we call the available range between frame instances. This second phase is performed individually, every node in the SHP group is allowed to change its local schedule to accommodate the new frames. Because of the use of a high reparability schedule, i.e. with maximized frame distances [10], there exist many available slots within the schedule for allocating the new frame instances, resulting in the schedule shown in Figure 3.

Even though the protocol operates in a distributed manner and nodes do not require to have full knowledge of the network, every node should keep three parameters for each link it is directly connected to. This includes (1) the current *link status* to remember which links have been registered as faulty, (2) the *link identifier*, which is unique in the network and used to refer to a specific link, and (3) the *link speed*, needed when repairing the schedule in order to calculate the transmission delays when sending frames on the new path.

Due to the large amount of data exchanged among nodes, and to increase the determinism of the SHP, in the synthesis of the initial schedule, empty slots are allocated in all links for the transmission of SHP frames. We define this bandwidth reservation (BR) with a period and time included in the schedule. E. g., for a given period of $1ms$ and a given time of $1\mu s$, the nodes will be able to send and receive protocol frames for one μs every ms , which corresponds to only 0.1% link utilization. We will use this BR period and time in all our evaluations, without losing generality.

Table II shows the frames of the SHP protocol for the Notification and Preparation phase and for the Update Schedule

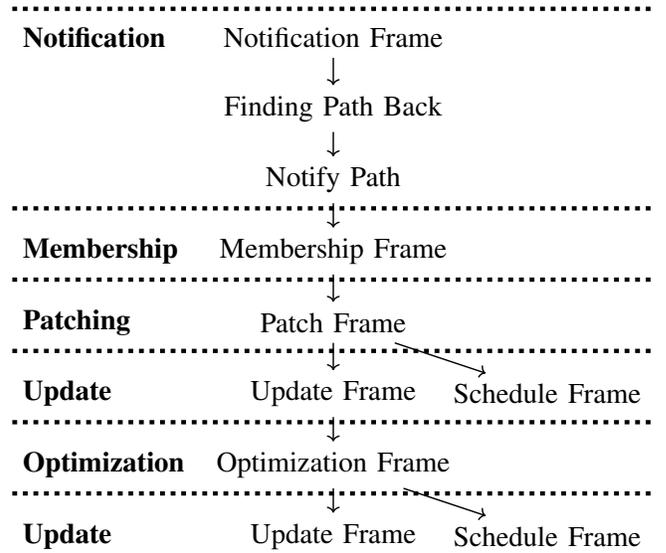


Fig. 4. Frames and Phases flowchart for the Self-Healing Protocol

phase. Figure 4 displays the diagram flow of the frames in the different protocol phases. The purpose of these frames during the SHP process will be explained in more detail in the following sections.

IV. SHP NOTIFICATION AND PREPARATION

The objectives of the first phase of SHP are (1) to notify the node sending frames through the link and (2) to find a new path that reconnects the two nodes. We will explain here the protocol designed for achieving these objectives.

A. Notification

Let $v_r \in V$ be a node (the *receiver*) receiving frames from a link $l \in L$ according to a schedule. Let $v_s \in V$ be the node (the *sender*) sending frames through link $l = (v_s, v_r) \in L$. Node v_r will consider l as faulty if there is frame omission over link l ; i.e. if a message is not received when scheduled¹.

Upon detection of a faulty link, node v_r starts the SHP by broadcasting the frame NOF, with the aim to notify node v_s that a new path needs to be found. The purpose of NOF is not only to notify v_s of the failure, but it is also used to search for a path $[v_r, \dots, v_s]$ needed for future communications. The construction of this path works as follows. Any node that receives a NOF via link $l' \in L$ will broadcast NOF, with link l' appended to the path that was contained in the received NOF. If a node receives NOF a second time, it will not broadcast the frame again. When v_s receives a NOF frame, it will start the process to find a new path $[v_s, \dots, v_r]$. To avoid a flooding of NOF over the whole network, only a maximum number of hops is allowed. Note that if a path $[v_r, \dots, v_s]$ does not exist, v_s will never receive any NOF, making the notification and, therefore, the self-repair impossible.

¹It is possible to use a counter of omissions, to avoid activation due to a transient fault, but since it does not change the protocol, we will here assume that the protocol is activated on the first detected omission.

TABLE II
SELF-HEALING PROTOCOL FRAMES

Protocol Frame	Abbr.	Starting at	Ending at	Others	Content	Purpose
Notification Frame	NOF	Receiver node	Sender node	Add path & broadcast	Faulty link ID + path the frame is following	Notify of the link failure to the sender node
Finding Path Back	PAF	Sender node	Receiver node	Add path & broadcast	Path receiver to sender + Path the frame is following	Sending node broadcasts to find a new path to allocate not transmitted frames
Notify Path	NP	Receiver node	Sender node	Relays frame	New found path	The found path is sent to the sender
Membership Frame	MF	Sender node	Membership nodes	Relays frame	Frame offsets available ranges	The sender (leader) notifies which nodes are part of the group patching the schedule
Patch Frame	PF	Membership nodes	Sender node	Relays frame	Patching result	Notify if the patching was successful or not
Schedule Frame	SF	Membership nodes	Neighbors		New schedule	Notify the neighbor to which new schedule should change
Update Frame	UF	Sender node	Membership nodes	Relays frame	Time to update to the new schedule	Send a common time when to update to the new obtained schedule
Optimization Frame	OF	Membership nodes	Sender node	Relays frame	Patching result	Notify if the patching was successful or not

Let us consider the network topology in Figure 1 with the schedule in Figure 2, and let us assume that l_7 has a failure. Node $v_r = v_5$ will detect the link failure when f_1 and f_2 no longer are received. To notify $v_s = v_3$, it will start to broadcast NOF through l_8, l_{10}, l_{11} , and l_{13} . Notice that v_3 will receive the notification immediately through l_8 and initialize the path $[v_3, \dots, v_5] \leftarrow [l_8]$. However, the other nodes do not stop broadcasting, e.g., node v_4 will receive NOF through l_{10} and broadcast NOF through l_4 and l_6 adding the information that it was received from path $[l_{10}]$. Another interesting observation is that v_3 will receive another NOF with path $[l_9, l_6]$. During the broadcast, several paths without cycles will be found, but the sender v_s will only consider the first path received, which is a good approximation of the shortest path. The possible selection of other paths, based on different criteria, is left as future work.

When v_s receives the notification of the faulty link (the first NOF), it starts broadcasting PAF (Finding Path Back) with the aim to find a path back to v_r . The nodes handle PAF in the same way they do with NOF, although now the objective is to construct the path to reach v_r from v_s . Additionally, PAF contains information about the path $[v_r, \dots, v_s]$, such that when v_r receives PAF, it learns the path to send information to v_s . From that moment on, the protocol is not based on broadcast anymore, and all communication happens through the loop path $[v_r, \dots, v_s, \dots, v_r]$. Node v_r sends frame NP (Notify Path) to communicate the path $[v_s, \dots, v_r]$ to v_s .

Continuing with our example, v_3 will send PAF through l_2 and l_5 containing the path $[v_5, \dots, v_3] = [l_8]$ with the objective to find the path $[v_3, \dots, v_5]$. The shortest path that SHP will detect, goes through v_4 , which will append l_5 to the path $[v_3, \dots, v_5] \leftarrow [l_5]$ and broadcast it through l_4 and l_9 . Node v_5 will receive the first PAF through l_9 and obtain the path $[v_3, \dots, v_5] \leftarrow [l_5, l_9]$. Node v_5 will also receive the information that it can communicate to v_3 through path $[l_8]$ and send an NP with the information $[v_3, \dots, v_5] = [l_5, l_9]$ which will be the new path to repair the schedule. Although not visible in this example, with other topologies, v_r might receive multiple possible paths. However, as indicated before,

SHP only considers the first obtained path.

If v_r never receives PAF, either because there is no path $[v_r, \dots, v_s]$ or because there is no path $[v_s, \dots, v_r]$, then SHP timeouts and the repair is not possible. In such a case, without some other fault tolerance mechanism, the frames scheduled through the faulty link will never reach v_r nor their final destination.

B. Membership

We coin the term *SHP group* to identify the set of nodes that will collaborate to repair the schedule. All the nodes present in the new path $[v_s, \dots, v_r]$, which was obtained in the notification phase, are members of the SHP group, because they own the schedules that need to be modified.

The SHP group requires a leader, the node v_s , to coordinate the changes, notify the nodes that will belong to the group and send the needed frames information to patch the schedules. When v_s obtains the path $[v_s, \dots, v_r]$ via the NP frame, it has already the knowledge that it is the leader of the group, and the paths to communicate with the nodes in the SHP group. For the affected frames, i.e. the frames that were transmitted on the faulty link, the leader also has the information of what we call the Available Transmission Ranges $ATR_f(i, l) = [t_{in}, t_{out}]$. The *ATR*s contains the range of possible offsets that each frame instance is allowed to use without violating the frame and network constraints. In simpler words, a range of all possible offsets in which the protocol can accommodate each of the affected frames. The leader already possesses the information of all needed $ATR_f(i, l_f)$, l_f indicating the faulty link, for all the frames transmitted through the faulty link with $t_{in} = \Phi_f(i, l_p)$, l_p as the link where f was received in v_s and $t_{out} = \Phi_f(i, l_f) + d_f(i, l)$, with d indicating the delay to transmit f through l . Finally, the leader equitably divides each $ATR_f(i, l_f)$ by the number of links in the path and sends it with a MF (Membership Frame) to all SHP group nodes through the path $[v_s, \dots, v_r]$.

In our previous example, $v_s = v_3$ is the SHP group leader, whereas v_4 and v_5 as the other members of the group. We can obtain from the schedule contained in v_3 that $ATR_{f_1}(1, l_7) =$

[2, 4] and $ATR_{f_2}(1, l_7) = [3, 5]$, which need to be divided by the number of links in the new path $[v_3, \dots, v_5] = [l_5, l_9]$. As the available slots for both ATR s are only three, we choose to divide in $ATR_{f_1}(1, l_5) = [2, 3]$, $ATR_{f_1}(1, l_9) = [4, 4]$, $ATR_{f_2}(1, l_5) = [3, 4]$ and $ATR_{f_2}(1, l_9) = [5, 5]$. Once v_3 calculates all the needed ATR s, it sends an MF with the needed information to each node in the group, notifying that they are part of a SHP group and indicating which new frames instances they must allocate. For example, v_4 will receive ATR s containing l_9 , while v_5 will receive an empty MF as it does not have to modify any outgoing link. The leader will also need to modify its schedule, but for obvious reasons, it does not need to send an MF to itself.

V. SHP SCHEDULE UPDATE

In the second phase, Schedule Update, every SHP group node tries to find a valid allocation for the received ATR s inside their local schedules. If all the nodes are successful, the leader can decide a point in time for all the SHP group nodes to change to their new obtained schedules. However, allocating the frames offset can be time-consuming, especially if we want to keep the high reparability of the initial schedule for subsequent link failures. The SHP cannot afford to take a long time to find a solution as the affected frames are not being transmitted while the self-healing is performed. For this reason, we divide the second phase into two steps: Patching and Optimization. The patching step aims to find a solution to transmit all frames in the network again, as fast as possible. In this step, we do not take into account the schedule reparability and can use a straightforward, fast heuristic to allocate the ATR s. Once the patching has been successful, the SHP can spend additional time to find a better schedule with higher reparability, using a more complex and hence time-consuming optimization algorithm.

A. Patching

Once an SHP group node receives a MF, it can start the patching process immediately. A pseudo-code of the patching algorithm we implemented is shown in Listing 1. The algorithm is simple; for every frame ATR , we try to allocate the frame at the start of its ATR (t_{in}). In the case it collides with another frame from the original schedule, we try again after the conflicted frame. We repeat this process until we find an available allocation, in which case we allocate the frame and move to the next ATR . If we do not have any possible allocation, we failed to patch the schedule. This can happen, e.g., if the utilization exceeds 100% or, as shown in the evaluation section, if the utilization is close to 100%. We leave for future work choosing another possible path obtained in the first protocol phase to overcome this problem. When checking if a frame collides, we need to consider all the frame instances, as they must be separated precisely by the frame period. If all frame ATR s find a possible allocation, the SHP group node has obtained the valid schedule containing the new frames.

Listing 1. Patching Algorithm Pseudo-code

```

1 function Patch_Algorithm (ATRs)
2   for  $ATR_f(i, l) = [t_{in}, t_{out}]$  in ATRs do

```

```

3   while collision( $S$ ,  $\Phi_f(i, l) = t_{in}$ ) or
4      $t_{in} + d_f(i, l) \leq t_{out}$  do
5     %% Calculate next possible allocation
6      $f_c = \text{get\_frame\_collides}(S, \Phi_f(i, l) = t_{in})$ 
7      $t_{in} = \Phi_{f_c}(j, l) + d_{f_c}(j, l) + 1$ 
8   end while
9   if  $t_{in} + d_f(i, l) > t_{out}$  then
10    return failure
11  else
12    allocate $S$ ,  $\Phi_f(i, l) = t_{in}$ 
13  end if
14 end for
15 return  $S$ 

```

After termination of the patching algorithm, the node notifies the leader with a PF (Patch Frame) of the result so it can determine the next step. Moreover, if the SHP group node finds a valid schedule, it will send the modifications of the schedule to its neighbor with a SF. The purpose of SF (Schedule Frame) is to convey the potential changes in the schedule to the neighbor node in order to also change the schedule in its receiving link in case the leader chooses to allow the schedule update.

B. Update

When the leader has received all PFs from all the other SHP group nodes, and if all were successful patching the schedule, it sends back a UF (Update Frame) containing the update time where all nodes need to switch to their new obtained schedule. However, if the leader receives a PF indicating that a node could not find a valid schedule, it will send a UF to abort the update and wait for the result of the optimization step.

Note that if a link that belongs to the path connecting the SHP group suffers a failure, the nodes will not be able to communicate, leading to timeouts of the SHP activation. However, we expect this probability to be small. Assuming independent link failures, it is calculated as the multiplication of the SHP response time, the number of links in the SHP group and the probability of a link failure. For common cases, assuming a link failure of 10^{-6} , the probability of this case is 10^{-7} . The consequences of the timeout will be termination of the SHP activation without any modification of the schedule.

Still, there exist a problematic case of an inconsistent Update step which results in an inconsistent modification of the schedule. If a link fails at a time when the node should receive UF but the link failure was still not detected, it will cause an update of the schedule in some nodes only, while others will not receive the order. This case, even if extremely unlikely, can be reduced by the implementation of an *ACK* frame after the schedule update. It does not entirely remove the possibility of inconsistent schedule updates but it reduces it to 10^{-9} , using the same values as in the previous calculations.

C. Optimization

The nodes start the optimization step once they finish the patching step. Because of the quick patch algorithm, the SHP can employ more time to find a high reparability solution to increase the success rate of the SHP for subsequent link failures. But optimization still needs to be quick because there can be cases where the patching might fail. The SHP employs an ILP solver in conjunction with a two-phase algorithm that

seeks to maximize the schedule reparability. Note that the ATRs for the patching and optimization are the same, so no new information is required from the leader.

We present the optimization algorithm pseudo-code in Listings 2. The first phase will try to allocate the new frames one by one without modifying the original schedule following an incremental approach. But, whenever a valid schedule cannot be found, a second phase in which all offsets can be modified is activated. This provides more opportunities for the ILP solver to find a valid solution as the whole link can be modified at the cost of an increase in response time. The maximize reparability call to the ILP solver tries to maximize the distances between frames [10].

Listing 2. Optimization Algorithm Pseudo-code

```

1 function Optimization_Algorithm (ATRs)
2   % Phase 1, not modify schedule
3   ILP <- add(S)
4   for ATRf(i, l) = [tin, tout] in ATRs do
5     ILP <- add(tin ≤ Φf(i, l) ≥ tout)
6   end for
7   S = Maximize_Reparability(ILP)
8   if S failed then
9     % Phase 2, modify schedule
10    S = ∅
11    for ATRf(i, l) = [tin, tout] in ATRs do
12      ILP <- add(tin ≤ Φf(i, l) ≥ tout)
13    end for
14    for f(i, l) in S do
15      ATRf(i, l) = [tin, tout] = get_ATR(f)
16      ILP <- add(tin ≤ Φf(i, l) ≥ tout)
17    end for
18    S = Maximize_Reparability(ILP)
19    if S failed then
20      return failure
21    end if
22  end if
23  return S

```

When the optimization algorithm has finished, it mimics the process of the patching step. The membership node sends an OF (Optimization Frame) with the result of the optimization. In case it obtained a valid schedule, it also transmits the potential new schedule to its neighbor with a SF. The SHP finishes activating the update step again. If all OF are successful, it sends back the update time in a UF to all the SHP group nodes. In case any of the nodes fail to find a schedule, the SHP could not repair the schedule after the link failure. The leader could notify to a higher instance of the link failure, such that further correcting actions can be performed.

D. Multiple Protocol Activations

It is possible to repair multiple simultaneous or consecutive link failures occurring when an SHP activation is still active. When multiple SHPs are active, a node might belong to multiple SHP groups at the same time. This case can be accommodated with a straightforward queue system to send protocol frames or perform actions to different SHPs queries. However, the response time might increase drastically, especially when a node needs to perform multiple optimizations for different SHPs at the same time. To decrease the response time in this case, we enable the combination of the optimization step of different SHPs as it is the most time consuming step. We allow a recent optimization call to be aborted if another is received. In this case, the solution of simultaneous SHPs will be found at the same time. The main drawback of this

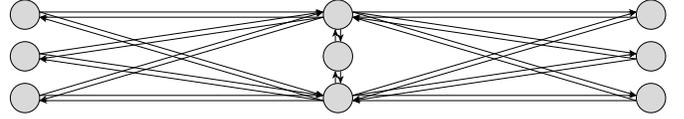


Fig. 5. Small Network Topology

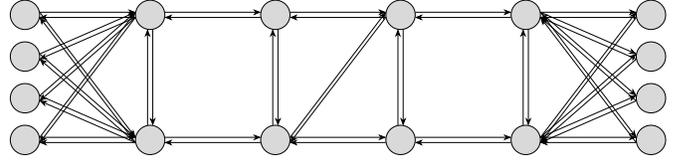


Fig. 6. Larger Network Topology

approach is that if a link failure cannot be repaired, the result of the combined optimization will be a failure, even if all the other links could be repaired if performed independently.

VI. EVALUATION

To evaluate the success rate of our distributed SHP approach to repair networks, we apply the protocol to the same two synthetic networks where a global knowledge repair algorithm was applied [10]. The first network is a small network (Figure 5) consisting of 3 switches, 6 end systems, and 28 links. The second network is a larger network (Figure 6) with longer paths, consisting of 8 switches, 8 end systems, and 54 links. We also synthesized an extremely large network three times larger than our large network, including 24 switches, 24 end systems, and 168 links. The networks incorporate two link classes related to capacity; 50 MB/s when the link connects a switch with an end system, and 100 MB/s when the link connects two switches.

We are interested in evaluating the response time in relation to network size and amount of traffic. We have incremented the number of frames considered in each network from 50 to up to 250. The schedule is obtained with an incremental approach scheduler [12] adapted to obtain high reparability schedules. We designed the traffic to amount for a high distribution, with a 10% of frames having one sender and only one receiver, 40% of frames having a random number of receivers, and 50% of frames being broadcasted to all the end systems. High distributed networks present higher utilization networks where all links have a similar utilization which we consider to be the main driver to increase response time for SHP.

Every frame size is set to 1500 bytes and the period is chosen randomly from 10, 20 or 40 ms, which generates a schedule with a 40 ms hyper-period. The maximum end-to-end latency is a tenth of the frame period. The time slots in the schedule are set to only one ns, which implies that every link will contain 4×10^7 time slots. Finally, we set the deadline to be equal to the period, and use the optimization parameters defined in [10].

This traffic yields a maximum link utilization of 28% for the small network, 30% for the large network and 31% for the extra large network. We do not consider higher utilization networks as we evaluate multiple link failures, which further increases the utilization around the affected links. E.g., after

two link failures, in some cases links achieve up to 90% utilization.

We have implemented a scheduler prototype that obtains high repairable schedules using the ILP Solver Gurobi v.8.1 with its Python API. We simulated the states of the SHP using our simulator also implemented using Python, together with the patching algorithm. The optimization algorithm was implemented using a modified incremental approach with the ILP Solver Gurobi v.8.1. The evaluations were performed on a MacBook Pro with 2.9 GHz CPU Intel Core i7 and 16 GB of RAM.

A. Repairability Results

We evaluate the success rate of SHP in comparison with a full network reschedule and a global knowledge repair algorithm. We utilize both networks with 250 frames to compare with our previous evaluation, considering up to three link failures. For the SHP, we differentiate two fault patterns: links can fail at a random point of time but only when no SHP is active, and all links fail simultaneously. We perform the study for all possible link failure combinations for the number of links studied and display the percentage of successful cases. We do not perform more than three link failures as the number of possible cases exceeds 10^5 .

We can observe in Table III the success rate for the small network, for full rescheduling, a globalized repair algorithm [10], the SHP with consecutive failures and the SHP with simultaneous failures respectively. Notice that the success rate of the SHP is almost identical to the global repair algorithm. For only a single link failure, it reaches a 100% success rate, the same as the full reschedule; the best possible success rate scenario. However, for more link failures, the localized repair decreases its success rate compared to the full reschedule, where the selection of not optimal paths to re-route the traffic affects its performance. Between the global repair and the SHP, we see that our protocol cannot solve a negligible number of cases. The reason for these failed cases is the distributed character of the SHP, which has to schedule the links separately at every SHP group node, missing some potential solutions from the state space. An interesting result is that simultaneous link failure with multiple SHP activations has the same success rate as independent link failures, which motivates the use of SHP when many failures are likely to happen at different points of the network at the same time. It is also important to recognize that the SHP also has difficulties with small networks topologies, where different paths to connect nodes are limited, and where after a few link failures the link utilization is very high. In our evaluations, we discovered that the SHP presents difficulties to allocate frames when the utilization is above 50% and starts to regularly fail to heal links with the utilization above to 60%. In order to increase the success rate, the traffic of the affected link could be split and allocated into different paths taking into account the utilization limitations described. However, we leave this for future work.

For the large network, we can observe a similar success rate between the global algorithm and the SHP, with an even closer success rate compared to the full reschedule. Larger

networks are more suited to localized repairs, since different available paths can be found. We notice that some cases could not be repaired with simultaneous link failures compared to independent protocol activation.

TABLE III
COMPARISON OF THE REPARABILITY BETWEEN A GLOBAL REPAIR ALGORITHM AND THE SELF-HEALING PROTOCOL

Network	Failures	Resched.	Global	SHP.	SHP Sim.
Small	1	1,0	1,0	1,0	1,0
	2	0,9682	0,9006	0,8915	0,8915
	3	0,9047	0,7288	0,7206	0,7206
Large	1	1,0	1,0	1,0	1,00
	2	0,9861	0,9809	0,9776	0,9770
	3	0,9570	0,9330	0,9310	0,9293
Extra Large	1	1,0	1,0	1,0	1,00
	2	0,9874	0,9218	0,9172	0,9165

B. Performance Results

We evaluate the performance of the SHP by investigating the response time to patch and optimize up to three link failures. Remember that when a schedule has been patched, all the frames are starting to be transmitted again, while when the schedule has been optimized, we modify the patched schedule to increase the reparability and increase the success rate for the subsequent link failures. For every network, we study up to 250 frames and a total of 1000 random link failure combinations at random times. We do not display the result variances, as they are highly dependent on which link is repaired and it would distort the displayed figures.

We can observe in Figure 7 the patching and optimization times for the small network after consecutive link failures, where a link can only fail when no SHP is currently active. An unexpected result is that there are no timing differences between the first, second and third link failures, which indicates that the deterioration after multiple link failures only affects the success rate, not the response time. We can also see that up to 150 frames, we have a patching time below 100 ms and an optimization time of less than half a second. Scalability issues appear for more than 200 frames, with more than six schedule cycles (240 ms) of frames lost on the faulty link and more than one second to optimize. The response time in the protocol is mostly a result of the patching and optimization algorithms, the rest of the protocol incurs on average less than 5 ms. We conclude that, similarly to the reparability results, the SHP has difficulties with very small networks. However, it is worth noting that this is a considerable improvement compared to reschedule, where the initial scheduling time took several minutes. Moreover, the SHP also takes into consideration uploading the new schedule into the network.

The patching and optimization times for the large network are displayed in Figure 8. We notice a significant difference in response times compared to the small network, where even for 250 frames the SHP needs less than 40 ms for patching and around 5 ms for 50 frames. The optimization time is also smaller, with less than half a second for all the studied cases. However, it has less uniformity, as with a larger network, there exists a more significant difference in the utilization of links.

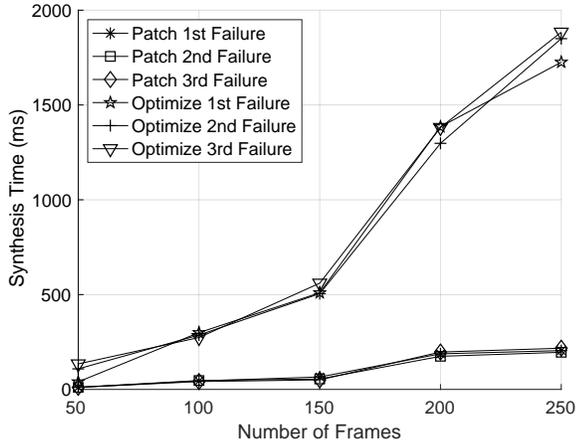


Fig. 7. Patching and optimization time to repair the schedule for the small networks after consecutive link failures for different number of frames

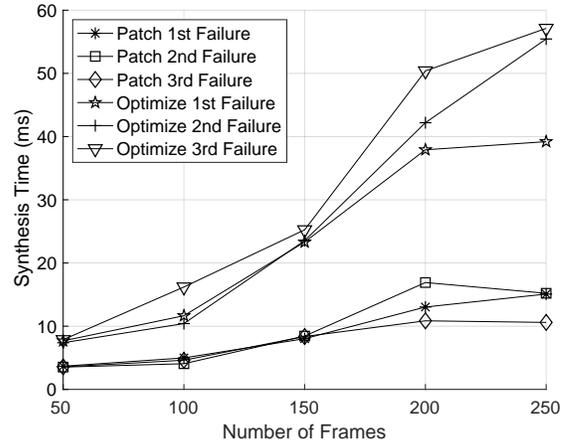


Fig. 9. Patching and optimization time to repair the schedule for the extra large networks after consecutive link failures for different number of frames

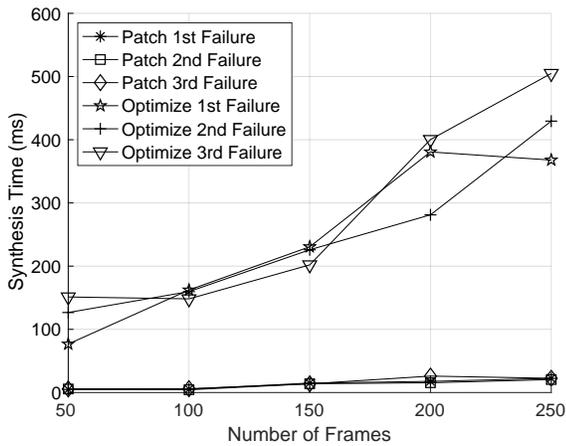


Fig. 8. Patching and optimization time to repair the schedule for the large networks after consecutive link failures for different number of frames

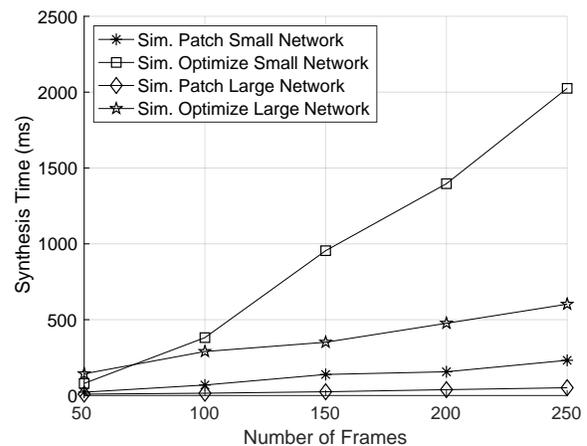


Fig. 10. Patching and optimization time to repair the schedule for two simultaneous link failures, for the small and large networks and different number of frames

To inspect if larger networks continue to influence the response times, evaluated the extra large network, three time larger than the large network. The patching time continues to be reduced, around 10 ms, and the optimization could repair the schedule in almost one schedule cycle (40 ms) in the worst case, as seen in Figure 9. We conclude that the SHP is faster in larger networks, as the traffic is more distributed.

Finally, we evaluate the response times when several links fail simultaneously. To enforce that nodes are part of multiple SHP groups at the same time, we impose two link failures on both link directions from the same physical link. This situation implies that both SHP activations have the same SHP group nodes and requires such nodes to try to patch and optimize schedules at roughly the same time. In Figure 10, we observe the patching and optimization times after both link failures, for the small and the large network. The response times are slower with multiple activations, with a small increase of the optimization time but almost double in the patching time compared with the previous evaluations. The capacity to

merge the optimization step allows the SHP to reduce the time compared to queuing the optimizations. However, we do not merge the patching, and therefore the node will start the first received patching order while queuing the second, resulting in about double response time for the latter.

VII. RELATED WORK

A varied range of research has been conducted to study how to enhance the static schedule to tolerate failures. Pop et al. proposed the re-execution of frames to deal with transient link failures [13] and quasistatic schedules where a set of schedules was synthesized as a prevention for the most likely permanent link failures [14]. An improvement of the Pop et al. replication was also proposed using frame replication over disjoint paths [15][16]. The same authors also enhanced this method including the topology to the synthesizing problem [17][18]. The idea of schedule mode changes as an improvement to quasistatic schedules was also considered, where

stacked schedules are generated, which can be modified to react to predefined events [19][20]. The crucial factor of all these studies is their fast reaction time, nearly instantaneously. However, they require a higher initial synthesis time, introduce extra traffic and most importantly they only cover a limited set of predefined events/failures, which for upcoming large-scale networks can only be a small fraction of all failures, since the number of failures is proportional to the network size. Hence, these techniques do not scale well or are not economically suitable for the whole network.

Methods to obtain schedules during run-time have also been studied. Zhang et al. proposed a cloud-based approach with a server containing a set of pre-computed schedules for different events [21][22]. If no schedule meets the requirements, the server synthesizes a new schedule and distributes it. Even though this approach can react to any change, a full schedule synthesis can take several minutes, and the synthesis time increases with network size. Other authors have aimed to meet the timing requirements by reducing the time to fully re-schedule. Nayak et al. proposed an incremental approach for time-sensitive software-defined networks that could find schedules during run-time in a few seconds [8]. However, the authors had to simplify the problem extensively, only networks with less than 13% utilization and a certain maximum path size could be scheduled, a utilization much smaller than the 60% utilization observed in our evaluations.

Raagaard et al. studied runtime reconfiguration of schedules for fog computing applying an incremental approach together with list scheduling heuristics [23]. When network change occurs, the reconfiguration algorithm removes all the affected frames and reschedules them iteratively. If not successful, a new schedule is attempted from scratch. Although this approach shares a relatively similar design scheme compared to SHP, it presents serious scalability issues due to its centralized nature. When a change occurs, their reconfiguration needs to notify the central node, reschedule and redistribute the changes in the schedule to the whole network again. For large networks, this process requires time and bandwidth resources that were not evaluated. Alternately, our SHP, being distributed, localises the changes needed in the network schedule, reducing the time to notify and update the schedule to a few milliseconds. In regards to the scheduling process, we appreciate in our evaluation that a link failure might cause the loss of up to 50% of the frames which would explain the exponential increase in reconfiguration time in their evaluation for larger networks. Alternatively, the proposed SHP localises the rescheduling to the few affected group of nodes that needs to change their local schedules. That not only reduces the complexity of the scheduling problem (and allows parallelization on the solving), but also avoids the scalability issues presented with larger networks. Moreover, the SHP can repair link failures occurring at different network locations due to the repair not affecting the schedules outside the SHP group.

Additional research has been performed to react to unpredictable changes at runtime. Avni et al. implemented a combination of offline and online techniques to tolerate k link failures where an online policy was activated for the recovery [24]. Their work has scalability issues as they rely on very

computationally demanding algorithms [25]. Additional online policies include [26], which implements an offline schedule with slacks to allow small adjustments of the frame timing. A similar concept was also applied to train communication networks where the traffic could be changed at the cluster level [27], [28]. In our approach, we repair the schedule at the required segment without any level limitations using rescheduling techniques instead of policies to enhance the success rate while seeking to keep a low response time.

In the context of TT communication, self-adaption has also been researched [29]. For instance, to accommodate new message streams or to update traffic patterns. Our approach does not address this problem, although it is possible that the self healing capabilities can be exploited in order to implement local adaption. The most evident case is addition of a new link, which can be seen as the reciprocal of a link crash. However these alternative uses of our protocol have not been investigated, as our focus is on fault tolerance exclusively.

VIII. CONCLUSIONS

Time-triggered networks need to be enhanced with flexibility and adaptability, particularly for large networks and considering link failures. Existing solutions are computationally, timing and cost expensive, or can only handle a limited number of predefined possible changes. We propose a distributed Self-Healing Protocol that recovers from link failures at runtime by forming a group of nodes which collaborate to repair a small schedule portion instead of the whole schedule. Our evaluations indicate that SHP may reach a similar repair success compared to a global knowledge approach, even in the presence of simultaneous links failures.

The SHP repair is several orders of magnitude faster than full rescheduling, requiring only around 200 ms for small networks and 10 ms for large networks, instead of several minutes (or even hours) of a complete rescheduling. We find that small network repairs tend to reallocate traffic in only a few links, causing saturation and increasing the repairing time. To increase the likelihood to repair after a link failure, we also introduce an optimization step that finds a high reparability schedule after the patching process in less than 2 seconds for small networks and less than half a second for larger networks. In the presence of simultaneous link failures, the patching time is doubled, but the optimization time stays almost unaltered. These results show that static schemes, e.g. [14], can be more efficient for small networks, but our protocol scales well with large networks. Both techniques are complementary and can be integrated to increase the performance of both approaches. We leave this integration as an open problem.

For future work, we would like to enhance our protocol to consider an extended fault model, for instance with node failures, and also the integration of new elements during runtime (i.e plug-and-play schedules); although we believe that a fully distributed solution, based on local information only, might not be efficient. On top of that, requiring all nodes to have solving capabilities might be expensive to implement. Thus, we plan to design a new protocol where only a set of high-performance nodes possesses solving capabilities. Moreover, such nodes,

while no protocol is active, will recollect information of their surrounding nodes and build knowledge of the network status. This knowledge will not only allow them to better react to different events, but it could also use the idle time to create schedules preventively, for more likely events.

REFERENCES

- [1] H. Kopetz and G. Bauer, "The Time-Triggered architecture," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 112–126, 2003.
- [2] 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.
- [3] R. Makowitz and C. Temple, "Flexray: a Communication Network for Automotive Control Systems," in *IEEE International Workshop on Factory Communication Systems*, 2006, Conference Proceedings, pp. 207–212.
- [4] W. Steiner, "TTEthernet Specification," *TTTech Computertechnik AG, Nov*, vol. 39, p. 40, 2008.
- [5] Institute of Electrical and Electronics Engineers, Inc. 802.1Qbv - Enhancements for Scheduled Traffic. [Online]. Available: https://standards.ieee.org/standard/802_1Qbv-2015.html
- [6] W. Steiner, P. G. Peón, M. Gutiérrez, A. Mehmed, G. Rodriguez-Navas, E. Lisova, and F. Pozo, "Next Generation Real-Time Networks based on IT Technologies," in *21st IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2017, pp. 1–8.
- [7] L. L. Bello, "The Case for Ethernet in Automotive Communications," *ACM SIGBED Review*, vol. 8, no. 4, pp. 7–15, 2011.
- [8] N. G. Nayak, F. Dürr, and K. Rothermel, "Incremental Flow Scheduling and Routing in Time-Sensitive Software-Defined Networks," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 5, pp. 2066–2075, 2018.
- [9] F. Pozo, H. H. Rodriguez-Navas, Guillermo, and W. Steiner, "SMT-based Synthesis of TTEthernet Schedules: a Performance Study," in *10th International Symposium on Industrial Embedded Systems (SIES)*, 2015, Conference Proceedings, pp. 162–165.
- [10] F. Pozo, G. Rodriguez-Navas, and H. Hansson, "Schedule Reparability: Enhancing Time-Triggered Network Recovery upon Link Failures," in *25th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2018, pp. 1–10.
- [11] 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.
- [12] W. Steiner, "An Evaluation of SMT-based Schedule Synthesis for Time-Triggered Multi-hop Networks," in *31st IEEE Real-Time Systems Symposium (RTSS)*, 2010, Conference Proceedings, pp. 375–384.
- [13] 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.
- [14] 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.
- [15] 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.
- [16] 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)*, 2015, Conference Proceedings, pp. 1–6.
- [17] V. Gavrilut, D. Tamas-Selicean, and P. Pop, "Fault-tolerant Topology Selection for TTEthernet Networks," in *Proceedings of the Safety and Reliability of Complex Engineered Systems Conference*. Citeseer, 2015, Conference Proceedings, pp. 4001–4009.
- [18] A. A. Atallah, G. B. Hamad, and O. A. Mohamed, "Fault-resilient Topology Planning and Traffic Configuration for IEEE 802.1 Qbv TSN Networks," in *24th IEEE International Symposium on On-Line Testing And Robust System Design (IOLTS)*, 2018, Conference Proceedings, pp. 151–156.
- [19] G. Durrieu, G. Fohler, G. Gala, S. Girbal, D. G. Pérez, E. Noulard, C. Pagetti, and S. Pérez, "DREAMS about Reconfiguration and Adaptation in Avionics," in *7th European Congress on Embedded Real Time Software and Systems (ERTS)*, 2016, Conference Proceedings.
- [20] F. Heilmann, A. Syed, and G. Fohler, "Mode-changes in COTS Time-Triggered Network Hardware without Online Reconfiguration," *ACM SIGBED Review*, vol. 13, no. 4, pp. 55–60, 2016.
- [21] L. Zhang, D. Roy, P. Mundhenk, and S. Chakraborty, "Schedule Management Framework for Cloud-Based Future Automotive Software Systems," in *22nd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2016, Conference Proceedings, pp. 12–21.
- [22] D. Majumdar, L. Zhang, P. Bhaduri, and S. Chakraborty, "Reconfigurable Communication Middleware for Flex Ray-based Distributed Embedded Systems," in *21st IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2015, Conference Proceedings, pp. 159–166.
- [23] M. L. Raagaard, P. Pop, M. Gutiérrez, and W. Steiner, "Runtime Reconfiguration of Time-Sensitive Networking (TSN) Schedules for Fog Computing," in *Fog World Congress (FWC)*. IEEE, 2017, Conference Proceedings, pp. 1–6.
- [24] G. Avni, S. Guha, and G. Rodriguez-Navas, "Synthesizing Time-Triggered Schedules for Switched Networks with Faulty Links," in *IEEE International Conference on Embedded Software (EMSOFT)*, 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] N. Wang, Q. Yu, H. Wan, X. Song, and X. Zhao, "Adaptive Scheduling for Multi-cluster Time-Triggered Train Communication Networks," *IEEE Transactions on Industrial Informatics*, 2018.
- [28] Q. Yu, T. Wang, X. Zhao, H. Wang, Y. Gao, C. Lu, and M. Gu, "Fast Real-Time Scheduling for Ethernet-Based Train Control Networks," in *Intl Conf on Parallel Distributed Processing with Applications, Ubiquitous Computing Communications, Big Data Cloud Computing, Social Computing Networking, Sustainable Computing Communications (ISPA/IUCC/BDCLOUD/SocialCom/SustainCom)*, Dec 2018, pp. 533–540.
- [29] M. Gutiérrez, A. Ademaj, W. Steiner, R. Dobrin, and S. Punnekkat, "Self-configuration of IEEE 802.1 TSN Networks," in *22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2017, Conference Proceedings, pp. 1–8.