Semi-Distributed Self-Healing Protocol for Online Schedule Repair after Network Failures

***Abstract*—Adaptive requirements for networks with strict timing restrictions do challenge the static nature of the time-triggered communication paradigm. Continuous changes in the network topology during operation require frequent rescheduling, followed by schedule distribution, a process that is excessively time-consuming as it was intended to be performed only during the design phase. The fully-distributed Self-Healing Protocol introduced a collaborative method to quickly modify the local schedules of the nodes during runtime, after link failures. This protocol gets the network back to correct operation in milliseconds, but it assumes that only the nodes are able to modify their local schedules, which limited the achieved improvement. This paper proposes to shift to a semi-distributed strategy, where high-performance nodes are responsible for the nodes and links within a small network segment. These nodes rely on their privileged view of the system in order to reduce the response time, increase the healing success rate, and extend the fault model to include switch failures.**

***Index Terms*—Scheduling, Real-Time, Networking, Time-Triggered, Fault-Tolerance**

I. Introduction

Time-Triggered (TT) networks are implemented in systems that require low latency, high determinism and high reliability. These networks operate following a so-called TT schedule, which is a static schedule specifying the transmission times for all the synchronous traffic over the network, i.e. over each link. This schedule is precomputed during the design phase and, in principle, does not change during system operation, which is a serious limitation, as it does not allow the system to adapt to changes of the underlying topology nor to traffic changes.

There is a growing interest in addressing the flexibility limitations of TT, in order to extend this paradigm towards larger time-critical applications, such as Real-Time Internet of Things [1], e.g. a fleet of autonomous vehicles [2]. In particular, the specific problem of achieving fault tolerance upon network failures has received much attention. A traditional way to achieve fault tolerance is by adding static redundancy into the TT schedule, for instance, by allocating several transmission slots for the same frame (temporal redundancy) or by scheduling one message stream over a number of disjoint paths (spatial redundancy). However, these approaches introduce extra complexity in the scheduling problem, which is an NP complete problem *per se*, and lead to low utilization of the network resources. As systems increase in size, and resource utilization becomes paramount, alternative methods based on dynamic rescheduling need to be developed.

All the methods for dynamic rescheduling follow the same procedure: they (1) identify when a schedule is not valid anymore, either because of a system failure or because of a traffic change, (2) calculate a schedule that is valid for the new situation and (3) distribute it among the nodes. The existing solutions differ significantly in the way these phases are implemented. A first division is between predictive and reactive strategies. Predictive rescheduling means that the system precomputes (in design time) a series of alternative, quasistatic schedules for a specific set of potential network failures [3]. Whenever one of the considered failures happens, the network selects the appropriate schedule immediately. This approach reduces the time needed for phases (2) and (3) dramatically, since the alternative schedules are precomputed and can already reside in the switches, but it is useful only for the considered failures. In contrast, reactive strategies do not assume anything about the potential failures, they calculate a valid schedule at runtime based on the available information and distribute it. Therefore, reactive strategies are more general but also slower because of the high complexity (NP-Complete) associated to obtaining such new schedules. Several techniques have been proposed for accelerating reactive rescheduling. For instance, applying a heuristic centralized algorithm that schedules only the affected frames [4]. The evaluation of this technique showed a very competitive response time upon component failures, in the range of seconds, but presented scalability issues when either the network size or the traffic were increased.

An interesting reactive approach, which makes dynamic rescheduling fast enough to be performed online, is the fully-distributed Self-Healing Protocol (SHP-FD) presented in [5] (cite TII2018). This approach modifies only a small section of the schedule, the one affected by the failure, instead of re-scheduling the whole network. Such a strategy allowed them to repair schedules after a link failure within milliseconds' range in almost every case, even after multiple consecutive link failures. Most importantly, it did not suffer from scalability problems when implemented in larger networks, thanks to its localized nature. On the negative side, the SHP-FD is implemented as a fully-distributed approach and presents some serious limitations when collaboration among the nodes is required in order to perform the rescheduling, making it unsuitable for covering other important failures like switch failure. These limitations are not present in the centralized approaches for reactive rescheduling.

Therefore, there seems to be a conflict between performance and rescheduling success. Centralized approaches are slower and not scalable enough, but can find better solutions because the central node has complete knowledge of the system. Fully

distributed approaches are faster and scalable, but the nodes can only use very partial knowledge of the system, thus not being able to find valid schedules in every occasion, even if such schedule exists.

In this paper we intend to overcome this dichotomy. We propose a Semi-Distributed Self-Healing Protocol (SHP-SD) that achieves the benefits of centralized approaches as well as the good performance and scalability of the SHP-FD. Our approach exploits the capabilities of specific switches that recollect information and have complete knowledge of their surrounding nodes and links. These privileged switches can thus compute and apply the required changes after a link failure in an equivalent manner to the SHP-FD. Additionally, thanks to having this complete knowledge of their surroundings, the switches in SHP-SD can implement three important features that are not present in SHP-FD: (a) A preemptive repair of potential failures in which a classification of the most disruptive failures is precomputed when the protocol is idle. (b) An advanced path selection mechanism that increases the success rate by selecting the most suited alternative path for every network segment affected by the failure. (c) A protocol extension to support switch failures. Our evaluation of SHP-SD shows a reduction of the repair time from hundreds of ms to 2 ms, for link failures, with a maximum of 10ms for switch failures. The success rate increased to values comparable to full, centralized rescheduling.

II. Preliminaries

A. Time-Triggered Networks

We define a multi-hop network as a directed graph $G = (V, L)$, where V includes Switches S and End Systems E, and L represents links with capacity C_l in Bytes per second. A node v_x can transmit information to another node v_y only if they are connected trough $(v_x, v_y) \in L$. Note that switched networks do not support a direct connection between two end systems. We also make the distinction between two switch categories regarding their resources: Regular Switches (S_r) and High-Performance Switches (S_h), where the latter possess more memory and computer capabilities. The purpose of S_h will be clarified in Section III.

A *sender* end system can transmit data to one or multiple *receivers* end systems through a frame $f \in F$ where F specifies the set of all network frames. A path p_f is a non-cyclic sequence of links $[(v_s, v_{s+1}), ..., (v_{r-1}, v_r)]$ that connects the sender v_s to the receiver v_r for a frame f. Since a frame can have multiple receivers, we define the tree path of frame f, denoted TP_f, as the union of all paths from the sender to each individual receiver. Finally, we specify a frame as a tuple $f = \langle T_f, D_f, Z_f, TP_f \rangle$; where T_f is the frame period, D_F is the frame deadline, Z_f is the frame size in Bytes and TP_f is the previously defined tree path.

B. Scheduling Problem

The scheduling problem aims to assign a transmission time (Offset Φ) for all frames over each link of their tree path such that all frames reach their destination within their deadlines. Given that the frames are periodic, we only require to determine the offsets within the *hyper-period* $T_F = LCM(T_f), \forall f \in F$, calculated as the least common multiple for all network frames. Note that each frame f may be instantiated within the hyper-period more than once, the number of frame instances is calculated as $N_f = \frac{T_F}{T_f}$. Formally, we define the solution to the scheduling problem as $\Phi_f : [1, N_f] \times L \to \mathbb{N}^+ \cup \{*\}$ where $\Phi_(i, l) = t$ states the transmission time of the i-th frame instance over the link l. In the case that $l \notin TP_f$ then $\Phi(i, l) = *$.

Network and frame specific constraints are established atop the scheduling problem to model certain desired characteristics such as the time for a switch to process a frame. Describing said constraints is outside the scope of this article, but interested readers can find them in TT scheduling literature [6], [7]. Moreover, it is possible to assign objective functions to obtain schedules with particular attributes. For this work, we want to obtain schedules with high reparability. This objective function maximizes the distances between frame transmission in order to allow modifications of particular frames while maintaining the rest of the schedule unaltered. Schedules with high reparability are shown to allow partial modifications of schedules affected by a link failure [8].

C. Fault Model

We consider link and regular switch failures that might cause a considerable loss of frames. Such failures can appear in the form of permanent failures, or transient failures long or frequent enough to significantly affect the network functionality. Failures can materialize at any point in time. Therefore, the network can suffer from more than a failure, as well as multiple failures occurring at the same time. Every node can detect link failures at incoming links based on their local schedule: if expected frames are omitted or consistently unsynchronized, the node activates the protocol.

Regarding the nodes, we assume that high-performance switches are fault free and that regular switches can exhibit only benign failure, i.e. crash or stop failures. With these assumptions, the high-performance switches can detect regular switch failures by periodically pinging the status of all the neighbouring switches.

D. Schedule repairs and self-healing

Using the terminology of [8], a repair of the network schedule happens when a new schedule is obtained not by solving the whole scheduling problem again, but by performing a number of changes over an existing schedule. We say that self-healing occurs if these adjustments are calculated by the affected switches in some distributed manner, and are not given by a centralized entity. The objective of the self-healing approach is to reduce the total repair time and minimize the number of frames that are lost while the repair is going on. This is achieved by acting only on the vicinity of the failure, which in most cases is enough for finding the required adjustments.

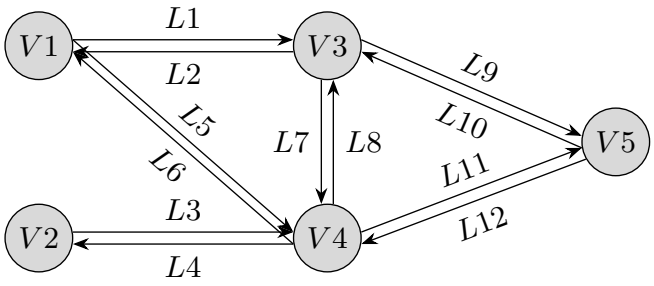


Fig. 1: Network Example

Our specific repair strategy is to reallocate the frames transmitted over the affected link on an alternative path. For example, if link 9 in Figure 1 has a failure, the protocol redistributes all affected frames on the path formed by link 7 and 11. The procedure to implement this strategy consists of five steps: notification, problem definition (or membership), patching, update and optimization. In the next section we will describe how they are implemented in a semi-distributed protocol.

III. Semi-Distributed Self-Healing Protocol

The most distinct characteristic of SHP-SDis that all S_h (and only them) undertake the task of adjusting the nodes schedules. As indicated, each S_h requests and stores information regarding the local schedules and the link status of the nearby nodes. It has been shown that such data can be collected in the tens of milliseconds range [9]. Thanks to this privileged information, whenever a S_h is notified of a failure, it already keeps all the information required to start the healing process immediately, without the need to organize a collaborative group of nodes, like it happened in SHP-FD . Note that to ensure the exchange of protocol frames among nodes, Bandwidth Reservation (BR) slots are allocated in all network links, in which only protocol frames can be transmitted. The steps of the protocol are as follows.

1) Notification: A node that detects a link failure, due to frame omissions in an incoming link, sends a Notification (NF) frame to its corresponding S_h . Recall that we we assume that the node already knows to which S_h to communicate and its path. In the case where the node is a S_h itself, the notification step is not necessary.

2) Define Healing Problem: Once the S_h receives the NF frame, it can start defining the strategy to heal the schedule. The protocol reroutes the affected frames of link $l_f = (v_x, v_y)$ through a new path $p_f = [(v_x, v_{x+1}), ..., (v_{y-1}, v_y)]$. To achieve this, the switch allocates the offsets of the new path links between the Available Transmission Ranges $ATR_f(i, p_f) = [t_{in}, t_{out}]$, where ATR contains the intervals of all possible offsets in which the instance i of frame f can be allocated without violating any schedule constraint. As p_f might contain multiple links, the ATR is divided equitably among the links in p_f, such that we obtain an $ATR_f(i, l), l \in p_f$ for every link in the path. At the end of the process, a repair problem has been defined for every link in p_f, which will be solved in order to identify where each affected frame shall be allocated.

3) Patching: The patching algorithm seeks to allocate all the affected frames into new offsets as fast as possible to reduce the number of frames lost during healing. A pseudo-code can be seen in Listing 1. For every ATR, the patching algorithm tries to allocate the frame at the start of t_{in}. If it collides with an already allocated frame, the algorithm tries to allocate it after the frame with which it collided. It repeats the process until the frame is allocated. When the algorithm allocates all the different frame ATRs, a repair adjustment has been found, and the update and optimization step starts. However, in the case that a frame cannot be allocated, the protocol skips the update phase and only executes the optimization phase.

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 , Φ_f(i,l) = t_in) or
          t_in + d_f(i,l) ≤ t_out do
4     f_c = get_frame_collides(S , Φ_f(i,l) = t_in)
5     t_in = Φ_{f_c}(j,l) + d_{f_c}(j,l) + 1
6    if t_in + d_f(i,l) > t_out then
7     return failure
8    else
9     allocateS, Φ_f(i,l) = t_in
10  return S
```

4) Update: Once the patching process obtains a new schedule adjustment, the S_h distributes the solution by transmitting Update (UF) frames to the nodes that will modify their schedules. It means that a UF frame is sent to every node that needs to change its local schedule, with the frame containing the modifications to the schedule and the time when the node shall swap to the given schedule. After this time, the network can be considered healed, since the new schedule enables transmission of all the frames again.

5) Optimization: Even though the patching algorithm heals the network, it drastically reduces the overall schedule reparability, which implies a reduction in the likelihood to repair future failures. On the other hand, the optimization algorithm, which maximizes the reparability, is computationally more expensive, and hence, more time-consuming. Nevertheless, once the network has been healed with a patch, the protocol can invest additional time to obtain a higher reparable solution as no frames are lost while optimizing. The pseudo-code of the optimization algorithm can be seen in Listing 2. Applying an Integer Linear Programming (ILP) Solver, the algorithm adds the link schedule as fixed offsets. Subsequently, all ATRs are also added to the ILP solver. Next, it also defines a cost function that maximizes the distances between the ATRs and the fixed frames allocated in the link schedule. Due to lack of space, such formulation is not described, but we point interested readers to the formulation of the cost function [8]. Once the algorithm obtains the solution, the update step starts. In the case no schedule could be found neither with the patching or optimization, the protocol could not heal the network.

Listing 2: Optimization Algorithm Pseudo-code

```
function Optimization_Algorithm (ATRs)
 ILP <- add(S)
 for ATR_f(i,l) = [t_in, t_out] in ATRs do
  ILP <- add(t_in ≤ Φ_f(i,l) ≥ t_out)
 end for
 S = Maximize_Reparability(ILP)
 return S
```

IV. Extensions

This section introduces extensions to the SHP-SD to both improve the performance in terms of response time and success rate. Furthermore, the detailed knowledge kept by the S_h allows extending the fault model and tolerate switch failures with minimal protocol modifications.

A. Preemptive Healing

The SHP-SD is a reactive protocol that waits for a failure to be detected and activates the process of healing the schedule. However, failures are rare occurrences; the protocol is idle most of the time. This implies that all the S_h and bandwidth resources are underutilized. We present a Pre-emptive Healing extension in which the unused resources can be utilized to decrease the average healing response time further. Some of the literature introduced the concept of back-up or quasi-static schedules [3], where a set of alternative schedules to tolerate potential failures were synthesized at design time. If the failure was detected during runtime and an alternative schedule existed, it would be swapped almost immediately. We want to combine such approaches with the SHP-SD to precompute healing modifications of the schedule against potential failures, particularly the most disruptive ones.

In the SHP-SD implementing preemptive healing, all S_h start to simulate a link failure to obtain an optimal adjustment (which results from the optimization step of the SHP-SD) and store it. This process is only allowed when no active failure is detected and immediately terminates if a failure is detected. However, even though failures are rare, the memory used for storing the alternative schedules is limited . Moreover, different failures have a considerable difference in healing time, depending on factors such as traffic on the faulty link [8]. Given the limit on resources, it is beneficial to precompute only the failures whose healing would require more resources and time; this would significantly reduce the number of lost frames in such complicated cases.

Selecting the most time-consuming failures is not trivial. In our experimentation, against common intuition, we found out that the healing of links with higher utilization would not always yield a higher response time. Other parameters, such as the available slots to maximize the reparability, also considerably affect the response time. E.g., when allocating frames to an empty link, where more opportunities for optimization of reparability are available, the time to obtain a solution increases. We implemented a Machine Learning solution applying Support Vector Machines to predict the complexity of healing a link failure or if it can fail to heal it. Every S_h contains an already trained model of previously simulated failures from different network traffics and topologies. The S_h employs the model to predict the complexity of all potential failures and to start simulating failures of the predicted more time-consuming failures. We trained our prediction model using the following information related to a link failure:

- Broken link utilization
- Broken link number of frame instances.
- New Path link utilization.
- New Path link number of frame instances.

Note that, as the new path contains multiple links, the prediction of a link failure also contains multiple values, one for each link of the new alternative path. We consider only the maximum time-consuming prediction over all links. Other selection strategies and their effectiveness are left for future work.

Applying preemptive healing to the SHP-SD does not require exhaustive changes to the protocol steps. After the detection of a failure and its notification to the corresponding SHP-SD , if that specific failure has been already predicted, the patching and optimization steps are not necessary, and the stored schedule modifications are distributed. In such a case, the response time becomes almost instantaneous. If the failure has not been predicted, the SHP-SD follows its original procedure and no reduction of response time is achieved.

B. Path Selection

We note that in SHP-FD only the shortest path was considered for repairs, for simplicity. However, in the SHP-SD, every S_h knows multiple paths that connect the nodes. When a failure is notified, the Path Selection extension introduces the possibility to select alternative paths that might improve the success rate or even reduce the healing time. The information kept by each S_h is:

- Several alternative paths to connect both ends of the faulty link (if they exist).
- Utilization and frame instances transmitted on such paths.
- A prediction of the possibility of unsucessfully healing for every path.

We propose a path selection algorithm that ranks all the alternative paths given their prediction of the complexity to be solved. In the same manner as the pre-emptive healing, we select the path that has a minimum complexity prediction from all its links. If multiple paths have the same complexity prediction, we choose the path with a minimum sum of link utilization. Other selection strategies and their effectiveness, such as average utilization, are left for future work.

1) Advance Path Selection: Even though the path selection extension is an improvement compared to only considering the shortest path, it does not appreciably increase the success rate. There still exist several cases where full rescheduling can find a valid schedule, but our protocol could not. This is because the SHP-SD seeks to reconnect the nodes connected to the affected link. However, alternative direct paths between two nodes are scarce since it is usually not a common practice in network topology design.

Typical network designs have alternative paths that are usually disjoint on the majority of switches. E.g., if $l_9 = (v_3, v_5)$ breaks in Figure 1 an alternative path that connects nodes v_3 and v_4 again is $[l_7, l_{11}] = [(v_3, v_4), (v_4, v_5)]$. The SHP-SD would then substitute in f_1 and f_2 the affected link with the new path resulting in the path $[l_1, l_7, l_{11}] = [(v_1, v_3), (v_3, v_4), (v_4, v_5)]$. However, with a more broad knowledge of the surrounding nodes, we can consider that both affected frames start at v_1, and there exist a path $[l_{5,11}] = [(v_1, v_4), (v_4, v_5)]$ that skips the need to transmit over v_3 and obtains a path that is shorter and most likely easier to heal. Moreover, f_2 already has a transmission scheduled over l_5, which further simplifies the problem as there is no need of a new reallocation on that link.

Note that implementing the advanced path selection extension requires some modifications when obtaining the affected frames ATR . Previously, all affected frames would require the same change in the paths after a failure, as a link was exchanged with a new path. However, as the new path selection allows to exchange sections or even entire paths, it means that the SHP-SD needs to consider a new path for every affected frame. E.g., if in the last example a new link l_n is introduced together with a new frame f_n with path $[l_n, l_9] = [(v_2, v_3), (v_3, v_5)]$, the new path for the whole frame would be $[l_3, l_{11}] = [(v_2, v_4), (v_4, v_5)]$ which is different than the paths obtained for frames f_1 and f_2.

The algorithm to obtain advanced paths can be seen in Listings 3. The idea behind it is to check all possible paths that connect the nodes of the affected frames of which the S_h has knowledge. First, the S_h iterates over all paths of the affected frames by the failure of l_n and proceed if that specific path contains the link l_n (some paths of an affected frame might not contain the affected link). The S_h iterates over all the nodes of the affected path p that are before l_n, including $v_s \in l_n = (v_s, v_r)$. The next loop is on the opposite direction of the path, over all the nodes after p that S_h has knowledge about, including v_r. Finally, the S_h calculates all simple paths that connect both nodes and store them. Once all possible paths for every node combination is obtained, the best past is selected using the previously introduced path selection strategy.

Listing 3: Smart Paths Algorithm Pseudo-code

```
1  function Smart Paths (affected frames)
2    for f, p in affected frames
3      if l_n in p
4        for node_b in p[0,l_n]
5          for node_a in [l_n, last]
6            all_p <= all_simple_p(node_b, node_a)
7        new_p = best_p(all_p)
```

C. Regular Switch Healing

In this subsection, we extend the fault model to allow permanent or temporal S_r failures that are long enough to require healing the schedule. Such failures can be regarded as multiple simultaneous failures of all links connected to the switch. Multiple simultaneous link failures were shown to be feasible to heal [8]. However, S_r failure healing demands two alterations on the protocol: the S_r failure notification and the selection of the paths for the affected frames.

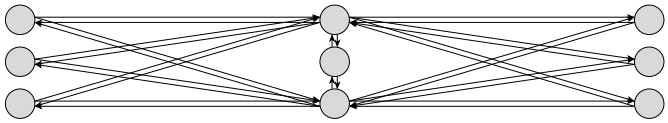


Fig. 2: Small Network Topology

1) Regular Switch Failure Notification: The same notification procedure to detect link failures cannot be applied to detect switch failures. If a switch fails, all the nodes connected to it will detect a link failure. However, the SHP-SD cannot distinguish between a switch failure and the unlikely case when all its links had a failure. Moreover, all the links that are sending frames to the affected switch will not be notified of its failure, as the node that should notify it is the faulty one. Nevertheless, as all S_h have knowledge of the nodes they are monitoring, we can implement a simple periodic ping frame to check the status of all nodes. If failures of links start to manifest together with a node that is not sending a ping response back, the SHP-SD concludes that the node has a failure and starts defining the healing problem.

2) Affected Frames Path Selection: The definition of the healing problem requires to individually determine the healing problem for all the links connected to the node, and then group the problem. However, a major change is required when selecting an alternative path. In the case of link failures, the protocol needed to find an alternative path that connects both ends of the faulty link. But this is not the case anymore, as when a switch fails, the protocol has to find a path that connects to nodes that come after (from the frame path perspective). This can be solved by applying the advanced path selection introduced in Section IV-B1. E.g., in Figure 1 if v_3, f_1 and f_2 are affected when transmitting to v_5. The advanced path algorithm will search paths that connect v_1 to v_5 instead, e.g., $[l_5, l_{11}] = [(v_1, v_4), (v_4, v_5)]$.

V. Evaluation

To evaluate the SHP-SD , we apply the protocol to the same two synthetic networks where a global knowledge repair algorithm was applied [8]. The first is a small network (Figure 2) consisting of 3 switches, 6 end systems, and 28 links. The second is a larger network (Figure 3) with longer paths, consisting of 8 switches, 8 end systems, and 54 links. A third super large network is formed as three interconnected larger networks, consisting of 24 switches, 24 end systems, and 166 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 concerning network size and amount of traffic. We have incremented the number of frames considered in each network from 50 to up to 250. 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

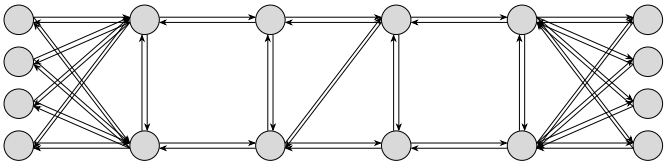


Fig. 3: Larger Network Topology

50% of frames being broadcasted to all the end systems. High distributed networks present higher utilization 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 of 40 *ms* hyper-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 [8] to compare the results.

We have implemented a scheduler prototype applying an incremental approach [6] modified to obtains high reparable schedules with C and the ILP Solver Gurobi v.8.1. We simulated the states of the SHP using our simulator implemented in Python. The patching algorithm is implemented in C, and the optimization algorithm was implemented in C and ILP Solver Gurobi v.8.1. The classification for the support vector machines depending on the optimization time can be seen in Table I. Class 0 means that a solution could not be found. The training has been performed with different networks and traffics, achieving an accuracy of 90% when assessing the networks evaluated in this work. A support vector machine was implemented in Python with the package *sklearn*. The evaluations were performed on a MacBook Pro with 2.9 GHz CPU Intel Core i7 and 16 GB of RAM.

TABLE I: Classification for the Support Vector Machine depending of the optimization class in milliseconds

Class	0	1	2	3	4	5
Min (ms)	-	0	100	1000	2500	5000
Max (ms)	-	100	1000	2500	5000	-

A. Reparability Results

We evaluate the success rate of the SHP-SD , its extensions with path selection and its extension with advanced paths against a fully distributed approach (SHP-FD) (cite should be the TII Journal), a global centralized approach [8] and full network reschedule. We perform our evaluation with all described networks considering traffic of 250 frames and up to three link failures for the small and large network and 2 for the super large network. We only consider link failures when no healing is active due to a previous link failure. We perform the study for all possible link failure combinations for the number of links and display the percentage of successful healing cases. Due to timing restrictions, we do not consider more than two link failures for the super large network as the possible combinations of two link failures is already superior to 10.000 cases.

We can observe in table II that full rescheduling has the highest probability of success as it will always find a schedule if it exists; this is taken as the the best possible success rate. For a single link failure, all cases have a perfect success rate, as there always exists an alternative path and the utilization of all links is still low for the healing algorithm to be challenging. However, when the number of link failure increases, healing approaches reduce their success rate compared to full rescheduling depending on the path selected and the healing algorithm applied. The success rate of SHP-FD and SHP-SD are the same because both apply the same shortest path and the same optimization algorithm. Moreover, their success rate is the lowest among the studied cases due to only considering the shortest path. The globalized approach has a negligible better success rate due to a centralized optimization algorithm, where the full new path (shortest path) is solved at the central node instead of solved individually for each path link. Note that in the SHP-SD , this centralized optimization algorithm could have been implemented, but we considered that the small success rate improvement did not pay off for the increase in time for the algorithm to find a solution. The SHP-SD with the Path Selection extension also minimally increases the success rate. The number of alternative paths in the studied networks is usually not large enough to produce a difference compared to the shortest path. But in some cases, the algorithm detected a better success rate alternative path, usually with lower utilization. In the deepening of our evaluation, we discovered that the healing presents difficulties to allocate frames when the link utilization is above 60%. The more substantial increase in success rate, closer to full rescheduling, happens when applying the Advanced Paths extension. As the number of alternative paths significantly increases, including finding ”shortcuts”, there exist a larger number of options where the optimization algorithm can be successful. Note that for larger networks, the success rate is closer to full reschedule, showing that larger networks are better suited for localized repairs, since they provide many more alternative paths.

B. Performance Results

We assess the performance of the SHP-SD by measuring the response time from the time the failure is detected to the time when the repaired schedule is operative. We measure both when the patch schedule is operative and when the optimize schedule. For every network, we study up to 250 frames and show the average repair time for all the possible combinations for one and two link failures. In the case of two link failures, we only consider the case that a second failure occurs after the first has been repaired (simultaneous failures has been already investigated in (cite TII 2018)). Moreover, we display the summation of both failures response times. We evaluate three different configurations: (1) the SHP-SD with no extensions, (2) only applying preemptive scheduling always assuming the chosen schedules have been already calculated, and (3)

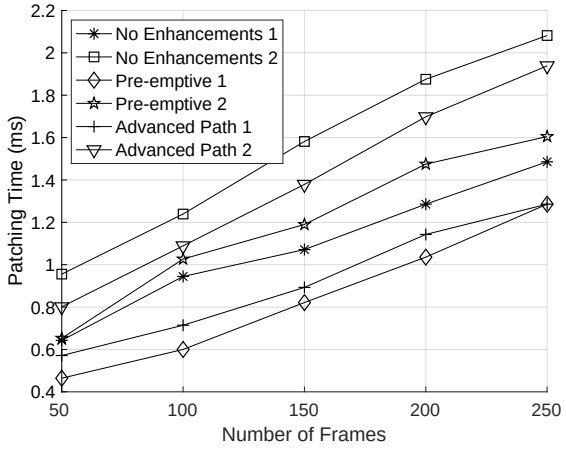
TABLE II: Comparison of the success rate between SHP-SD and its extensions against centralized and fully distributed approaches

Network	Failures	Rescheduling	Global	SHP	SSHP	SSHP Path Selection	SSHP Advanced Paths
Small	1	1,0	1,0	1,0	1,0	1,0	1,0
	2	0,9682	0,9006	0.8915	0.8915	0,9032	0,9378
	3	0,9047	0,7288	0,7206	0,7206	0,7338	0,8234
Large	1	1,0	1,0	1,0	1,0	1,0	1,0
	2	0,9861	0,9809	0,9776	0,9776	0,9794	0,9839
	3	0,9570	0,9330	0,9310	0,9310	0,9330	0,9489
S. Large	1	1,0	1,0	1,0	1,0	1,0	1,0
	2	0,9874	0,9218	0,9172	0,9172	0,9189	0,9509

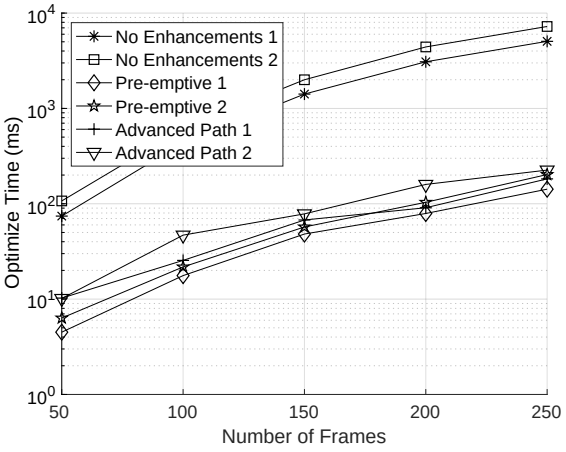
applying preemptive and advanced path selection together. Note that we limit the number of preempted schedules per S_h to only four.

We can observe in Figure 4 the healing times for link failures in the small network. The first distinction is the large difference in response time between patching and optimizing the schedule, with an average patching time for all cases inferior to 2 ms whereas the optimization can take up to a few seconds. These results accentuate the patching step importance, as after the patch schedule is operative, all the frames are being transmitted again. We can also notice the reduction on average time from seconds to hundreds of milliseconds when applying preemptive schedule, even though only four failures are preempted. This is because a small number of link failures are very time-consuming to optimize while the majority do not require more than hundreds of milliseconds. These failures are usually localized in the network backbone, where the traffic is concentrated. The fact that the average response time for two failures increases only around 20%, instead of becoming double, supports our assumption that the response time varies greatly depending of the faulty link. Note that applying the advanced path enhancement increases the response time, as the number of links that need to be modified increases. However, the increase is not significant because the number of modifications at each link is reduced. We can perceive the same response pattern in the large and super large networks in Figure 5 and Figure 6, respectively.

In Figure 7, we present the patching (continuous line) and optimization (dotted lines) time to repair a node failure for all three networks. We consider the average time to repair all the possible combinations of regular switches. As a switch failure can be seen as multiple link failures, the response times increase, but still they do not exceed 10 ms for the patching step. As for the optimization step, we studied it without preemption, but for a real implementation, we would recommend to first preempt node failures and then link failures, since healing the former is more complex. In the experiments, we obtained a maximum of 13 seconds response time, with similar response times for the different networks.



(a) Patching Time

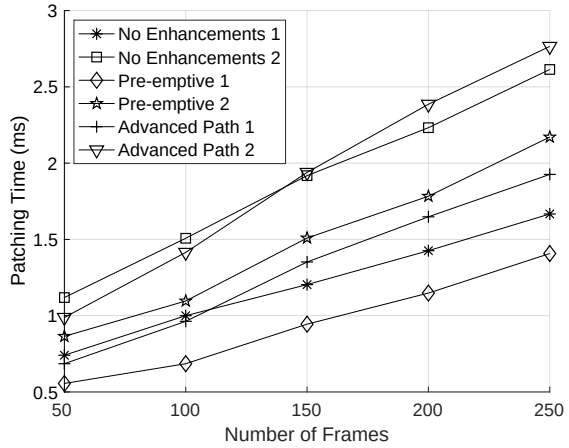


(b) Optimization Time

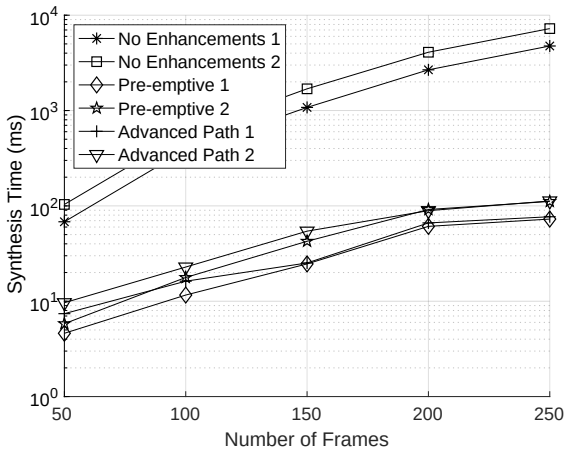
Fig. 4: Patching and Optimization time for different number of frames for the small network in milliseconds

VI. Related Work

Extensive literature exists about developing an adaptive time-triggered paradigm solely focused on increasing the network fault-tolerance capabilities. These strategies apply predictive replication, in which the frames or even the whole schedule are replicated in design time to cope with potential failures. Pop et al. have proposed to replicate and re-execute frames to deal with transient failures [10]. The frame re-execution was later improved by Wisniewski et al. replicating the frames in disjoints paths. This solution is more robust and can also tolerate permanent failures [11] [12]. Including

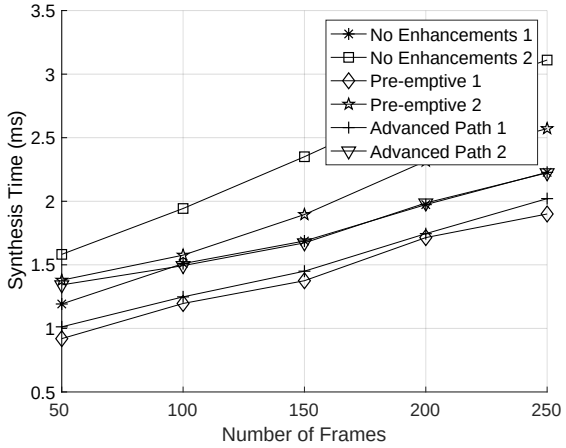


(a) Patching Time

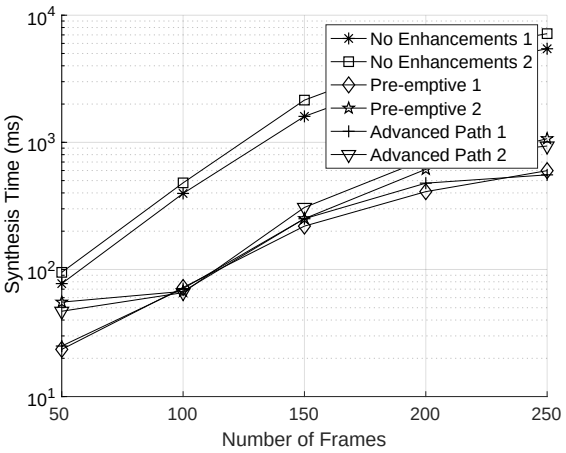


(b) Optimization Time

Fig. 5: Patching and Optimization time for different number of frames for the large network in milliseconds



(a) Patching Time



(b) Optimization Time

Fig. 6: Patching and Optimization time for different number of frames for the extra large network in milliseconds

the topology design problem into the scheduling problem allowed to extend the number of disjoints paths and increase the number of tolerated failures [13]. Novak et al. reduced the utilization required to apply frame replication with the introduction of F-shaped frames, as replicated frames are only transmitted if required [14]. In terms of schedule replication, Izosimov et al. introduced the notion of dormant quasi-static schedules, a set of schedules synthesized to be exchanged if certain failures occurred during runtime [3]. These works achieve an almost instantaneously response time after a failure. However, they only cover cases that were taken into account at design time, if a failure occurs outside of this set, the network is not able to tolerate it. Moreover, as the network size increases, the set of potential failures increases exponentially, becoming harder to cover the majority of cases.

A strategy to address unpredicted failures or even new components additions is to recalculate the schedule during runtime, a reactive strategy. Rescheduling will return a valid schedule if a solution exists. Zhang et al. have proposed a central node in the cloud that solves new schedules when a change occurs in the network [15]. Moreover, it stores the schedule in case the same configuration is needed again. However, obtaining a new schedule may take several minutes, an unacceptable amount of time in most safety-critical applications. To cope with the complexity problem, researchers have explored two options: reduce the complexity of the problem and partial rescheduling. Nayak et al. proposed to simplify the problem to reduce the synthesis time [16]. However, this approach still requires seconds to obtain a schedule and cannot schedule networks of more than 13% utilization. Raagaard et al. opted for partial rescheduling, where only the affected frames after a failure are modified applying a quick list scheduling heuristic [4]. Such heuristics struggle to obtain high utilization schedules. Moreover, they present scalability issues when the traffic and

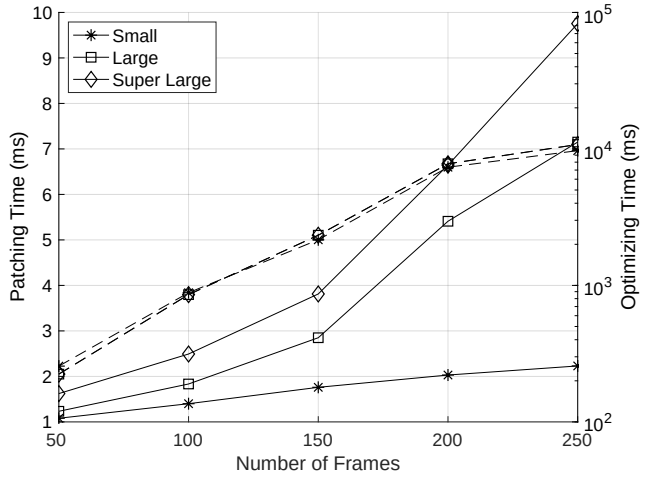


Fig. 7: Patching and Optimization time of nodes for different number of frames for all networks in milliseconds

network size is substantial. Wang et al. avoided a central approach and proposed a more scalable solution encapsulating the scheduling problem to the cluster level, where only a segment of the schedule required to be changed after a network modification [17]. But it was applied only to the specific case of train time-triggered networks and its extensability is limited. In our approach, we consider reducing the complexity by implementing a generalized encapsulation after a link failure and only reschedule the affected frames in such segment. Moreover, we shift the scheduling and distribution closer to the failure, further reducing the time to update the schedule when a solution is found.

VII. Conclusion

The semi-distributed self-healing protocol is able to repair time-triggered schedules at run-time after link failures in a few milliseconds. Moreover, after only a few seconds, it optimizes its reparability to increase the robustness for subsequent link failures. The employment of high-performance switches that posses information of their surroundings allows implementing extensions to the protocol to improve response time, success rate and extend the fault model. A Machine Learning algorithm was implemented to select which failures are more time-consuming to heal, which are precomputed when the protocol is idle, reducing the healing time of the hardest cases considerably. Moreover, an advanced path selection was proposed that increased the protocol success rate close to full rescheduling. Finally, the protocol was enhanced with minimal modifications to heal the majority of switch failures in a few milliseconds.

References

[1] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog Computing and its Role in the Internet of Things," in *Proceedings of the 1st Workshop on Mobile Cloud Computing (MCC)*. ACM, 2012, Conference Proceedings, pp. 13–16.

[2] 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. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0968090X15003447

[3] 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 11th conference on Design, Automation and Test in Europe (DATE)*. ACM, 2008, Conference Proceedings, pp. 915–920.

[4] M. L. Raagaard, P. Pop, M. Gutiérrez, and W. Steiner, "Runtime Reconfiguration of Time-Sensitive Networking (TSN) Schedules for Fog Computing," in *Proocedings of the Fog World Congress (FWC)*, 2017, Conference Proceedings, pp. 1–6.

[5] F. Pozo, G. Rodriguez-Navas, and H. Hansson, "Work-in-Progress: A Hot-Patching Protocol for Repairing Time-Triggered Network Schedules," in *Proceedings of the 24th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2018, Conference Proceedings, pp. 89–92.

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

[7] F. Pozo, G. Rodriguez-Navas, and H. Hansson, "Methods for Large-Scale Time-Triggered Network Scheduling," *Electronics*, vol. 8, no. 7, 2019. [Online]. Available: https://www.mdpi.com/2079-9292/8/7/738

[8] ——, "Schedule Reparability: Enhancing Time-Triggered Network Recovery Upon Link Failures," in *Proceedings of the 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2018, Conference Proceedings, pp. 147–156.

[9] ——, "A Semi-Distributed Self-Healing Protocol for Run-Time Repairs of Time-Triggered Schedules," in *Proceedings of the 24nd International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2019, Conference Proceedings, pp. 1–4.

[10] 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.

[11] L. Wisniewski, M. Schumacher, J. Jasperneite, and C. Diedrich, "Increasing Flexibility of Time-Triggered Ethernet based Systems by Optimal Greedy Scheduling Approach," in *Proceedings of the 20th Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2016, Conference Proceedings, pp. 1–6.

[12] L. Wisniewski, V. Wendt, J. Jasperneite, and C. Diedrich, "Scheduling of Profinet IRT Communication in Redundant Network Topologies," in *Proceedings of the 16th World Conference on Factory Communication Systems (WFCS)*. IEEE, 2016, Conference Proceedings, pp. 1–4.

[13] V. Gavrilut, B. Zarrin, P. Pop, and S. Samii, "Fault-tolerant Topology and Routing Synthesis for IEEE Time-sensitive Networking," in *Proceedings of the 25th International Conference on Real-Time Networks and Systems (RTNS)*. New York, NY, USA: ACM, 2017, pp. 267–276. [Online]. Available: http://doi.acm.org/10.1145/3139258.3139284

[14] A. Novak, Z. Hanzalek, and P. Sucha, "Scheduling of safety-critical time-constrained traffic with f-shaped messages," in *Proceedings of the 13th International Workshop on Factory Communication Systems (WFCS)*. IEEE, May 2017, pp. 1–9.

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

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

[17] N. Wang, Q. Yu, H. Wan, X. Song, and X. Zhao, "Adaptive Scheduling for Multi-cluster Time-Triggered Train Communication Networks," *Transactions on Industrial Informatics*, vol. 15, no. 2, pp. 1120–1130, 2019.