Fault Tolerant List Scheduler for Time-Triggered Communication in Time-Sensitive Networks

Maryam Pahlevan, Sarah Amin, and Roman Obermaisser
University of Siegen, 57068 Siegen, Germany
Email: maryam.pahlevan@uni-siegen.de; sarah.amin@uni-siegen.de; roman.obermaisser@uni-siegen.de

Abstract—The performance and dependability of modern mission-critical systems significantly depends on the communication infrastructure. In this context, the Time-Sensitive Networking (TSN) task group addressed different requirements of cyber-physical systems such as timing and reliability constraints. TSN provides real-time capabilities through sharing a global time reference and employing transmission schedule tables called Gate Control Lists (GCL). On the other hand, TSN masks faulty behaviors within a system through a technique called Frame Replication and Elimination for Reliability (FRER). FRER fulfills the safety requirements of mission-critical systems by message replication and transmission of message replicas over redundant paths. The scheduling problem for the GCL synthesis is NP-complete. For simplification of the scheduling process, several state-of-the-art solutions provide schedulers for fault-free networks. However, this assumption is very optimistic and in practice networks experience different faulty-behaviors over time. This paper extends our heuristic TSN scheduler which was developed for fault-free TSN systems to support the FRER mechanism. Our fault-tolerant TSN scheduler focuses on enhancing the reliability of a mission-critical system while meeting the deadlines of time-critical jobs. To achieve this goal, we introduce a novel reliability analysis approach for a mission-critical system with a TSN communication infrastructure. This approach models and evaluates the reliability of a system based on the reliability of message transmissions between safety-critical jobs. The reliability of message transmissions is computed based on the reliability of the network components that form the forwarding paths. Thereby, our reliability model enables the system designers to plan networks more optimally.

Index Terms—TSN, GCL, FRER, scheduling, reliability model, system reliability analysis

I. Introduction

Due to tremendous success and wide spread deployment of Ethernet technologies, the Time Sensitive Network (TSN) [1] group has introduced a series of IEEE 802.1 sub-protocols to offer a fault-tolerant and deterministic communication infrastructure for executing safety-critical applications over an Ethernet based network. In TSN networks, devices which support the IEEE 802.1ASRev [2] synchronize to a global clock reference. The TSN scheduling approach, Time Aware Shaper (TAS) [3] enforces a gate control list (GCL) at each port. The port-specific GCL determines the specific instant of time each egress queue is permitted to transmit a message. The computation of a TSN schedule is a NP-complete problem and the ever increasing number of network devices significantly increases the execution time for scheduling. Consequently, several solutions reduce the complexity of scheduling process by making abstractions. For instance, most of the state-of-the-art schedulers solve the routing and scheduling problem sequentially rather than computing them together. Most of the studies also assume that the communication infrastructure is completely fault-free which is a highly optimistic assumption. In practice, a TSN network may experience different changes over time such as device reconfiguration or device failure during transmission of messages. Failures in the TSN-aware devices of a cyber-physical system may lead to irreparable environmental damages and huge financial loss. Fault-tolerance based on temporal and spatial redundancy is required to solve the safety-requirements. Transient or intermittent faults are mitigated through temporal redundancy which is the transmission of two copies of the same message at different instants. In contrast, Frame Replication and Elimination for Reliability (FRER) is used to alleviate both transient and permanent failures. In FRER, every message is replicated and forwarded over one or more redundant routes. Several studies have been performed to implement fault-tolerant schedulers for Time-Triggered (TT) communication but most of these studies focus on certain faults (e.g. link failures or device crashes) to simplify the process.

The goal of this paper is to extend the time-triggered scheduler introduced in [4] to include optimization of system reliability while satisfying the real-time constraints of the application. The algorithm considers, i) redundant and non-redundant real-time jobs, ii) the redundancy in the platform model and the reliability of the TSN platform devices (e.g. end systems, switches, and links) and iii) TSN-based fault tolerance mechanisms such as FRER. We also introduce a new reliability technique for safety-critical systems. Our reliability model considers real-time jobs as system constituents and the relation between different jobs as control dependencies. This approach calculates the reliability of the overall system by first calculating the reliability of each message using the reliability of TSN-aware devices and then using it to compute the reliability of each safety-critical job. In addition to the control dependencies introduced in [4], this paper also introduces conditional

Manuscript received January 23, 2021; revised June 15, 2021.
doi:10.12720/jcm.16.7.250-258

transfers between different safety-critical jobs. It means that the overall system reliability is calculated using only those messages that are essential to the receiving job. This work only considers the permanent failures but FRER also supports transient failures. To implement FRER and to reduce network utilization, we assume that a message is duplicated on the junction of a disjoint path.

The remainder of this paper is structured as follows: Section II discusses the related work on fault-tolerant schedulers for real-time systems, Section III presents the problem and outlines the system and fault models, Section IV describes our reliability model and Section V discusses our fault-tolerant list scheduler. The results are presented in Section VI and the last section concludes this paper.

II. Related Work

In recent years, several algorithms have been proposed regarding the TAS scheduling problem. Authors in [5] and [6] use a fixed routing policy to synthesize the GCL of the TSN-aware devices. Most of these studies ignore the dependencies between the routing and scheduling constraints, providing only a sub-optimal solution. To address this problem, we have proposed a list scheduler in [4] that considers both the routing and scheduling constraints in a single step. There are other recent works that also consider joint routing and scheduling problem but unlike them, our work supports inter-flow dependencies and job scheduling. These features play an important role in the deployment of TSN-aware devices vital for modern cyber-physical systems. However, these scheduling algorithms assume that the safety-critical system remains completely fault-free during the execution of the application. This assumption is highly optimistic, since in practice the system may encounter any number of faults during its function.

In case of failures in the communication network, authors in [7] developed the greedy list scheduler for the fault-tolerant communication over the multi-bus heterogeneous systems. This scheduling strategy addresses the transient bus failures with the data fragmentation. Moreover, authors in [8] proposed a TT scheduler that masks multiple link failures through a localized fault-tolerant protocol rather than spatial redundancy that increases the bandwidth of non-critical traffic. The authors in [9] proposed a method that uses a CEGAR-based approach to find a (*k*,*l*)-resistant transmission schedule. In this approach, the algorithm delivers at least l copies of a message to the receiving application through k separate links. The mentioned works only consider the faults in the links while assuming that the system processors are fault-free.

For the failures in the system processors, [10] proposed a distributed list scheduler that computes the fault-tolerant schedules for the multi-processor real-time architecture via task redundancy. This work assumed that all tasks run over the same cycle. The authors in [11]-[14] use Primary-Backup (P/B) approach to address processor failures during the execution of the application. A contention-aware fault-tolerant (CAFT) scheduler was proposed in [15] that uses active replication to solve the occurrence of failures in multiple number of processors. Similarly, [16] compute a fault-tolerant schedule through active replication and re-execution of jobs. Both these algorithms use optimal task binding and resource allocation to hamper the need for any additional hardware resources. These works consider faults only in processors with the assumption that the links are completely fault-free. In [17] and [18], authors proposed algorithms that consider faults in both the system processors and the communication network but these works do not consider the timing constraints of the safety-critical systems or component redundancy to mitigate failures.

In this paper, we introduce a fault-tolerant TT scheduler that meets the temporal requirements of the hard real-time system while optimizing the overall system reliability. In contrast to the state-of-the-art, we consider both the processor and link failure while optimizing the overall system reliability. Moreover, we use a conditional graph with conditional precedence constraints between the safety-critical jobs that lead to a more efficient and realistic transmission schedule.

III. Problem Formulation

In this paper, we develop a fault-tolerant scheduler based on the scheduling and routing constraints defined in [4] with the purpose to increase the reliability of the safety-critical system. To achieve this goal, we use a novel reliability analysis technique to model and compute the reliability of the system. Following constraints defined before in [4] are also used in this work,

1. The scheduling is non-preemptive i.e. an end-system executes only one job at a time.
2. Each TT message is transmitted through a component only once to avoid loops.
3. Each TT message is transmitted through a link only if there is no hindrance in the transmission i.e. no other message is transmitted through the link during that duration.
4. Our model uses periodic TT messages and all the iterations of a message are transmitted through the same link.
5. A job starts its execution only when all the essential TT messages from the predecessor jobs are received.
6. Each TT message must reach the destination within the deadline of the receiver job.

A. System Model

Our fault-tolerant time-triggered model is defined through two distinct graphs, i.e. a conditional application graph and an architecture graph. The application graph represents the set of real-time jobs that are implemented on the system depicted by the architecture graph.

1) Application model

The system application is represented through a conditional directed acyclic graph $G_c =< J, E >$. In this graph each vertex $j_i \in J$ presents a non-divisible real-time

job and each conditional directed edge $e_{ik} \in E$ depicts the precedence constraint between jobs j_i and j_k where j_i is the parent of j_k. Each real-time job $j_i \in J$ is represented through a tuple $< w_i, r_i, d_i >$ where w_i is the worst case execution time, r_i is the probability of successful execution (reliability) and d_i is the dead-line for the execution of the job. Each TT message $e_{ik} \in E$ is defined through the tuple $< p_{ik}, t_{ik}, r_{mik}, i_{ik}, c_{ik} >$. Here p_{ik} is the time-period, t_{ik} is the transmission time and is r_{mik} the probability of successful transmission (reliability) of the message. i_{ik} represents the injection time or in other words the time at which sender job j_i starts the transmission of the message. Lastly c_{ik} is the conditional transfer, i.e. it determines if the message is essential for the receiver job j_k or if it can be replaced with a TT message from other parent jobs. This is an important characteristic as a job cannot start its execution before receiving all the required incoming messages. To simplify matters, we assume that a receiver job has at least two incoming conditional edges that can be substituted with each other for a successful execution. An example of such a graph is given in Fig. 1.

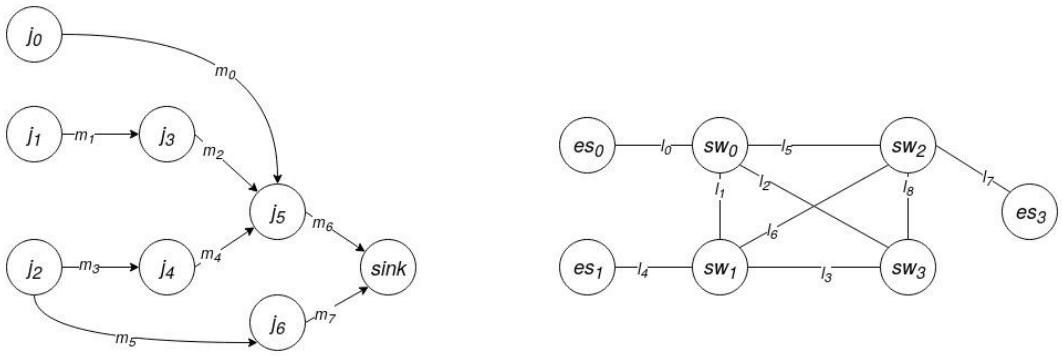

Fig. 1. An example of system model (the left graph is an application graph and the right one is an architecture graph)

2) Architecture model

The system architecture is depicted through a undirected graph $G_A =< S, L >$. In this graph, each vertex $s_i \in S$ is either a TSN end system or a TSN switch, i.e. $s_i = ES_i \cup SW_i$ whereas each edge $l_i \in L$ is a duplex physical link between TSN aware devices, i.e. TSN end system and TSN switch. A duplex link means that two messages can transmit on the link simultaneously provided that they traverse in opposite directions. Each network component $c_i \in S \cup L = G_A$ is defined by a reliability r_{ci} which is the probability of the components c_i to operate successfully. An example of the system architecture is given in Fig. 1. It has to be noted that a real-time jobs are allocated only on end systems and messages are scheduled on the TSN-aware switches that lie between the sending and receiving end systems.

3) Fault model

Our fault model considers only one permanent failure in any network component including TSN-aware devices and links, at a given time. It means that a failure in a link does not effect the operation of its connected devices. Due to limited number of resources, our solution only considers the duplication of TT messages at the junction of disjoint paths in the system. However, this method can easily be extended to tolerate several permanent failures in system components by including various redundant routes and multiple message replications at the expanse of a greater network load. In addition to redundant routes, our method also considers redundant real-time jobs through conditional edges as described in Section III-A.1. It means that if one of the redundant real-time jobs executes successfully then the failure of the other redundant jobs will not effect the operation of the safety-critical system.

IV. Reliability Model

The reliability of the safety-critical system is calculated using the reliability of the real-time jobs and the reliability of the TT messages. This section explains our reliability model in detail.

A. Reliablity of Safety-Critical Messages

The successful transmission of a TT message depends upon the reliability of all the components constituting the transmitting path. The reliability r_{ci} of a system component $c_i \in G_A$ can be expressed as [19],

$$r_{ci}(t) = e^{-\lambda t} \tag{1}$$

where λ is the failure rate that specifies the number of faults a device experiences per unit of time. Moreover, the total reliability of the TSN-aware devices connected in series with each other is given as [19],

$$R(t) = \prod_{i=1}^{N} r_{ci}(t) \tag{2}$$

Whereas the total reliability of a message transmitted on a path where TSN-aware devices are connected in parallel to each other is given as [19],

$$R(t) = 1 - \prod_{i=1}^{N} (1 - r_{ci}(t)) \tag{3}$$

As described in Section III.B, a TT message is duplicated at the junction of two parallel paths and each copy is transmitted over one or the other disjointed path. However, the device at the other end of the junction transmits only one copy and discards the other. To calculate the total reliability of a TT message, we model the redundant routes in the form of series and parallel components where each system component represents a

separate module. These modules are connected together through the specified route topology. A reliability model for the transmission of m_3 which is sent from es_0 to es_1 is given in Fig. 2. For simplicity, it is assumed that TSN end systems are fault-free. We also assume that all TSN switches are identical and have the same failure rate. The same assumption is made in case of the duplex links. Therefore, we denote the reliability of a switch and a link as r_{sw} and r_l respectively. Since the successful transmission of a message correlates with the reliability of the overall system so according to Eq. 2 and Eq. 3, the reliability of message m_3 in Fig. 2 is,

$$r_{m_3}(t) = r_l^2 r_{sw}^2 (r_l + r_l^2 r_{sw} - r_l^3 r_{sw}) \quad (4)$$

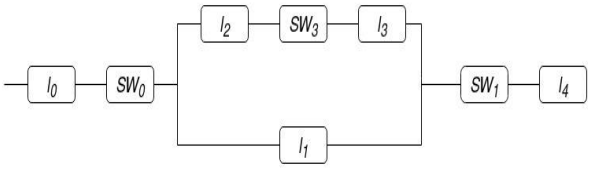

Fig. 2. Reliability model of message m_3

B. Reliability of Safety-Critical Jobs

The reliability of a job depends upon the successful execution of its predecessors, transmission of all the required TT messages and correct operation of the assigned end system. If a job has no predecessor then its reliability is equal to the reliability of the assigned end system. We depict the job reliability model in the form of series and parallel systems using the conditional transfer described in Section III.A.1. In this context, each job and the corresponding TT message is considered as a separate module. The modules are connected together through conditional control dependencies described in the application graph. An example of such a reliability model is given in Fig. 3 for job j_5. Here the conditional path shows that the TT messages from job j_4 are essential for the execution of j_5 but TT messages from either j_0 or j_3 are sufficient to start j_5.

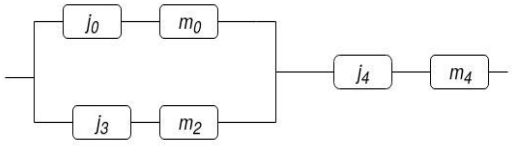

Fig. 3. Reliability model of job j_5

Eq. 2 and Eq. 3 can also be applied here to solve the series-parallel connection of the modules. Hence, according to Fig. 3 the total reliability of job j_5 can be given as,

$$r_5 = [1 - (1 - r_0 r_{m_0})(1 - r_3 r_{m_2})] r_4 r_{m_4} \quad (5)$$

Since $r_3 = r_{es} r_{m_1}$, $r_4 = r_{es} r_{m_3}$ and $r_0 = r_{es}$ therefore Eq. 5 can be simplified to Eq. 6. Here the reliability of the messages m_1, m_2, m_4 and m_6 are calculated through the method described in Section 4.1.

$$r_5 = (r_{m_0} + r_{m_1} r_{m_2} - r_{m_0} r_{m_1} r_{m_2}) r_{m_3} r_{m_4} \quad (6)$$

It has to be noted here that the described technique to calculate job reliability is only applicable when the respective job shares no module with its respective predecessors and incoming TT messages. In reality, it is quite possible that the predecessor jobs and incoming messages share one or more network modules. In such cases, it is not possible to use series/parallel module structure to calculate job reliability. Alternatively, the model is first expanded for each shared module and then the job reliability is calculated using different operational conditions and the total probability of the job completing its execution on the component.

$$r_i = r_{c_i} . Prob_{c_i} + (1 - r_{c_i}) . Prob_{\overline{c_i}} \quad (7)$$

where r_i is the reliability of the ith job, r_{c_i} is the reliability of the respective network module c_i , $Prob_{c_i}$ is the probability of the job completing its execution on a fault-free c_i and $Prob_{\overline{c_i}}$ is the probability of the job completing its execution on a faulty c_i. For better understanding, consider the reliability model given in Fig. 3 for the last part i.e. $r_{m_3} r_{m_4}$ of Eq. 6. The reliability model has four common components i.e. SW_1 , l_4 , SW_0 and l_1 . The common SW_1 and l_4 components are connected in series with each other therefore only one of each is considered for the reliability model. But the model is expanded for the other two components and Eq. 7 is used to calculate the reliability of this statement.

$$r_{s_4} = r_{sw_0} . Prob_{sw_0} + (1 - r_{sw_0}) . Prob_{\overline{sw_0}} \quad (8)$$

where $Prob_{sw_0}$ and $Prob_{\overline{sw_0}}$ are given as,

$$Prob_{sw_0} = r_{l_1} . Prob_{(sw_0, l_1)} + (1 - r_{l_1}) . Prob_{(sw_0, \overline{l_1})} \quad (9)$$

$$Prob_{\overline{sw_0}} = r_{l_1} . Prob_{(\overline{sw_0}, l_1)} + (1 - r_{l_1}) . Prob_{(\overline{sw_0}, \overline{l_1})} \quad (10)$$

Fig. [4c-4e] represents the expansion of the reliability model of $r_{m_3} r_{m_4}$ on SW_0 and l_1. The expansion continues until a model is obtained that has no common components and after that series-parallel structure is used to calculate the reliability of j_5. This approach is applied to every component that constitutes the reliability model of job j_5 until a model is achieved that has no common component.

C. Reliability of Safety-Critical System

The reliability of a safety-critical system depends upon the successful execution of all the mission critical jobs within

the system. In order to calculate the reliability, a dummy sink vertex is added to the system application graph such that all jobs that have no successors are connected to this vertex as predecessors. In Fig. III.A.1, a sink vertex is added as a successor to jobs j_5 and j_6. Since the sink vertex is a dummy node without any impact on

the application, it is assumed that the reliability of all the TT messages transmitted to the sink vertex is always equal to one. Moreover, the overall system reliability is equal to the reliability of the sink vertex of the application graph. So the reliability of the system given in Fig. 1 is,

$$r_t = r_{sink} = r_5 r_6 \quad \text{since} \quad r_{m_8} = r_{m_9} = 1 \tag{11}$$

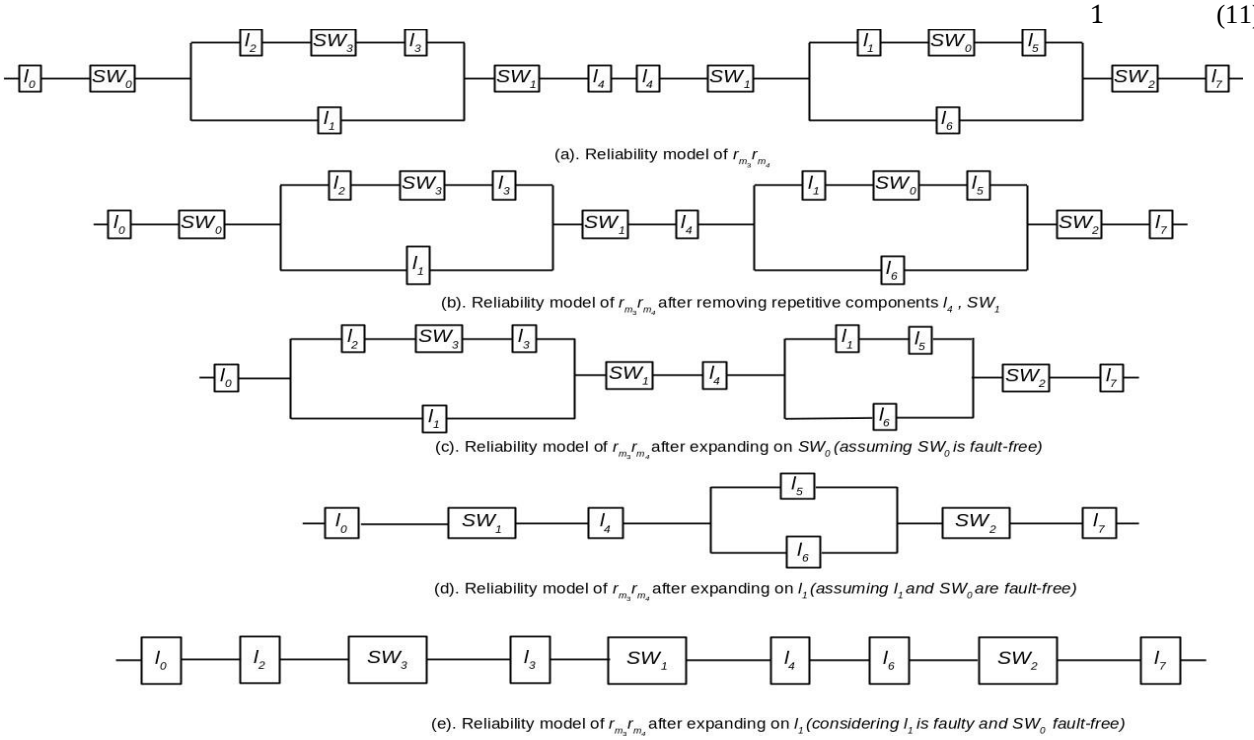

Fig. 4. Reliability model of $r_{m_3} r_{m_4}$ when it is expanded on common network components

V. Fault-Tolerant List Scheduler

We modify the list scheduler presented in [4] so that it fulfills the temporal requirements of the mission-critical applications while increasing the overall reliability of the system. The list scheduler works in two phases [20], firstly priorities are assigned to the jobs in the application graph and then these jobs are scheduled onto end systems based on their priority and precedence constraints. We used the critical path method to calculate the priorities of the jobs. The details of the priority assignments are given in [4]. Once the priorities are assigned, the scheduler adds the jobs whose precedence constraints have been fulfilled to a ready list. The list is then arranged in descending order of the priorities of the jobs. The scheduler then selects the jobs from this list and assigns them to an eligible end system (*J.CanRunOn*) one by one. Once an end system has been selected, the scheduler finds all possible routes between the sender and receiver end systems using the multiplication adjacency matrix. Then permutation is used to calculate all the possible redundant routes between the sender and receiver. For each pair, the message reliability is calculated and the pair of forwarding routes with the best reliability values are selected. The scheduler then finds the earliest injection time of each job's ingress flow. After that it obtains the message's arrival time (a_{ik}) based on the transmission delay of the message ($e2e_{ik}$). The details of the transmission delay calculation are given in [4]. If the chosen time slot does not meet the job's deadline, then another end system is selected. Algorithm 1 gives a pseudo-code representation of the fault-tolerant list scheduler.

Algorithm 1 Fault-tolerant List Scheduler

```
1: procedure FAULTTOLERANTLISTSCHEDULER
2:     Assign priority to each job
3:     J_sorted ← sort jobs descending based on priorities
4:     ∀j ∈ J_sorted is not scheduled:
5:     Scheduler(j)
6:     SystemReliability ← calculate the system reliability
7:     return SystemReliability
8: procedure SCHEDULER(Job j)
9:     if unscheduled job j has incoming flow then
10:        ∀e_ik ∈ E_TT.incoming: Scheduler(j_i)
11:        is_pre_job_schedule ← true
12:    else if is_pre_job_schedule or job j has no child then
13:        for p ∈ j.CanRunOn do
14:            for e_ik ∈ E_TT.incoming do
15:                RedPathe_ik ← FindMostReliableRedPaths
16:                i_ik ← FindEarliestInjectTime
17:                a_ik ← i_ik + e2e_ik
18:                if a_ik > d_k then go to next end-system
19:                r ← CalculateJobReliability
20:                r_m_ik ← CalculateMessageReliability
21:    return
```

VI. Experiments and Evaluation

Our fault-tolerant list scheduler was implemented in C++ and the experiments were carried out on a T460 Linux ThinkPad with 32GB memory and Intel i5 processor. The system model used for the experiments were generated using the SNAP library [21]. We have five

separate test cases with different properties that are defined in Table I. The characteristics used for these experiments are as follows:

a) System Architecture: For the architecture graph, we use the grid network with different number of components as defined in Table I. Fig. 5 gives an example of an architecture that is used for the use case 1. The use case 1 represents the small size network while the use case 2 and 3 represent medium size and large scale networks respectively.

b) System Application: Each safety-critical application is generated in the form of a Random Forest Fire Directed graph using the SNAP library [21]. The number of jobs in the safety-critical application varies between 8 and 16 as shown in Table I. The period of the TT flows is chosen from the set <50,100,150> (ms) and the transmission time of a TT frame from one device to the neighboring device is taken as 10 μs. To simplify things, we make the assumption that all jobs have the same worst-case execution time i.e. 80 μs and deadline i.e. 1.75 ms.

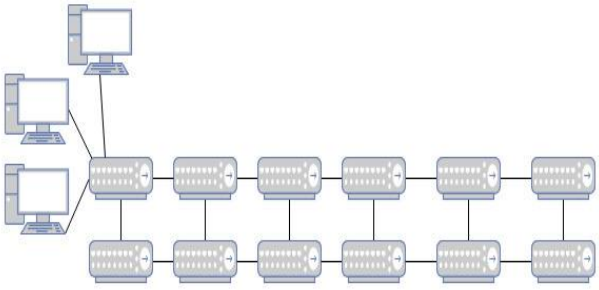

Fig. 5. Grid network topology used for case 1

TABLE I: THE CHARACTERISTICS OF THE EXPERIMENTAL USE CASE

Use Case	1	2	3	4	5
Number of nodes	71	83	95	47	71
Number of links	76	89	102	50	76
Number of jobs	12	14	16	8	12
Number of messages	30	60	90	20	50
Link/Device reliability distribution		.99:10% .991:10% … .999:10%		.99:25% .993:25% … .999:25%	

A. Results

We have divided our results in two separate scenarios. The first scenario comprises of the first three use cases given in Table I. It focuses on comparing the average system reliability and average network load between the schedules generated by the list scheduler given in [4] which computes schedule with optimized makespan for the fault-free TSN systems and our fault-tolerant list scheduler. We compare these factors with respect to varying number of messages, devices and links. In the second scenario, we study the trend of system reliability with varied level of job redundancy and component reliability. Use case 4 and 5 in Table 1 give the properties of this scenario. For each mentioned use case, we generate 100 synthetic system models that have different inter-flow dependencies and use the grid network topology.

1) List Scheduler vs. Fault-tolerant list scheduler

In the first part of the experiments, we study the effect of the basic list scheduler and fault-tolerant list scheduler on the average system reliability and average network load respectively. This part uses the properties of the first three use cases given in Table I. We assume that all the TT messages are essential for the execution of successor jobs and that the reliability of each network component remains constant throughout the execution of the application. The characteristics of these models, i.e. number of system components etc, are given in Table I. The reliability of the devices, in each case, varies between 0.99 and 0.999 by a probability of 10%.

For the use case 1 given in Table I, Fig. 6a shows the average system reliability of 100 synthetic models for the basic list scheduler and Fault-Tolerant List Scheduler (FTLS). The figure shows that the average system reliability is improved by approximately 9% after the usage of the fault-tolerant list scheduler. This improvement is carried out through message duplication and transmission of messages on redundant paths in the fault-tolerant list scheduler compared to the single copy of each TT message scheduled over only one path in the basic list scheduler. This improvement in system reliability, however, has few drawbacks. Message duplication leads to a higher number of scheduled messages by FTLS that increases the overall network load. This is proven through the results given in Fig. 6b where the network load is approximately 2 % higher in FTLS compared to the basic list scheduler. Therefore, FTLS increases the system reliability at the expanse of the network load. The increased number of messages in FTLS also increases the overall makespan of the schedule. Fig. 6c shows that, for use case 1, the average makespan for the schedule obtained through the basic list scheduler is 0.943 ms whereas it is 1.055 ms in case of FTLS. So, the basic list scheduler provides a more optimal solution than FTLS but at the expanse of lower system reliability.

For use case 2, our fault-tolerant scheduler improves the system reliability approximately by 7% at the expanse of a 2% increase in the network load as shown in Fig. 6a, 6b. The average makespan for FTLS is 1.498 ms compared to 1.325 ms obtained through the basic list scheduler as shown in Fig. 6c. Similarly, for use case 3, FTLS improves the system reliability by approximately 4% at an increased network load of 2% (see Fig. 6a, 6b). The average makespan for basic list scheduler is 1.630 ms compared to the increased makespan of 1.851 ms in FTLS.

TABLE II: AVERAGE EXECUTION TIME OF LS AND FTLS FOR USE CASE 1-3

Scheduler	use case 1 (s)	use case 2 (s)	use case 3 (s)
LS	7	20.2	58.7
FTLS	8	35	625.5

The last factor to consider in all three cases is the schedulability ratio that, theoretically, should decrease in

FTLS because of the message duplication compared to the basic list scheduler. However, in use case 1 the schedulability ratio remains same for both schedulers mainly because the total number of messages is low compared to the number of devices and links. So the duplication of messages does not affect the chances of obtaining an optimal schedule. This is shown in Fig. 6d. When the number of messages are increased to 60 in use case 2 without a considerable increase in the number of devices and links, the schedulability ratio decreases by 4% as shown in Fig. 6d. Use case 3, however, shows a significant drop in the schedulability ratio from 92 in the basic list scheduler to 24 in FTLS. Since the number of messages, in this case, is thrice more than in use case 1 so the total number of message duplication is also increased approximately three times but there is not enough increase in the number of devices and links to cater this change. Therefore, there is a lower chance of getting an optimal schedule in this case.

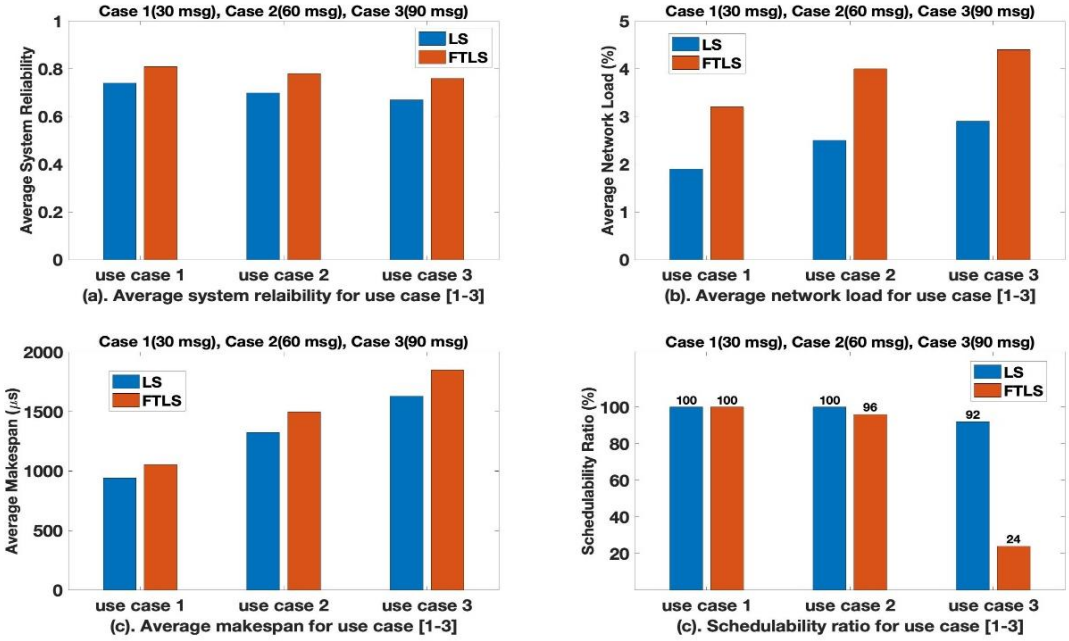

Fig. 6. Experimental results derived from the schedules generated by FTLS and basic LS for different number of nodes, links, and TT messages

Table II shows the average execution time of FTLS compared to the basic list scheduler. It shows that the execution time of FTLS dramatically increases with each use case. It is mainly because FTLS needs to explore a bigger search space to find valid fault-tolerant schedules compared to the basic list scheduler.

2) System Reliability vs. Varied level of job redundancy

In the second part of the experiments, we study the impact of a varying degree of redundancy for different jobs of the application graph on the system reliability. The characteristics for this set of experiments is given in use case 4 of Table I. In the first scenario of the use case, we assume that the jobs in the application graph require all the TT messages from their predecessors to start their execution. In the second scenario, the condition of the control transfer values alternates between two feasible conditions, i.e. essential and substitutable, with a probability of 0.5. In both scenarios, it is assumed that the reliability of the network devices (i.e. end systems and switches) varies from 0.993 to 0.999 while the links are fault-free. Fig. 7 shows the results of these experiments. It is visible from the results that the system reliability is greatly improved in scenarios where only half of the TT streams are essential. A higher level of redundancy between different jobs of the system application results in a more reliable TT communication schedule.

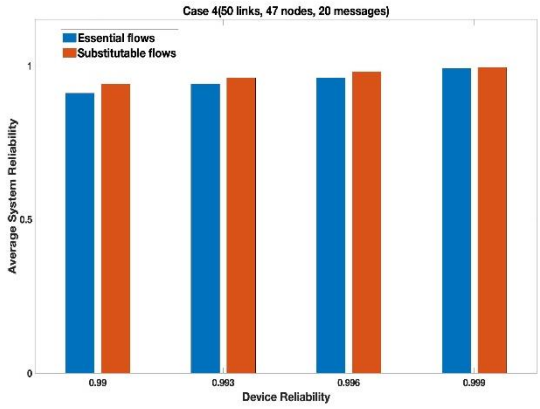

Fig. 7. Average of system reliability of schedules computed by FTLS for varying level of job redundancy

3) System Reliability vs. Varied component reliability

In the last part of the experiments, we study the sensitivity of overall system reliability with respect to different network components. Use case 5 in Table I represents the characteristics for these set of experiments. For this purpose, in the first scenario, we assume that the

link is fault free (i.e. r_l =1) whereas reliability of end systems and switches varies from 0.993 to 0.999. In the second scenario reliability of the links varies from 0.993 to 0.999 and r_{es}=r_{sw}=1. The results for these experiments are shown in Fig. 8. From the figure, it is visible that the average system reliability is increased by 7 % in case of a greater link reliability compared to other TSN-aware devices. This implies that more reliable physical links lead to a more fault-tolerant schedule.

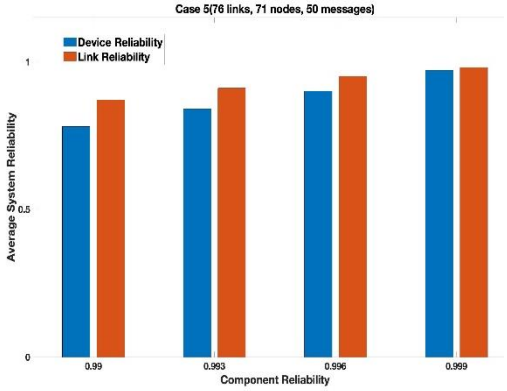

Fig. 8. Average of system reliability of schedules computed by FTLS for varying component reliability

VII. Conclusion

This paper presents a fault-tolerant list scheduler that schedules safety-critical applications while maximizing system reliability. The process supports the TSN redundancy management of message duplication and the elimination of message replicas. Our reliability model calculates system reliability by considering the reliability of each TSN-aware device and physical link that form the message forwarding paths. Our approach differs from other state-of-the-art fault-tolerant schedulers that only focus on either the device or the link failures. We also consider the conditional control transfer between different jobs, which raises the reliability of the overall system. The results illustrate that the fault-tolerant list scheduler enhances system reliability considerably compared to the basic list scheduler at the expense of an increase in the network load. Our results show that overall system reliability is highly improved by increasing job redundancy and enhancing the reliability of the physical links in the system. Our fault-tolerant scheduler improves the overall system reliability significantly compared to the basic list scheduler, but it requires a longer time to coverage to a feasible solution. In future work, we will look into new optimization algorithms that diminish this issue.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

Maryam Pahlevan designed the reliability model, developed the scheduler and conducted the research; she also evaluated the empirical results; Sarah Amin wrote the manuscript; Prof. Roman Obermaisser supervised the finding of this work and contributed to design and validation of the scheduling solution; all authors had approved the final version.

Acknowledgement

This work was supported by the DFG research grant ADISTES OB384/6-1.

References

[1] Institute of Electrical and Electronics Engineers, Time-Sensitive Networking. (2017). In *Time-Sensitive Networking Task Group.* [Online]. Available: http://www.ieee802.org/1/pages/tsn.html

[2] Institute of electrical and electronics engineers, inc. 802.1as-rev - timing and synchronization for time-sensitive applications. (2017). In *Time-Sensitive Networking Task Group.* [Online]. Available: http://www.ieee802.org/1/pages/802.1AS-rev.html

[3] Institute of electrical and electronics engineers, inc. 802.1qbv-enhancements for scheduled traffic. (2016). In *Time-Sensitive Networking Task Group.* [Online]. Available: http://www.ieee802.org/1/pages/802.1bv.html

[4] M. Pahlevan, N. Tabassam, and R. Obermaisser, "Heuristic list scheduler for time triggered traffic in time sensitive networks," in *Proc. 16th International Workshop on Real-Time Networks (RTN)*. ACM, 2018.

[5] S. S. Craciunas, *et al.*, "Scheduling real-time communication in IEEE 802.1 qbv time sensitive networks," in *Proc. 24th International Conference on Real-Time Networks and Systems*. ACM, 2016, pp. 183–192.

[6] P. Pop, M. L. Raagaard, S. S. Craciunas, and W. Steiner, "Design optimisation of cyber-physical distributed systems using ieee time-sensitive networks," *IET Cyber-Physical Systems: Theory & Applications*, vol. 1, no. 1, pp. 86–94, 2016.

[7] C. Arar, H. Kalla, S. Kalla, and H. Riadh, "A reliable fault-tolerant scheduling algorithm for real time embedded systems," 2013.

[8] L. Su, H. Wan, Y. Qin, X. Zhao, Y. Gao, X. Song, C. Lu, and M. Gu, "Synthesizing fault-tolerant schedule for time-triggered network without hot backup," *IEEE Transactions on Industrial Electronics*, vol. 66, no. 2, pp. 1345–1355, 2018.

[9] G. Avni, S. Guha, and G. Rodriguez-Navas, "Synthesizing time-triggered schedules for switched networks with faulty links," in *Proc. International Conference on Embedded Software (EMSOFT)*, 2016, pp. 1–10.

[10] M. C. E. Hugue and P. D. Stotts, "Guaranteed task deadlines for fault-tolerant workloads with conditional branches," *Real-Time Systems*, vol. 3, no. 3, pp. 275–305, 1991.

[11] C. Dima, A. Girault, C. Lavarenne, and Y. Sorel, "Off-line real-time fault-tolerant scheduling," in *Proc. Ninth Euromicro Workshop on Parallel and Distributed Processing*, 2001, pp. 410–417.

[12] D. Mosse, R. Melhem, and S. Ghosh, "Analysis of a fault-tolerant multiprocessor scheduling algorithm," in *Proc. IEEE 24th International Symposium on Fault-Tolerant Computing*, 1994, pp. 16–25.
[13] H. Lee, J. Kim, and S. J. Hong, "Evaluation of two load-balancing primary-backup process allocation schemes," *IEICE Transactions on Information and Systems*, vol. 82, no. 12, pp. 1535–1544, 1999.
[14] R. Al-Omari, G. Manimaran, and A. K. Somani, "A fault–tolerant dynamic scheduling algorithm for multiprocessor real–time systems," in *Proc. Fault-tolerant Computing Symp. (FTCS), FAST ABSTRACTS*, 1999, pp. 63–64.
[15] A. Benoit, M. Hakem, and Y. Robert, "Contention awareness and fault-tolerant scheduling for precedence constrained tasks in heterogeneous systems," *Parallel Computing*, vol. 35, no. 2, pp. 83–108, 2009.
[16] V. Izosimov, P. Pop, P. Eles, and Z. Peng, "Design optimization of time-and cost-constrained fault-tolerant distributed embedded systems," in *Proc. Design, Automation and Test in Europe*, 2005, pp. 864–869.
[17] S. M. Shatz, J. P. Wang, and M. Goto, "Task allocation for maximizing reliability of distributed computer systems," *IEEE Transactions on Computers*, vol. 41, no. 9, pp. 1156–1168, 1992.
[18] A. Dogan and F. Ozguner, "Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 308–323, 2002.
[19] I. Koren and C. M. Krishna, *Fault-tolerant Systems*, Elsevier, 2010.
[20] O. Sinnen, *Task Scheduling for Parallel Systems*, vol. 60. John Wiley & Sons, 2007.
[21] Snap library 4.0, user reference documentation. [Online]. Available: https://snap.stanford.edu/snap/doc/snapuser-ref/index.html

Maryam Pahlevan received the Master degree in communication engineering - networking technology from the Aalto University, Helsinki, Finland, in 2013. She is currently a Ph.D. student at the Embedded Systems chair, Department of Elektrotechnik and Informatik, University of Siegen. Her research interests include the communication infrastructures of integrated networked systems, and time-triggered communication protocols (e.g., TTEthernet or TSN). She was involved in the European research project called SAFE4RAIL which was conducted by 11 academic and industrial partners.

Sarah Amin received the Master degree in electronic communications and computer engineering from the University of Nottingham, Nottingham, United Kingdom, in 2014. She is currently a Ph.D. student at the Embedded Systems chair, Department of Elektrotechnik and Informatik, University of Siegen. Her research interests include multicore/manycore scheduling, embedded systems, safety-critical systems and incremental system design. She was involved in the German Government Funded (DFG) research project ADISTES.

Roman Obermaisser received the Ph.D. degree in computer science from the Vienna University of Technology, Vienna, Austria, in 2004. He is currently a Professor at the Embedded Systems chair, Department of Elektrotechnik and Informatik, University of Siegen. His research interests include cyber-physical system architectures, communication infrastructures of advance integrated systems, and time-triggered communication protocols (e.g., TTEthernet or TSN). He was involved in many European research projects like DREAMS, SAFE4RAIL and SAFEPOWER.