Online Rerouting and Rescheduling of Time-Triggered Flows for Fault Tolerance in Time-Sensitive Networking

Zhiwei Feng, *Member, IEEE*, Zonghua Gu, *Senior Member, IEEE*, Haichuan Yu, Qingxu Deng, *Member, IEEE*, and Linwei Niu, *Member, IEEE*

***Abstract*—Time-sensitive networking (TSN) is an industry-standard networking protocol that is widely deployed in safety-critical industrial and automotive networks thanks to its quality-of-service (QoS) mechanisms, esp. deterministic transmission and bounded end-to-end delay for time-triggered (TT) flows. In this article, we focus on TT flows and address the issue of fault tolerance against permanent and transient faults with both spatial and temporal redundancy. We present an efficient heuristic algorithm for online incremental rerouting and rescheduling of disrupted flows, assuming the paths and schedules of existing flows stay fixed. It is complementary to and can be combined with offline routing and scheduling algorithms for achieving fault tolerance based on frame replication and elimination for reliability (FRER) (IEEE 802.1CB). Performance evaluation shows that our approach is able to better recover the system's degree of redundancy (DoR) and has a higher acceptance rate than related work.**

***Index Terms*—Fault tolerance, online recovery, time-sensitive networking (TSN).**

I. Introduction

DETERMINISTIC real-time communication is a crucial requirement in modern embedded and cyber–physical systems, e.g., safety-critical networks in automotive and industrial automation applications. Time-sensitive networking (TSN) [1], [2], [3] specifies two main types of traffic shapers to regulate traffic arrival and transmission in the network, including the IEEE 802.1 Qbv time-aware shaper (TAS) for time-triggered (TT) traffic, a TT shaper where all network nodes have a synchronized clock, and transmission decisions are based on precomputed time schedules. TAS aims to provide deterministic transmission service and bounded end-to-end delay for TT traffic; and the IEEE 802.1Qav credit-based shaper (CBS) for audio video bridging (AVB) traffic, an event-triggered shaper that does not rely on the synchronized clock. The other traffic class is best effort (BE), with no timing guarantees. A flow is defined as a sequence of frames (packets) between a source end system (ES) and a destination ES, traversing one or more switches in-between. In this article, we focus on TAS for TT flows, which are typically safety-critical and have hard real-time constraints.

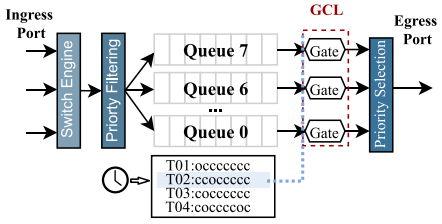


Fig. 1. Architecture of an IEEE 802.1Qbv switch with an example GCL.

We consider multihop TSN switched Ethernet over full-duplex physical links, consisting of ordered operations to open or close transmission gates for each port. Fig. 1 shows the architecture of an IEEE 802.1Qbv-enabled switch, which consists of the following major components [4].

1) *Scheduled FIFO Queues:* There are eight independent FIFO queues controlled by transmission gates. The incoming traffic is filtered by the packet filtering unit, which sends a packet to its designated queue. This information is encoded as the class of service (CoS) in the priority code point (PCP) header in the Ethernet frame.
2) *Gate Control List (GCL):* TT flows are periodic and shaped by the TAS on egression ports. TAS synchronously grants transmission to traffic queues based on a predefined schedule in GCLs, which can periodically trigger gate-open and gate-close events with a gate control cycle. Depending on the specific implementation, the time granularity between events can be as low as 1 ns. The schedule is located in a GCL look-up table that is distributively configured to each TSN node. For

Manuscript received 2 August 2022; accepted 2 August 2022. Date of publication 9 August 2022; date of current version 24 October 2022. This work was supported in part by the National Natural Science Foundation of China under Grant 62072085 and Grant 61903356; in part by the Liaoning Revitalization Talents Program under Grant XLYC1902017; and in part by the National Science Foundation of U.S. under Grant HRD-2135345. This article was presented at the International Conference on Embedded Software (EMSOFT) 2022 and appeared as part of the ESWEEK-TCAD special issue. This article was recommended by Associate Editor A. K. Coskun. *(Corresponding author: Qingxu Deng.)*

Zhiwei Feng, Haichuan Yu, and Qingxu Deng are with the School of Computer Science and Engineering, Northeastern University, Shenyang 110819, China (e-mail: fengzw@mail.neu.edu.cn; iamyhc@live.cn; dengqx@mail.neu.edu.cn).

Zonghua Gu is with the Department of Applied Physics and Electronics, Umeå University, 90187 Umeå, Sweden (e-mail: zonghua.gu@umu.se).

Linwei Niu is with the Department of Electrical Engineering and Computer Science, Howard University, Washington, DC 20059 USA (e-mail: linwei.niu@howard.edu).

Digital Object Identifier 10.1109/TCAD.2022.3197523

example, the GCL in Fig. 1 specifies that at time T01, the gate for Queue 7 is open, and the gates for Queues 0–6 are closed; at time T02, the gate for Queue 5 is open, and all other gates are closed, and so on.

3) *Time Synchronization:* To allow TT transmission that is distributed through the network, a timer is globally synchronized with all the switches in the same network using precision time protocols (PTPs), e.g., IEEE 802.1AS [5].

Two types of faults may compromise the reliability of TT flows: 1) *Permanent faults* may cause link or switch failure and disrupt the transmission service and 2) *Transient faults* such as packet losses or bit-flips caused by electromagnetic interference (EMI) may compromise the message transmission at the time of fault occurrence but do not affect subsequent messages. For fault tolerance, TSN supports seamless redundancy with frame replication and elimination for reliability (FRER) (IEEE 802.1CB) [6]. According to FRER, multiple redundant paths (routes), defined as multiple paths that do not share any common switches, are allocated for each TT flow. Frames are replicated at the source and transmitted through separate paths to the destination, and duplicates are eliminated at destinations. While effective against single-point failures, FRER introduces significant overheads since it requires at least twice the bandwidth to transmit the redundant packets for fault tolerance.

In this article, we extend the static offline approach of FRER to have online incremental rerouting and rescheduling. We handle both permanent and transient faults by combining spatial and temporal redundancy to transmit redundant message copies either on disjoint paths or on the same path. Our approach consists of an offline phase and an online phase. In the offline phase, for each flow, we build its set of redundant paths based on FRER and its candidate redundant paths used during online rerouting. In the online phase, upon detection of a link or switch failure due to a permanent fault, we perform online incremental rerouting and rescheduling to find another redundant path and offset assignment (schedule) for the disrupted flow while keeping the paths and offset assignments of existing flows fixed and cannot be modified. This unique assumption brings new challenges in terms of worst-case delay (WCD) analysis and offsets assignment, and we propose novel algorithms to address these challenges.

This article is structured as follows. We discuss related works in Section II; present our system model and definitions in Section III; present our approach in Section IV; present performance evaluation results in Section V, and conclude this article in Section VI.

II. Related Work

A. Flow Interleaving and Flow Isolation

In this section, we focus on the frame interleaving, which is one of the guarantees for deterministic communication behavior of TT flows presented in [7]. Fig. 2 illustrates the *flow interleaving* problem, and two approaches to avoiding it as presented in [7]. Two flows f_1 and f_2 arrive at switch v_b from

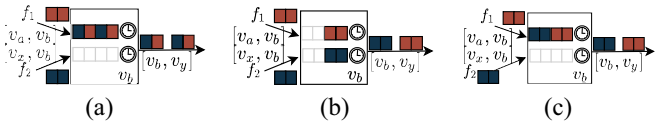


Fig. 2. Flow interleaving and flow isolation within an egress port of a switch. The arrows denote message transmission paths, not time, e.g., the blue frames arrive earlier than the red frames in (b) and (c). (a) Flow interleaving in the same egress queue. (b) Flow isolation in different egress queues. (c) Flow isolation with nonoverlapping time slots.

different links $[v_a, v_b]$ and $[v_x, v_b]$ at roughly the same time and are forwarded via the same egress port. Within the switch, they may be assigned to the same queue or two different queues. Due to the finite precision of time synchronization, the order in which the individual frames arrive and are placed in the queues at runtime is nondeterministic. Both flows are assigned to the same queue in Fig. 2(a). If the timed gate of the queue is opened first for the time interval needed to transmit two frames and sometime later for another two frames, then any combination of the respective frames of both flows may occur on the egress port. Hence, different flow instances in different periods may experience different delays through the switch, causing high jitters in each flow's delay at the switch. In order to meet a flow's end-to-end deadline, the WCD introduced on each egress port along each path should be added together, and the high jitters caused by flow interleaving are not desirable in hard real-time systems. Additionally, the nondeterminism caused by flow interleaving breaks the temporal isolation among different flows since any change in one flow may affect other flows' delays and jitters.

We formally define the concept of conflicting flows.

Definition 1 (Conflicting Flows): Two flows are conflicting flows if they traverse the same link $[v_a, v_b]$, or if they traverse two different links $[v_a, v_b]$ and $[v_x, v_b]$ to reach the same switch v_b, and are assigned to the same queue within v_b.

Craciunas *et al.* [7] discussed two solutions to avoid the flow interleaving problem by either placing the conflicting flows in different queues or isolating them on the time axis. In Fig. 2(b), the two flows arriving during interfering intervals are placed in different queues, and each frame is dispatched deterministically according to each queue's associated timed gate. But this is not always feasible since each switch only has eight queues, typically much smaller than the number of flows that go through each switch. In Fig. 2(c), the two flows are assigned to the same queue in switch v_b, so they must be isolated on the time axis to enforce deterministic order of transmissions in nonoverlapping time slots, i.e., one flow instance must be transmitted completely before the other flow instance is put into the queue. Fig. 2(c) shows that f_1 is transmitted before f_2, but it is also possible for f_2 to be transmitted before f_1. This leads to an important constraint in incremental scheduling for (assigning offsets to) a flow f_1 along a given path, i.e., f_1's transmission duration must fit within an idle window of another conflicting flow. Otherwise, f_1 may be delayed or *blocked* by idle windows of other conflicting flows that are not large enough to fit it.

TABLE I
SUMMARY OF RELATED WORKS ON TSN FAULT TOLERANCE. ABBREVIATIONS: ALO: ANT LION OPTIMIZATION, GAS: GENETIC ALGORITHMS, CP: CONSTRAINT-PROGRAMMING, GRASP: GREEDY RANDOMIZED ADAPTIVE SEARCH PROCEDURE, ILP: INTEGER LINEAR PROGRAMMING, SA: SIMULATED ANNEALING, SMT: SATISFIABILITY MODULO THEORY, AND UBS: URGENCY-BASED SCHEDULER

Work	Traffic Class	Problem Formulation	Algorithm	Offline/Online	Redundancy Type
Avni *et al.* (2016) [8]	TT	Routing + Scheduling	SMT	Offline	Spatial
Gavrilut *et al.* (2017) [9]	UBS	Topology selection + Routing	Heuristic, GRASP, CP	Offline	Spatial
Atallah *et al.* (2018) [10]	TT	Topology selection + Routing + Scheduling	Heuristic	Offline	Spatial
Pahlevan *et al.* (2021) [11]	TT	Routing + Scheduling	Heuristic	Offline	Spatial
Syed *et al.* (2021) [12]	TT	Routing + Scheduling	Heuristic	Offline	Spatial
Huang *et al.* (2021) [13]	TT	Routing + Scheduling	ALO	Offline	Spatial
Reusch *et al.* (2021) [14]	TT	Routing + Scheduling	CP, SA	Offline	Spatial
Dobrin *et al.* (2019) [15]	TT + BE	Scheduling	ILP	Offline	Temporal (ARQ)
Park *et al.* (2019) [16]	TT	Routing + Scheduling	GA	Offline	Temporal (ARQ)
Feng *et al.* (2022) [17]	TT + BE	Routing + Scheduling	Dijkstra, SMT	Offline	Temporal (ARQ)
Álvarez *et al.* (2017-21) [18]–[20]	TT	Implementation	None	Offline	Temporal (Proactive)
Atallah *et al.* (2018) [21]	AVB	Routing + Scheduling	ILP	Offline	Temporal (Proactive)
Feng *et al.* (2021) [22]	TT + BE	Routing + Scheduling	Dijkstra, SMT	Offline	Temporal (Proactive)
Zhou *et al.* (2021) [23]	TT	Routing + Scheduling	SMT, Heuristic	Offline	Spatial + Temporal (Proactive)
Pozo *et al.* (2018) [24]	TT	Routing + Scheduling	ILP, Heuristic	Offline + Online	Temporal (Retransmission)
Kong *et al.* (2021) [25]	TT	Routing + Scheduling	ILP, Heuristic	Offline + Online	Spatial
Our work	TT	Routing + Scheduling	Heuristic	Offline + Online	Spatial + Temporal (Proactive)

B. Related Works on TSN Fault Tolerance

Table I summarizes and compares related works on TSN fault tolerance. We can distinguish between two types of redundancy for achieving fault tolerance in TSN: 1) *spatial redundancy* through routing each flow through multiple disjoint redundant paths (i.e., FRER) can be used to tolerate both permanent and transient faults and 2) *temporal redundancy* through sending redundant copies of frames/messages of a flow on the same path can be used to tolerate transient faults only. Another axis to classify related works is offline, online, or both. Next, we provide detailed descriptions of each work.

1) Offline Design Based on Spatial Redundancy: Avni *et al.* [8] addressed the problem of finding a (k, l)-resistant schedule, which guarantees that at least l copies of a message arrive at their destination by the timeout with at most k link failures. Gavrilut *et al.* [9] presented algorithms for joint topology selection and routing synthesis to achieve fault tolerance of the urgency-based scheduler (UBS) traffic type. Atallah *et al.* [10] presented a heuristic algorithm for joint topology selection, routing, and scheduling to achieve fault tolerance of TT flows. Pahlevan *et al.* [11] presented a heuristic list scheduling algorithm for fault-tolerant routing and scheduling of TT flows to maximize system reliability. Syed *et al.* [12] presented four different dynamic routing and scheduling heuristic algorithms based on vector bin-packing to support FRER in a TSN-based in-vehicle network. Huang *et al.* [13] presented a meta-heuristic ant lion optimization (ALO)-based algorithm for fault-tolerant routing and scheduling of TT flows. Reusch *et al.* [14] presented constraint-programming (CP) and simulated annealing (SA)-based algorithms for routing and scheduling of TT flows to satisfy both fault tolerance (with FRER) and security (with the Tend-system authentication protocol) requirements.

2) Offline Design Based on Temporal Redundancy: With standard TSN, the talker end-system sends only one copy of each message on each redundant path. To provide temporal redundancy for tolerating transient faults, higher-level protocols such as automatic repeat request (ARQ) may be used, which relies on ACK/NACK messages and/or timeouts to trigger the retransmission of lost frames. Dobrin *et al.* [15] generated an offline schedule for TT flows by using the integer linear programming (ILP) solver and provisioning for the specified number of retransmissions upon faults, and perform priority assignment for rate-constrained (AVB) flows to satisfy timing and fault tolerance requirements. Park *et al.* [16] presented a genetic algorithm (GA)-based optimization framework for routing and scheduling of TT flows to maximize system reliability, assuming that end-systems support the ARQ protocol and switches support frame preemption. Feng *et al.* [17] proposed an offline reservation-based fault-tolerant scheduling algorithm for IEEE 802.1 Qbv. The satisfiability modulo theory (SMT) solver generates resource allocation for an alternative message according to the constraints established to ensure deterministic transmission.

To address the shortcomings of ARQ, i.e., increased delay and jitter caused by frame retransmission and loss of ACK/NACK messages, Álvarez *et al.* [18], [19], [20] proposed proactive transmission of replicated frames (PTRFs), which works by transmitting several copies (replicas) of each frame in a preventive manner, to increase the chances of each frame reaching its destination. If more than one replica of the same frame reaches the listener end-system successfully, then the listener end-system delivers only the first-received copy to the application and discards the rest. They designed three different variants of PTRF.

1) End-to-end estimation and replication, where only the talker end-system replicates frames, only the listener end-system eliminates surplus replicas, and the switches

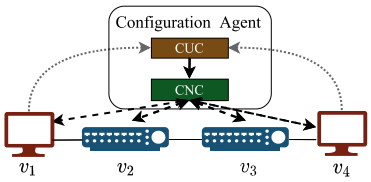


Fig. 3. CA consisting of CUC and CNC.

are standard TSN switches that simply forward every received frame. The number of replicas is calculated to achieve a given end-to-end reliability value.

2) End-to-end estimation and link-based replication, where both the talker end-system and switches replicate frames, and both switches and the listener end-system eliminate surplus replicas.
3) Link-based estimation and replication, where the talker end-system and each switch may produce a different number of replicas, and the number of replicas is calculated separately for each link. Our work is motivated by PTRF and adopts the first approach of PTRF, i.e., end-to-end replication (assuming the number of replicas is given), to provide temporal redundancy for tolerating transient faults. On top of PTRF, we also propose an integration of PTRF with the IEEE Std 802.1CB standard (FRER) for space redundancy and efficient runtime online reconfiguration.

Atallah *et al.* [21] presented an ILP formulation for reliability-aware routing of AVB streams, i.e., determining the path and number of replicas for each AVB stream to meet the minimum reliability requirement for each stream and maximize overall network reliability. Feng *et al.* [22] presented a pro-active fault-tolerant TSN scheduling algorithm, which determines the number of transmissions per instance for TT flows in case of transient failures and also provides degraded network transmission service for non-TT flows, i.e., transmitting one message with multiple periods. Zhou *et al.* [23] presented SMT-based and heuristic techniques for routing and scheduling TT flows that combine spatial redundancy and temporal redundancy, considering reliability constraints and end-to-end deadlines.

3) Online Recovery Approaches: There are two aspects of online recovery: 1) the low-level online reconfiguration mechanism/protocol supported by the network and 2) the high-level online routing and scheduling algorithm for generating new configurations used for online recovery.

Regarding the online reconfiguration mechanism, the IEEE 802.1 TSN Working Group proposed the standard IEEE 802.1Qcc [26] to specify the configuration and runtime reconfiguration behavior of the network devices, which enables online recovery algorithms for fault tolerance of TSN. It specifies three network configuration models: 1) fully distributed; 2) centralized network/distributed user; and 3) fully centralized. Fig. 3 shows a fully centralized model with a configuration agent (CA) that performs configuration and reconfiguration of a TSN network at runtime. It consists of the central user configurator (CUC), which receives information from the ESs regarding changes to the traffic patterns in terms of appeared and disappeared flows (dotted arrows); and the centralized network configurator (CNC), which uses the received information to generate a new schedule for the network and distributes it to all network devices (dashed arrows). The delays in the transmission of configuration messages and reconfiguration of the nodes may cause inconsistent intermediate states where only a subset of all nodes are reconfigured while others have old configurations. To prevent this situation, Kostrzewa and Ernst [27] proposed a safe reconfiguration protocol that achieves traffic isolation, fault recovery, and controlled adjustments of network performance.

There are only a few related works on online routing, and scheduling algorithms for online recovery, which are orthogonal to and independent of the reconfiguration mechanism, e.g., either one of the three models specified in IEEE 802.1Qcc or some other protocol such as [27]. Pozo *et al.* [24] presented a repair algorithm capable of reacting to unpredictable link failures in TSN by modifying the schedule such that all frames are transmitted again. However, the timing constraints may be violated when a link failure occurs close to the deadline of a TT flow due to a lack of spatial redundancy. Kong *et al.* [25] presented a three-mode recovery scheme for TT flows, including full functionality, reduced functionality, and emergency halt modes. Runtime recovery for TT flows is explored using ILP and a heuristic routing and scheduling (HRS) algorithm. This work only handles permanent faults, not transient faults.

In relation to related works, our work has the following distinguishing characteristics.

1) We handle both permanent and transient faults by combining spatial and temporal redundancy.
2) We perform online recovery with rerouting and rescheduling, which is complementary to and can be used in combination with any offline routing/scheduling algorithm to form a complete (offline + online) algorithm (c.f., line 5 in Algorithm 2).
3) During online recovery, the routes and schedules of existing flows are kept unchanged to reduce disruption and runtime overhead.

Note on AVB Flows: In addition to TT flows, AVB flows scheduled with CBS may also play an essential role in safety-critical systems and have soft or hard timing constraints. There are some works (e.g., [28], [29], and [30]) on formal timing analysis of AVB flows in the presence of TT flows, and some works (e.g., [28] and [31]) on routing and scheduling of both TT and AVB flows so that all flows are schedulable. There may be multiple different problem formulations of fault-tolerant scheduling of (TT + AVB) flows: 1) offline design based on spatial and/or temporal redundancy to provide fault tolerance for both TT and AVB flows; 2) offline design based on spatial and/or temporal redundancy to provide fault tolerance for TT flows only, but not for AVB flows; 3) online reconfiguration for fault recovery of both TT and AVB flows; and 4) online reconfiguration for fault recovery of TT flows only, but considering their impact on AVB flows. However, we are not aware of any related work that addresses any one of the four problem formulations. Most related works on TSN fault tolerance listed in Table I address TT flows

only or (TT + BE) flows. The only work that addresses AVB flows, Atallah *et al.* [21], simply imposes constraints on link utilization upper bound (e.g., from 25% to 75%) to guarantee schedulability of flows traversing the link, so it is not specifically targeting AVB flows. We think one reason for the lack of related work may be that, from a research perspective, there is not much novelty in adding fault tolerance to offline design optimization of (TT + AVB) flows; another reason may be due to the computational complexity of timing analysis for AVB flows, either by computing the worst-case response time [29], [30] or by network calculus [32], especially for online reconfiguration. Our work in this article addresses TT flows as the only safety-critical flows, but we leave it as a possible future work to address AVB flows.

III. System Model and Notations

We model a TSN network as a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{L})$, where $\mathcal{V}$ denotes the set of nodes, which may be either end-systems or switches, and $\mathcal{L} \subseteq \mathcal{V} \times \mathcal{V}$ denotes the physical links between pairs of nodes. We only consider TT flows and use $\mathcal{F}$ to denote the set of all TT flows. Each flow f_i is periodic with period T_i. Non-TT flows, either AVB or BE flows, can be transmitted during *idle windows*, i.e., time slots not occupied by TT flows. Flow instance f_i^k is the kth instance of f_i, which sends one flow instance per period $T_i \in \mathbb{N}$ of f_i. $\mathcal{HP} \in \mathbb{N}$ denotes the hyperperiod, defined as the least common multiple (LCM) among all the periods of TT flows. Each flow $f_i \in \mathcal{F}$ has a talker end-system (source) $src(f_i) \in \mathcal{V}$ and a listener end-system (destination) $dest(f_i) \in \mathcal{V}$. $\mathcal{P}_i$ denotes f_i's set of N redundant paths; $\mathcal{P}_i^n$ denotes the nth redundant path in $\mathcal{P}_i$. The path $\mathcal{P}_i^n = \{[v_1, v_2], [v_2, v_3], \ldots, [v_e - 1, v_e]\}$ consists of a linear path from node v_1 to node v_e with e nodes and $e-1$ links, including both end-systems and switches. $|\mathcal{P}_i^n|$ denotes the total number of nodes in $\mathcal{P}_i^n$, and $N_i^n(j)$ denotes the jth node of $\mathcal{P}_i^n$, i.e., $N_i^n(j) = src(f_i)$. $[v_a, v_b]$ denotes the physical link from node v_a to v_b. $c_i^{[v_a,v_b]} \in \mathbb{N}$ denotes the transmission duration of flow f_i through link $[v_a, v_b]$. Flow transmission on each link is nonpreemptive and cannot be interrupted. $\Phi_{i,j}^{[v_a,v_b]} \in \mathbb{N}$ denotes the transmission offset of f_i^j on link $[v_a, v_b]$ relative to the start of the hyperperiod, i.e., f_i^j starts transmission on $[v_a, v_b]$ at time $\Phi_{i,j}^{[v_a,v_b]}$, and finishes transmission at time $\Phi_{i,j}^{[v_a,v_b]} + c_i^{[v_a,v_b]}$. $iro_i \in \mathbb{N}$ denotes the initial release offset of f_i at $src(f_i)$,[1] which may be any value in the interval $[0, T_i)$. Its value may be dependent on application task scheduling on the talker node, or it may be an optimization variable for load balancing on the time axis. (In our experiments, we set $iro_i = 0$ for simplicity.) Q_i denotes the queue ID of f_i in all the switches it passes through, which reflects its priority level. D_i denotes the end-to-end deadline of f_i.

We define $\mathcal{F}_{exi}^{[v_a,v_b]}$ and $\mathcal{F}_{\text{rer}}^{[v_a,v_b]}$ as the set of existing and rerouted flows transmitted traverse link $[v_a, v_b]$.

[1]Note that iro_i may be different from f_i's transmission offset at the talker ES $\Phi_{i,j}^{[src(f_i),v_x]}$, due to possible interference that may delay its start of transmission.

Definition 2: A path is a *broken path* if it passes through any failed link or switch due to permanent faults. A flow is a *disrupted flow* if at least one path among the flow's set of redundant paths contains a failure. A *rerouted flow* is a recovered flow after rerouting its broken path to a new healthy path.

For a given flow f_i, we define its degree of redundancy (DoR) for permanent or transient faults to measure its degree of fault resilience.

Definition 3: A flow f_i's DoR for *permanent faults* $\kappa_i^p \in \mathbb{N}$ is defined as the number of redundant paths assigned to the flow, i.e., f_i can tolerate up to $\kappa_i^p - 1$ permanent faults.

Definition 4: A flow f_i's DoR for *transient faults* $\kappa_i^t \in \mathbb{N}$ is defined as the number of redundant message copies transmitted either on the same path or on different redundant paths, i.e., f_i can tolerate up to $\kappa_i^t - 1$ transient faults.

If a transient fault corrupts one of the messages of flow f_i,κ_i^p and κ_i^t are not affected because the fault duration is transient, and the failure does not propagate to subsequent messages of f_i. If a permanent fault occurs and causes loss of a redundancy path, then both κ_i^p and κ_i^t are decreased. To recover the DoRs for both permanent and transient faults (κ_i^p and κ_i^t), new disjoint paths need to be established to bypass the failed link. To recover the DoR for transient faults (κ_i^t) only, additional redundant messages can be transmitted on an existing path.

We give some examples to clarify the DoR concept: if flow f_i has a single path from its talker to its listener, and a single message transmitted on the path, then $\kappa_i^p = \kappa_i^t = 1$. If flow f_i has a single path and two redundant messages transmitted on the path, then $\kappa_i^p = 1, \kappa_i^t = 2$. If flow f_i has two redundant paths and a single message transmitted on each path, then $\kappa_i^p = \kappa_i^t = 2$. If flow f_i has two redundant paths and two messages transmitted on each path, then $\kappa_i^p = 2, \kappa_i^t = 4$. If a path of flow f_i is disrupted by a link failure, then κ_i^p is decreased by 1, and κ_i^t is decreased by the number of messages transmitted along the path.

Table II contains the main notations used in this article.

IV. Our Fault Resilience Algorithm

In this section, we first present the overall fault resilience algorithm in Section IV-A, then present the component algorithms that it invokes, including: Algorithm 2 for the offline phase collection of redundant paths in Section IV-B, Algorithm 3 for offset assignment for a rerouted flow in Section IV-C, and WCD analysis for a rerouted flow in Section IV-D.

A. Overall Algorithm

Algorithm 1 shows our overall online recovery algorithm to recover the DoR of each TT flow. It consists of two phases: 1) an offline phase and 2) an online phase, explained as follows.

1) *Line 2:* In the offline phase, for each f_i, we build its set of r_i redundant paths $\mathcal{P}_i$ and k_i candidate redundant paths $\mathcal{CP}_i$ used for online recovery. The value of $r_i = \max(\kappa_i^p, \kappa_i^t)$ is set to satisfy both DoRs for permanent and transient faults.

TABLE II
MAIN NOTATIONS

Symbol	Description
$\mathcal{G}$	Network topology
$\mathcal{V}$	Set of nodes (end-systems and switches) in $\mathcal{G}$
$\mathcal{L}$	Set of links between nodes
f_i	TT flow i
$\mathcal{F}$	Set of all TT flows in the system
$\mathcal{P}_i$	Set of redundant paths of f_i
$r_i = \lvert\mathcal{P}_i\rvert$	Number of redundant paths of f_i
$\mathcal{CP}_i$	Set of candidate redundant paths of f_i
$k_i = \lvert\mathcal{CP}_i\rvert$	Number of candidate redundant paths of f_i
$\mathcal{P}_i^n$	The n-th redundant path of f_i in $\mathcal{P}_i$
$\mathcal{CP}_i^n$	The n-th candidate redundant path of f_i in $\mathcal{CP}_i$
$len(\mathcal{P}_i^n)$	Length of path $\mathcal{P}_i^n$ (number of nodes along the path)
nm_i	Number of redundant messages of f_i on the failed link
$wcd(\mathcal{P}_i^n)$	Worst-Case Delay of path $\mathcal{P}_i^n$
$N_i^n(j)$	The j-th node in $\mathcal{P}_i^n$
$[v_a, v_b]$	Physical link from node v_a to v_b
T_i	Period of f_i
D_i	End-to-end deadline of f_i
Q_i	Queue ID of f_i
$c_i^{[v_a,v_b]}$	Transmission duration of flow f_i over link $[v_a, v_b]$
iro_i	Initial release offset of f_i at $src(f_i)$
f_i^k	The k-th instance of f_i
$\mathcal{HP}$	Hyper-period of the TT schedule
$\Phi_{i,j}^{[v_a,v_b]}$	Transmission offset of f_i^j on link $[v_a, v_b]$
κ_i^p	f_i's Degree of Redundancy for permanent faults
κ_i^t	f_i's Degree of Redundancy for transient faults
$\mathcal{F}_{exi}$	Set of existing flows
$\mathcal{F}_{rer}$	Set of re-routed flows
$\mathcal{F}_{exi}^{[v_a,v_b]}$	Set of existing flows that traverse link $[v_a, v_b]$
$\mathcal{F}_{rer}^{[v_a,v_b]}$	Set of re-routed flows that traverse link $[v_a, v_b]$

Algorithm 1: Overall Fault Resilience Algorithm (Offline + Online)

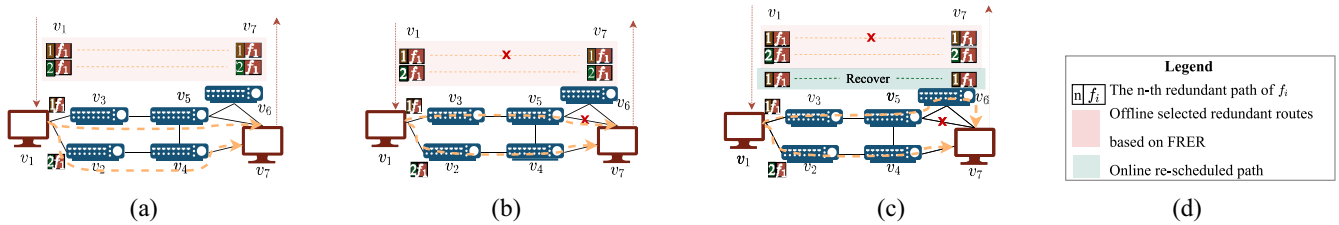
Input: $\mathcal{G}, \mathcal{F}, \{\kappa_i^t, \kappa_i^p | \forall f_i \in \mathcal{F}\}$

Output: paths and offsets $\{\mathcal{P}_i^n, \Phi_{i,k}^{[N_i^n(j), N_i^n(j+1)]} | f_i \in \mathcal{F}\}$

1 **Offline phase**;
2 Invoke Alg. 2 in Section IV-B to build an initial solution with a set of $r_i = \max(\kappa_i^p, \kappa_i^t)$ redundant paths $\mathcal{P}_i$ for each flow f_i, and a set of additional k_i candidate redundant paths $\mathcal{CP}_i$;
3 **Online phase**;
`// The online recovery algorithm is run if a permanent fault causes failure of a link.`
4 Remove the failed link from the network topology $\mathcal{G}$;
5 Move the disrupted flows due to the failed link from $\mathcal{F}$ to $\mathcal{F}_{\text{rer}}$ to be re-routed and re-scheduled;
6 **foreach** $f_i \in \mathcal{F}_{\text{rer}}$ **do**
`// Assume` nm_i `number of redundant messages from` f_i `are transmitted on the failed link.`
7 $\kappa_i^p = \kappa_i^p - 1, \kappa_i^t = \kappa_i^t - nm_i$;
8 **foreach** *candidate path* $\mathcal{CP}_i^n \in \mathcal{CP}_i$ **do**
9 Calculate its $wcd(\mathcal{CP}_i^n)$ with Eqn. (2) in Section IV-D;
10 **end**
11 Tentatively add the path $\mathcal{CP}_i^n$ with $\min\{wcd(\mathcal{CP}_i^n)\}$ to f_i's set of redundant paths $\mathcal{P}_i$, and invoke Alg. 3 in Section IV-C to schedule it (assign offsets);
12 **if** *Alg. 3 succeeds for* $\mathcal{CP}_i^n$ **then**
13 Add $\mathcal{CP}_i^n$ to f_i's set of redundant paths $\mathcal{P}_i$;
14 $\kappa_i^p = \kappa_i^p + 1, \kappa_i^t = \kappa_i^t + nm_i$;
15 **else**
16 **foreach** *path* $\mathcal{P}_i^n \in \mathcal{P}_i$ **do**
17 Calculate its $wcd(\mathcal{P}_i^n)$ with Eqn. (2);
18 **end**
19 Tentatively add a duplicate path by replicating the path $\mathcal{P}_i^n$ with $\min\{wcd(\mathcal{P}_i^n)\}$, and invoke Alg. 3 to schedule it;
20 **if** *Alg. 3 succeeds for* $\mathcal{P}_i^n$ **then**
21 Add the duplicate path $\mathcal{P}_i^n$;
22 $\kappa_i^t = \kappa_i^t + nm_i$;
23 **else**
24 Goto Line 4;
25 **end**
26 **end**
27 **end**

2) *Line 3:* Perform routing and scheduling with an offline algorithm for all flows and all r_i redundant paths of each flow f_i, assuming no temporal redundancy, i.e., one message is transmitted on each redundant path initially (but after executing the online recovery phase, more than one messages may be transmitted on each path).
3) *Lines 4–7:* Upon detecting a link failure,[2] collect the set of disrupted flows $\mathcal{F}_{\text{rer}}$ due to the failed link. Assuming nm_i number of redundant messages of flow f_i are transmitted on the failed link, then its DoR for permanent faults κ_i^p is decremented by 1; its DoR for transient faults κ_i^t is decremented by nm_i.
4) *Lines 8–10:* Calculate the WCD of each of the k_i candidate paths, used for selecting the candidate path with minimum WCD to be added to f_i's set of paths.
5) *Line 11:* We aim to recover the DoR for both permanent and transient faults via spatial (and temporal if $nm_i > 1$) redundancy by tentatively adding an additional path $\mathcal{CP}_i^n$ to f_i's set of redundant paths $\mathcal{P}_i$, chosen as the path with the minimum WCD $\min\{wcd(\mathcal{CP}_i^n)\}$ among f_i's set of candidate paths $\mathcal{CP}_i$, and sending nm_i redundant messages along it.
6) *Lines 12–14:* If the path is feasible, i.e., Algorithm 3 succeeds in assigning offsets to it, then the path is actually added, and its DoRs for both permanent and transient faults are restored to their original values.
7) *Lines 16–19:* If Algorithm 3 fails to assign feasible offsets to the additional path $\mathcal{CP}_i^n$, then we only aim to recover the DoRs for transient faults via temporal redundancy. We tentatively add a duplicate path by replicating one of f_i's existing set of redundant paths $\mathcal{P}_i$ with the shortest WCD (i.e., sending an *additional* nm_i redundant messages along it).
8) *Lines 20–22:* If the path is feasible, i.e., Algorithm 3 succeeds in assigning offsets to it, then the path is actually

[2]We only consider link failures since a switch failure caused by a permanent fault can be considered as equivalent to failures of all the links connected to it.



(a) (b) (c) (d)

Fig. 4. Toy example of online rerouting. (a) Normal operation with two redundant paths for flow f_1-based FRER. (b) Upon link failure on $[v_5, v_7]$, and flow f_1 loses one redundant path. (c) Flow f_1 is rerouted through $\{[v_5, v_6], [v_6, v_7]\}$. (d) Legend.

added, and its DoR for transient faults is restored to its original value.

9) *Lines 23–25:* If Algorithm 3 fails to assign feasible offsets to the duplicate path, then we give up on flow f_i and keep its current DoRs, and move on to the next flow in $\mathcal{F}_{\text{rer}}$.

Fig. 4 shows a toy example to illustrate the basic operations of Algorithm 1. Suppose f_1 is a TT flow in that transmits messages from v_1 to v_7, and the required DoRs are $\kappa_1^p = \kappa_1^t = 2$.

1) *Fig. 4(a):* In the offline phase, we build its set of $r_i = \max(\kappa_i^p, \kappa_i^t) = 2$ redundant paths, including $\mathcal{P}_1^1 = \{[v_1, v_3], [v_3, v_5], [v_5, v_7]\}$ and $\mathcal{P}_1^2 = \{[v_1, v_2], [v_2, v_4], [v_4, v_7]\}$, and k_i candidate redundant paths $\mathcal{CP}_i$ (not shown in the figure).
2) *Fig. 4(b):* Link $[v_5, v_7]$ is broken at time t due to a permanent fault. Since f_1 now has only one path, its DoRs $\kappa_1^p = \kappa_1^t$ both drop to 1.
3) *Fig. 4(c):* We recover the DoRs for both permanent and transient faults via spatial redundancy by adding an additional path $\mathcal{P}_i^n$, i.e., $\{[v_1, v_3], [v_3, v_5], [v_5, v_6], [v_6, v_7]\}$, selected as the path with minimum WCD among its k_i candidate redundant paths. Assuming Algorithm 3 succeeds in assigning feasible offsets to this new path, so now f_1 again has $r_i = 2$ redundant paths, and its DoRs $\kappa_1^p = \kappa_1^t$ are both incremented by 1 to become 2 again.

B. Offline Phase

For each flow f_i between a given source-destination pair of end-systems, we collect a group of $r_i + k_i$ redundant paths, including a set of $r_i = \max(\kappa_i^p, \kappa_i^t)$ redundant paths $\mathcal{P}_i$, and a set of additional k_i candidate redundant paths $\mathcal{CP}_i$, used as a pool of backup paths to choose from if one or more of the r_i paths in $\mathcal{P}_i$ fails due to a permanent fault, The choice of k_i is a design hyperparameter that can be set by the designer. The set of $r_i + k_i$ redundant paths is collected among all possible paths between the source and destination with the criteria of having the shortest distances as measured by the number of hops.

For any given path $\mathcal{P}_i^n \in \mathcal{P}_i \cup \mathcal{CP}_i$, we define its NH_i^n as the Number of Hops along the path $\mathcal{P}_i^n$, equal to the number of switches along the path excluding the source and destination end-systems: $\text{NH}_i^n = \text{len}(\mathcal{P}_i^n) - 2$. We define its residual bandwidth B_i^n as the minimum residual bandwidth among all of its component links, i.e., $B_i^n = \min_{m=1}^{\text{len}(\mathcal{P}_i^n)-1} BW([N_i^n(m), N_i^n(m+1)])$, where $BW([N_i^n(m), N_i^n(m+1)])$ denotes the residual bandwidth over the link $[N_i^n(m), N_i^n(m+1)]$, i.e., the currently available bandwidth after subtracting the occupied bandwidth from the total available bandwidth on the link.

Algorithm 2: Offline Routing and Scheduling

Input: $\mathcal{F}, \{\kappa_i^p, \kappa_i^t | \forall f_i \in \mathcal{F}\}, k_i$
Output: $\mathcal{P}_i$ and $\mathcal{CP}_i$
1 **foreach** *Flow* $f_i \in \mathcal{F}$ **do**
2 $r_i = \max\{\kappa_i^p, \kappa_i^t\}$;
3 Collect the set of $r_i + k_i$ redundant paths $\mathcal{P}_i \cup \mathcal{CP}_i$ for f_i, based on the shortest distance metric;
4 Select the set of r_i redundant paths $\mathcal{P}_i$ for f_i, as the top r_i paths among the $r_i + k_i$ paths, ranked by QoS metric ρ_i^n in Eqn. (1);
5 Perform scheduling by assigning offsets for all flows and all r_i redundant paths of each flow f_i with an offline algorithm (assuming no temporal redundancy);
6 **end**

We define a quality-of-service (QoS) metric ρ_i^n based on [33]

$$\rho_i^n = w_1 \frac{\text{NH}_{\min}}{\text{NH}_i^n} + w_2 \frac{B_i^n}{B_{\max}} \tag{1}$$

where weighting factors $w_1 + w_2 = 1$; $\text{NH}_{\min}$ denotes the minimum NH_i^n among the $r_i + k_i$ paths $\mathcal{P}_i \cup \text{CP}_i$; and $B_{\max}$ is the maximum of B_i^n among the $r_i + k_i$ paths.

For each flow f_i, we rank the set of $r_i + k_i$ paths by the metric ρ_i^n in (1), and select the top-ranking r_i paths among them as the initial set of redundant paths $\mathcal{P}_i$ for flow f_i. That is, we preferably chose a path with a higher QoS metric, i.e., higher bandwidth B_i^n and/or smaller number of hops NH_i^n, in order to avoid bandwidth bottlenecks along the path.

Algorithm 2 is mostly self-explanatory. For line 5, any offline TSN schedule synthesis algorithm may be used. (In our experiments, we use the ILP formulation in [34], which performs scheduling assuming the routed paths of all flows are obtained from a preprocessing step.)

C. Offset Assignment for the Rerouted Flow

Consider the set of rerouted disrupted flows $\mathcal{F}_{\text{rer}}$ that have been assigned new paths, but have not been scheduled, we address the problem of rescheduling, i.e., assigning time slots to each flow $f_i \in \mathcal{F}_{\text{rer}}$ on each link along its new path, so that its end-to-end deadline D_i is satisfied, and there is no flow interleaving with the existing set of flows $\mathcal{F}_{\text{rer}}$. Since TT flows are scheduled along the path statically, and flow transmission on each link is nonpreemptive, we use the following terms

Algorithm 3: Efficient Algorithm for Offset Assignment of a Rerouted Flow f_i

Input: a path $\mathcal{P}_i^n$ for flow $f_i \in \mathcal{F}_{rer}$
Output: either feasible offsets $\{\Phi_{i,k}^{[v_a,v_b]} | k = 1, ..., \frac{\mathcal{HP}}{T_i}\}$ for $\mathcal{P}_i^n$, or Fail

1 // For each period T_i of f_i within the hyperperiod
2 **for** $k = 1, ..., \frac{\mathcal{HP}}{T_i}$ **do**
3 // For each node along the path
4 **for** $j = 1, ..., |\mathcal{P}_i^n| - 1$ **do**
5 // Set initial offset values
6 **if** $j == 1$ **then**
7 $\Phi_{i,k}^{[N_i^n(j),N_i^n(j+1)]} = (k-1)T_i + iro_i$;
8 **else**
9 $\Phi_{i,k}^{[N_i^n(j),N_i^n(j+1)]} = \Phi_{i,k}^{[N_i^n(j-1),N_i^n(j)]} + c_i^{[N_i^n(j-1),N_i^n(j)]}$;
10 **end**
11 Find the smallest feasible offset $\Phi_{i,k}^{[N_i^n(j),N_i^n(j+1)]}$ that avoids flow interleaving;
12 **if** $\Phi_{i,k}^{[N_i^n(j),N_i^n(j+1)]} + c_i^{[N_i^n(j),N_i^n(j+1)]} > D_i$ **then**
13 **return** Fail
14 **end**
15 **end**
16 **end**
17 **return** $\{\Phi_{i,k}^{[v_a,v_b]} | k = 1, ..., \frac{\mathcal{HP}}{T_i}\}$

interchangeably for a flow along a given path: *rescheduling, offset assignment*, and *time slot assignment*. Recall that we perform incremental scheduling, which means that schedules of existing flows $\mathcal{F}_{\text{exi}}$ cannot be modified to accommodate new flows.

Algorithm 3 shows the heuristic algorithm for online rescheduling of the set of rerouted flows $\mathcal{F}_{\text{rer}}$ by assigning offsets to each switch along each new path. We use $\mathcal{F}_{\text{exi}}$ to denote the set of existing flows that have already been scheduled. Note that the hyperperiod $\mathcal{HP}$ is a constant that is calculated based on all the flows in the system.

1) *Lines 1 and 2:* Self-explanatory with the inline comments.
2) *Lines 3 and 4:* For flow instance f_i^k, set the initial offset of the 1st link along the path (from the source end-system) to be the start of its period: $\Phi_{i,k}^{[N_i^n(1),N_i^n(2)]} = (k-1)T_i + iro_i$.
3) *Lines 5 and 6:* For flow instance f_i^k, set initial offset of the jth link along the path to be the transmission finish time of the $(j-1)$th link: $\Phi_{i,k}^{[N_i^n(j),N_i^n(j+1)]} = \Phi_{i,k}^{[N_i^n(j-1),N_i^n(j)]} + c_i^{[N_i^n(j-1),N_i^n(j)]}$. This offset is the earliest possible time for f_i^k to start transmission, but it may cause flow interleaving, hence it may need to be adjusted/pushed to a later time point for flow isolation.
4) *Line 8:* Find the smallest feasible offset $\Phi_{i,k}^{[N_i^n(j),N_i^n(j+1)]}$ that avoids flow interleaving, i.e., start of the first/earliest idle window on the link $[N_i^n(j), N_i^n(j+1)]$ with window size not less than its transmission duration on the link, so that f_i^k can fit into the idle window.

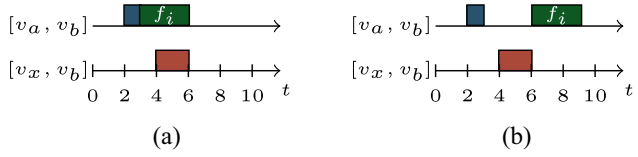


Fig. 5. Example of offset assignment for flow f_i on link $[v_a, v_b]$ with Algorithm 3. (a) Initial offset assignment. (b) Final offset assignment.

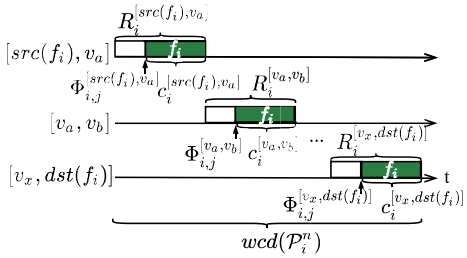


Fig. 6. Illustration of the end-to-end WCD of path $\mathcal{P}_i^n$ in (2).

5) *Lines 9–11:* If the offset assignment causes f_i^k to miss the end-to-end deadline of f_i, return Fail.

Fig. 5 shows a toy example to illustrate the operation of Algorithm 3 on a single link. The blue and red boxes denote the time slots assigned to existing flows along links $[v_a, v_b]$ and $[v_x, v_b]$, respectively. Assume a disrupted flow f_i is rerouted to pass through $[v_a, v_b]$, and its transmission duration $c_i^{[v_a,v_b]} = 3$. (In the figure, we use f_i and omit the superscript k in f_i^k for the sake of simplicity.) We assume the flow that occupies the red box shares the same egress queue as f_i in switch v_b. Hence, it is a conflicting flow and must be assigned a nonoverlapping time slot from f_i to avoid flow interleaving (refer to Fig. 2). Suppose after lines 3–7, the flow instance f_i^k is assigned an initial offset of 3 on $[v_a, v_b]$, as shown in Fig. 5(a). But this offset assignment is infeasible due to flow interleaving with the flow occupying the red box on link $[v_x, v_b]$. After executing line 8 to avoid flow interleaving, its offset is postponed to 6, which is a feasible offset assignment, since its transmission duration $c_i^{[v_a,v_b]} = 3$ fits within the idle window of $[6, 9]$.

D. WCD Analysis for the Rerouted Path

In this section, we present an algorithm for computing a pessimistic estimate of the end-to-end WCD for a rerouted path $\mathcal{P}_i^n$, denoted wcd$(\mathcal{P}_i^n)$, without performing offset assignment using Algorithm 3 for the sake of computational efficiency. Equation (2) shows that the end-to-end WCD of path $\mathcal{P}_i^n$ is computed by summing over the WCD on each link $[v_a, v_b]$ along the path, denoted $R_i^{[v_a,v_b]}$. Fig. 6 illustrates (2) graphically, including the end-to-end WCD wcd$(\mathcal{P}_i^n)$, WCD on each link $R_i^{[v_a,v_b]}$, transmission offset $\Phi_{i,j}^{[v_a,v_b]}$, and transmission duration $c_i^{[v_a,v_b]}$

$$\text{wcd}(\mathcal{P}_i^n) = \sum_{[v_a,v_b] \in \mathcal{P}_i^n} R_i^{[v_a,v_b]}. \tag{2}$$

Next, we focus on the issue of computing the WCD $R_i^{[v_a,v_b]}$ on a single link $[v_a, v_b]$.

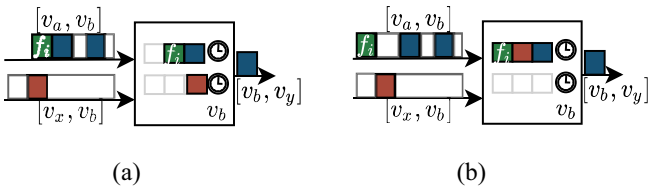


Fig. 7. Illustration of interference to f_i due to conflicting flows. (a) Flow f_i on link $[v_a, v_b]$ and a flow on another link $[v_x, v_b]$ are assigned to different queues in v_b, hence they are not conflicting flows. (b) Flow f_i on link $[v_a, v_b]$ and a flow on another link $[v_x, v_b]$ are assigned to the same queue in v_b, hence they are conflicting flows.

Fig. 7 illustrates two types of conflicting flows. In both subfigures, there is a conflicting flow (denoted by blue boxes) on the same link $[v_a, v_b]$ as f_i (the green box). In Fig. 7(a), it is the only conflicting flow, since the flow on another link $[v_x, v_b]$ (denoted by a red box) is assigned to a different queue within v_b; In Fig. 7(b), the flow on link $[v_x, v_b]$ is also a conflicting flow, since it is assigned to the same queue within v_b.

Next, we present a series of lemmas for computing worst-case interference caused by conflicting flows to a rerouted flow f_i on link $[v_a, v_b]$, and a theorem for computing the WCD of flow f_i.

Lemma 1 (Maximum Number of Interferences by a Conflicting Flow Within Any Time Interval): Consider an existing conflicting flow f_j on link $[v_*, v_b]$, which may be either the same link $[v_a, v_b]$ or another link $[v_x, v_b]$. During any time interval of length Δ, at most $(\lfloor(\Delta/T_j)\rfloor + 2)$ instances of f_j may cause interference to the rerouted flow f_i.

Proof: Since we do not make any assumptions on the offset assignments of existing flows, in the worst case, there are $\lfloor(\Delta/T_j)\rfloor$ whole periods for each conflicting flow f_j during Δ. Before and after $\lfloor(\Delta/T_j)\rfloor$ whole periods of f_j, there are up to two incomplete periods within Δ that may contain up to two flow instances of f_j. Thus, during Δ, at most $(\lfloor(\Delta/T_j)\rfloor + 2)$ instances of f_j may arrive at the same switch v_b as f_i. When the last flow instance of f_j finishes transmission, f_i can start transmission if there is no other interference. The lemma is thus proven. ∎

Lemma 2 (Worst-Case Interference by a Conflicting Flow Within Any Time Interval): During any interval with length Δ, the worst-case interference caused by a conflicting flow f_j on link $[v_*, v_b]$ to the rerouted flow f_i on link $[v_a, v_b]$ is $(\lfloor(\Delta/T_j)\rfloor + 2)(c_j^{[v_*,v_b]} + c_i^{[v_a,v_b]} - 1)$, where $c_j^{[v_*,v_b]}$ denotes the transmission duration of f_j on link $[v_*, v_b]$, and $c_i^{[v_a,v_b]}$ denotes the transmission duration of f_i on link $[v_a, v_b]$.

Proof: A conflicting flow f_j's interference to the rerouted flow f_i may consist of two parts: its own message transmission duration $c_j^{[v_*,v_b]}$, and the idle windows in-between its message transmissions. Since message transmission is nonpreemptive, f_i can only be transmitted when an idle window is no less than $c_i^{[v_a,v_b]}$. The worst-case interference due to idle windows happens when every idle window has size $(c_i^{[v_a,v_b]} - 1)$, so that the flow instance of f_i cannot fit within it (assuming integer time unit). In the worst case, there may be up to $(\lfloor(\Delta/T_j)\rfloor + 2) - 1$ idle windows in-between transmissions of all flow instances during Δ. An additional idle window would be generated between the beginning of Δ and the first appearing instance. We then can obtain the total worst-case interference of f_i by f_j within time interval Δ over $[v_a, v_b]$ as $(\lfloor(\Delta/T_j)\rfloor + 2)(c_j^{[v_*,v_b]} + c_i^{[v_a,v_b]} - 1)$. ∎

TABLE III
FLOWS IN FIG. 8. ALL TIME UNITS ARE IN US

Flow	c_i	$T_i = D_i$	Path	Existing flow
f_1	1	20	$\{[v_1, v_2], [v_2, v_3]\}$	Yes
f_2	1	40	$\{[v_4, v_2], [v_2, v_3]\}$	Yes
f_3	3	40	$\{[v_4, v_2], [v_2, v_3]\}$	No

Based on Lemmas 1 and 2, the following lemmas follow by summing over all conflicting flows on the same link (Lemma 3) or on different links (Lemma 4).

Lemma 3 (Total Interference From Conflicting Flows on the Same Link): During any time interval with length Δ, the worst-case interference to the rerouted flow $f_i \in \mathcal{F}_{\text{rer}}^{[v_a,v_b]}$ on link $[v_a, v_b]$ from conflicting flows on the same link $[v_a, v_b]$ is

$$I_s(i, \Delta, v_a) = \sum_{j \in \mathcal{F}_{exi}^{[v_a,v_b]}} \left(\left\lfloor \frac{\Delta}{T_j} \right\rfloor + 2\right)\left(c_j^{[v_a,v_b]} + c_i^{[v_a,v_b]} - 1\right) \tag{3}$$

where $\mathcal{F}_{exi}^{[v_a,v_b]}$ denotes the set of existing flows on link $[v_a, v_b]$.

Lemma 4 (Total Interference From Conflicting Flows on Different Links): During any time interval with length Δ, the worst-case interference to the rerouted flow $f_i \in \mathcal{F}_{\text{rer}}^{[v_a,v_b]}$ on link $[v_a, v_b]$ from conflicting flows on different links $[v_x, v_b]$ that share the same queue as f_i in switch v_b is

$$I_o(i, \Delta, v_a) = \sum_{\substack{[v_x,v_b] \in \mathcal{L} \\ v_x \neq v_a}} \sum_{\substack{j \in \mathcal{F}_{exi}^{[v_x,v_b]} \\ Q_j = Q_i}} \left(\left\lfloor \frac{\Delta}{T_j} \right\rfloor + 2\right)\left(c_j^{[v_x,v_b]} + c_i^{[v_a,v_b]} - 1\right). \tag{4}$$

Flow f_i's WCD $R_i^{[v_a,v_b]}$ on link $[v_a, v_b]$ consists of its own transmission duration plus the worst-case interferences from both types of conflicting flows, as shown in Theorem 1.

Theorem 1 (WCD On a Given Link): Given a rerouted TT flow $f_i \in \mathcal{F}_{\text{rer}}^{[v_a,v_b]}$ and existing flows, f_i's WCD $R_i^{[v_a,v_b]}$ on the link $[v_a, v_b]$ can be computed as the minimum fixed point of the following recursive equation:

$$R_i^{[v_a,v_b]} = c_i^{[v_a,v_b]} + I_s\left(i, R_i^{[v_a,v_b]}, v_a\right) + I_o\left(i, R_i^{[v_a,v_b]}, v_a\right) \tag{5}$$

where $I_s(i, R_i^{[v_a,v_b]}, v_a)$ and $I_o(i, R_i^{[v_a,v_b]}, v_a)$ are obtained by Lemmas 3 and 4, respectively.

A Toy Example: Consider three flows, f_1, f_2, and f_3, shown in Fig. 8(a) and Table III. The column c_i denotes each flow f_i's transmission duration on each link. f_1 and f_2 are existing flows; f_3 is a rerouted flow resulting from the online phase of Algorithm 1. The routes and schedules of existing flows f_1 and f_2 are fixed, and f_3 needs to be scheduled online to meet its deadline in the presence of f_1 and f_2. We now calculate its WCD $R_3^{[v_4,v_2]}$.

Flows f_2 and f_3 are always conflicting flows, since they traverse the same link $[v_4, v_2]$ before reaching switch v_2. In

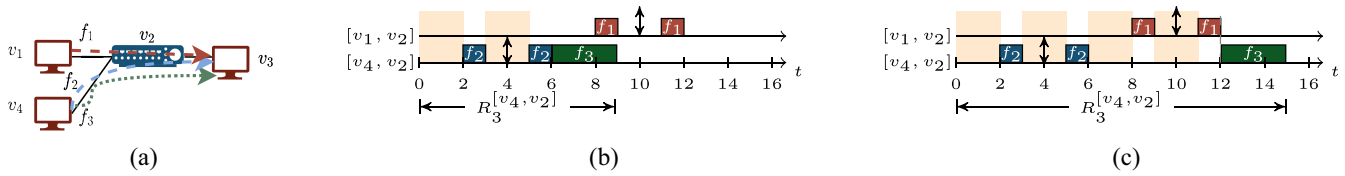


Fig. 8. Example of computing the WCD $R_3^{[v_4,v_2]}$ of f_3 on link $[v_1, v_2]$. On the time axis in (b) and (c), $t = 0$ denotes the start of the busy period based on the release time of f_3. (a) Topology. (b) f_1 and f_3 are assigned to different queues in v_2, hence they are not conflicting flows. (c) f_1 and f_3 are assigned to the same queue in v_2, hence they are conflicting flows.

Fig. 8(b), we assume that flows f_1 and f_3 are assigned to two different queues in switch v_2 ($Q_1 \neq Q_3$), hence they are not conflicting flows; in Fig. 8(c), we assume that f_1 and f_3 are assigned to the same queue in switch v_2 ($Q_1 = Q_3$), hence they are conflicting flows. We treat each case separately.

Case 1 in Fig. 8(b): Since f_1 is not a conflicting flow with f_3, we only need to consider the interference of f_2 (on the same link) to f_3. According to (5), we have

$$\begin{aligned} R_3^{[v_4,v_2]} &= c_3^{[v_4,v_2]} + I_s\left(3, R_3^{[v_4,v_2]}, v_4\right) \\ &= c_3^{[v_4,v_2]} + \left(\left\lfloor \frac{R_3^{[v_4,v_2]}}{T_2} \right\rfloor + 2\right)\left(c_2^{[v_4,v_2]} + c_3^{[v_4,v_2]} - 1\right) \\ &= 3 + \left(\left\lfloor \frac{R_3^{[v_4,v_2]}}{40} \right\rfloor + 2\right)(1 + 3 - 1) \\ &= 9 + 3\left\lfloor \frac{R_3^{[v_4,v_2]}}{40} \right\rfloor = 9. \end{aligned}$$

Case 2 in Fig. 8(c): Since both f_1 and f_2 are conflicting flows with f_3, we need to consider both the interference of f_1 (on another link) to f_3, and the interference of f_2 (on the same link) to f_3. According to (5), we have

$$\begin{aligned} R_3^{[v_4,v_2]} &= c_3^{[v_4,v_2]} + I_s\left(3, R_3^{[v_4,v_2]}, v_4\right) + I_o\left(3, R_3^{[v_4,v_2]}, v_4\right) \\ &= c_3^{[v_4,v_2]} + \left(\left\lfloor \frac{R_3^{[v_4,v_2]}}{T_2} \right\rfloor + 2\right)\left(c_2^{[v_4,v_2]} + c_3^{[v_4,v_2]} - 1\right) \\ &\quad + \left(\left\lfloor \frac{R_3^{[v_4,v_2]}}{T_1} \right\rfloor + 2\right)\left(c_1^{[v_1,v_2]} + c_3^{[v_4,v_2]} - 1\right) \\ &= 3 + \left(\left\lfloor \frac{R_3^{[v_4,v_2]}}{40} \right\rfloor + 2\right)(1 + 3 - 1) \\ &\quad + \left(\left\lfloor \frac{R_3^{[v_4,v_2]}}{20} \right\rfloor + 2\right)(1 + 3 - 1) \\ &= 15 + 3\left\lfloor \frac{R_3^{[v_4,v_2]}}{40} \right\rfloor + 3\left\lfloor \frac{R_3^{[v_4,v_2]}}{20} \right\rfloor = 15. \end{aligned}$$

The WCD $R_3^{[v_2,v_3]}$ can be computed similarly, and we omit the detailed derivations. Finally, the WCD of f_3 can be obtained by summing up all delays on each link that f_3 pass through, i.e., $\text{wcd}(f_3) = R_3^{[v_4,v_3]} = R_3^{[v_4,v_2]} + R_3^{[v_2,v_3]}$.

V. Performance Evaluation

The experiment platform is a workstation with 64-bit 4-core 2.3-GHz Intel Core i5-6300HQ with 32-GB memory. We adopt a similar experimental setup to [7]. We generate 80 random network topologies, each with eight switches and eight end-systems. Each flow's source and destination nodes are chosen randomly from the end-systems. Every end-system is attached to three switches, and every switch is attached to at least three other switches. The time granularity used by the scheduler (macrotick) is 1 us. Each flow f_i's period T_i is uniformly randomly distributed within [80, 160] us, with deadline equal to period $D_i = T_i$. Flow f_i's queue ID in each switch Q_i is uniformly randomly chosen as an integer within [0, 7]. Each flow has two redundant paths, and one message is transmitted on each path, i.e., $\kappa_i^p = \kappa_i^t = 2$. The link bandwidth is 1 Gb/s. Every message has the same size of 500 bytes, so its transmission delay on each link is 4 us. All flows start transmitting at $t = 0$. We set $k_i = 8$, i.e., we collect an additional eight candidate redundant paths for use during online fault recovery. For offline path selection described in Section IV-B, we set equal weights $w_1 = w_2 = 0.5$ and impose a maximum number of 5 hops for every path, i.e., $\text{NH}_i^n \leq 5$ in (1).

We compare the following methods.

1) *FRER:* The offline configuration obtained by Algorithm 2 based on the ILP-based offline scheduling algorithm in [34], enhanced to establish two redundant paths for each flow. We use Gurobi (https://www.gurobi.com) as the ILP solver. It also serves as the initial configuration for the three online recovery methods below. There is no online recovery phase.
2) *OUR:* Algorithm 1.
3) *OUR-Ran:* Algorithm 1 with random selection of the candidate path $\mathcal{CP}_i^n \in \mathcal{CP}_i$ on line 11, instead of the candidate path with minimum WCD $\min\{\text{wcd}(\mathcal{CP}_i^n)\}$.
4) *HRS:* The HRS algorithm in [25]. The period of TAS is referred to as the base period, and it is divided into smaller synchronized time slots with uniform size, and the duration of each time slot must be larger than the nonqueuing end-to-end delay of a maximum-sized frame. We adopt these parameter settings for HRS: the base period is 80 us; the maximum number of hops is 5. The time slot length is set to be the end-to-end delay $5 \times 4 = 20$ us, since each hop has a transmission delay of 4 us. So the base period contains $80/20 = 4$ time slots.

We first evaluate the fault tolerance performance in terms of the DoR values for both permanent and transient faults upon link failures. We set the total number of TT flows $\mathcal{F}$ in the system to be 20. For each network topology, we randomly inject a permanent fault to cause a random link failure and

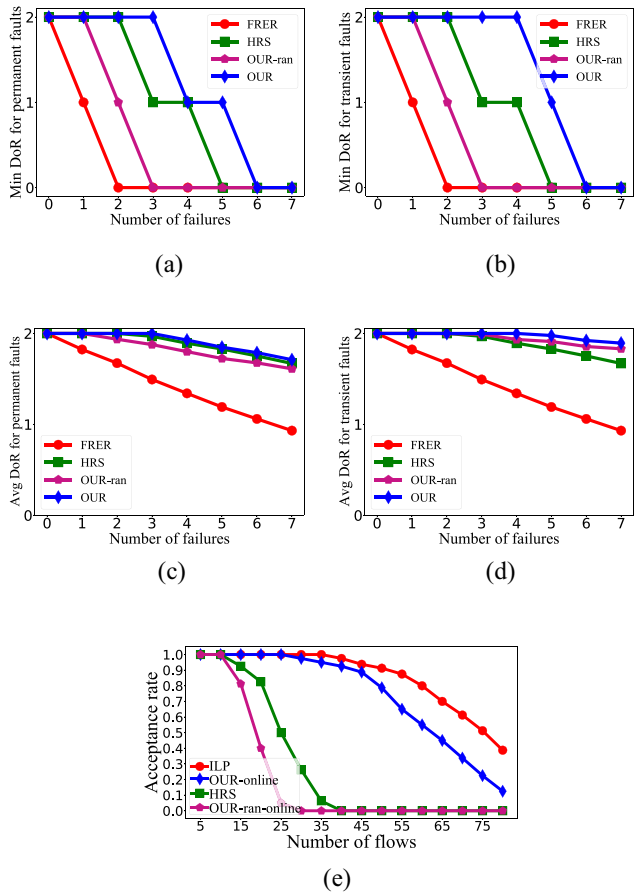


Fig. 9. Experiment results with random network topologies. (a) Recovered minimum DoR for permanent faults versus the number of link failures. (b) Recovered minimum DoR for transient faults versus the number of link failures. (c) Recovered average DoR for permanent faults versus the number of link failures. (d) Recovered average DoR for transient faults versus the number of link failures. (e) Acceptance rate versus the number of flows.

run each of three online recovery methods. After executing the online recovery phase, we record the following metrics for all flows f_i: minimum DoR values for permanent faults $\min(\kappa_i^p)$ and for transient faults $\min(\kappa_i^t)$; average DoR values for permanent faults $\text{avg}(\kappa_i^p)$ and for transient faults $\text{avg}(\kappa_i^t)$.[3] We repeat the experiment for seven rounds, with one additional link failure in each round starting from the configuration after recovery from the previous round, hence the number of consecutive link failures grows from 0 to 7. The results are plotted in Fig. 9(a)–(d).

1) *For the Minimum DoR Values* $\min(\kappa_i^p)$ *and* $\min(\kappa_i^t)$ *in Fig. 9(a) and (b):* For FRER, they both drop linearly (until reaching 0) with the number of failures since it has no online recovery phase. For the three online methods, the performance ranking is OUR>HRS>OUR-ran for both $\min(\kappa_i^p)$ and $\min(\kappa_i^t)$. All three methods can provide perfect protection against one single-link failure ($\min(\kappa_i^p) = \min(\kappa_i^t) = 2$). They drop to 0 ($\min(\kappa_i^p) = \min(\kappa_i^t) = 0$) upon 6, 5, and 3 link failures for OUR, HRS, and OUR-ran, respectively. This happens when at least one of the flows' talker and listener end-systems become disconnected.
2) *For the Average DoR Values* $\text{avg}(\kappa_i^p)$ *and* $\text{avg}(\kappa_i^t)$ *in Fig. 9(c) and (d):* We observe similar performance rankings, with OUR consistently being the best-performing method, but with smaller differences among the three online methods. [OUR-ran outperforms HRS slightly in Fig. 9(d), which may be within the statistical noise due to the randomness in the experimental setup.]

Next, we consider the fault-free scenario, in order to evaluate the general effectiveness of different methods in finding feasible solutions. For each of the 80 network topologies, we vary the number of TT flows from 5 to 80 with a step size of 5, so the total number of test scenarios is $80 \times 16 = 1280$. We define the metric of *acceptance rate* as the percentage of problem instances for which all flows are successfully routed and scheduled with no spatial or temporal redundancy, i.e., even there is only one single flow fails to be routed and scheduled, the set of flows $\mathcal{F}$ is not accepted. We consider the following approaches.

1) The ILP-based offline scheduling algorithm in [34].
2) HRS in [25].
3) *OUR-Online:* OUR (Algorithm 1) with online phase only.
4) *OUR-Ran-Online:* OUR-ran with online phase only.

Fig. 9(e) plots the acceptance rates versus the number of flows. Each data point is the average value over all 80 network topologies for a given number of flows. The acceptance rates for all methods decrease with increasing number of TT flows. The ILP-based offline scheduling algorithm has the best performance, since it performs routing and scheduling for all flows at the same time, while the three online methods consider each flow sequentially as each one is generated. The three online methods have an acceptance rate ranking of OUR-online>HRS>OUR-ran-online, which is consistent with the results in Fig. 9(a)–(d).

VI. Conclusion

In this article, we address the issue of fault tolerance against permanent and transient faults for TT flows in TSN and propose an efficient heuristic algorithm for online incremental rerouting and rescheduling of disrupted flows, where the paths and schedules of existing flows stay fixed. Performance evaluation experiments by injection of permanent faults indicate that our approach can better recover the system's DoR and has a higher acceptance rate than the comparison baselines.

References

[1] L. Deng, G. Xie, H. Liu, Y. Han, R. Li, and K. Li, "A survey of real-time ethernet modeling and design methodologies: From AVB to TSN," *ACM Comput. Surveys*, vol. 55, no. 2, pp. 1–36, 2022.
[2] Y. Lin, X. Jin, T. Zhang, M. Han, N. Guan, and Q. Deng, "Queue assignment for fixed-priority real-time flows in time-sensitive networks: Hardness and algorithm," *J. Syst. Architect.*, vol. 116, 2021, Art. no. 102141.
[3] X. Jin, C. Xia, N. Guan, and P. Zeng, "Joint algorithm of message fragmentation and no-wait scheduling for time-sensitive networks," *IEEE/CAA J. Automatica Sinica*, vol. 8, no. 2, pp. 478–490, Feb. 2021.

[3]The minimum DoR values are important, if we assume all TT flows to be safety-critical, and the flow with the lowest reliability determines the overall system reliability. The average DoR values are also useful metrics, since they describe the average fault tolerance capability among all flows.

[4] X. Dai, S. Zhao, Y. Jiang, X. Jiao, X. S. Hu, and W. Chang, "Fixed-priority scheduling and controller co-design for time-sensitive networks," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, 2020, pp. 1–9.

[5] "IEEE/ISO/IEC international standard for information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—part 1AS: Timing and synchronization for time-sensitive applications in bridged local area networks," ISO/IEC/IEEE Standard 8802–1AS:2021(E), pp. 1–422, 2021.

[6] *IEEE Standard for Local and Metropolitan Area Networks–FRER for Reliability*, IEEE Standard 802.1CB-2017, pp. 1–102, 2017.

[7] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, "Scheduling real-time communication in IEEE 802.1 Qbv time sensitive networks," in *Proc. 24th Int. Conf. Real Time Netw. Syst.*, 2016, pp. 183–192.

[8] G. Avni, S. Guha, and G. Rodriguez-Navas, "Synthesizing time-triggered schedules for switched networks with faulty links," in *Proc. 13th Int. Conf. Embedded Softw.*, 2016, pp. 1–10.

[9] V. Gavrilut, B. Zarrin, P. Pop, and S. Samii, "Fault-tolerant topology and routing synthesis for IEEE time-sensitive networking," in *Proc. 25th Int. Conf. Real Time Netw. Syst.*, 2017, pp. 267–276.

[10] A. A. Atallah, G. B. Hamad, and O. A. Mohamed, "Fault-resilient topology planning and traffic configuration for IEEE 802.1 Qbv TSN networks," in *Proc. IEEE IOLTS*, 2018, pp. 151–156.

[11] M. Pahlevan, S. Amin, and R. Obermaisser, "Fault tolerant list scheduler for time-triggered communication in time-sensitive networks," *J. Commun.*, vol. 16, no. 7, pp. 250–258, 2021.

[12] A. A. Syed, S. Ayaz, T. Leinmʹuller, and M. Chandra, "Fault-tolerant dynamic scheduling and routing for TSN based in-vehicle networks," in *Proc. IEEE VNC*, 2021, pp. 72–75.

[13] K. Huang *et al.*, "Reliability-aware multipath routing of time-triggered traffic in time-sensitive networks," *Electronics*, vol. 10, no. 2, p. 125, 2021.

[14] N. Reusch, S. S. Craciunas, and P. Pop, "Dependability-aware routing and scheduling for time-sensitive networking," 2021, *arXiv:2109.05883*.

[15] R. Dobrin, N. Desai, and S. Punnekkat, "On fault-tolerant scheduling of time sensitive networks," in *Proc. 4th Int. Workshop Secur. Dependabil. Crit. Embedded Real Time Syst. (CERTS)*, 2019, p. 5.

[16] T. Park, S. Samii, and K. G. Shin, "Design optimization of frame preemption in real-time switched ethernet," in *Proc. Design, Autom. Test Europe Conf. Exhibit. (DATE)*, 2019, pp. 420–425.

[17] Z. Feng, Q. Deng, M. Cai, and J. Li, "Efficient reservation-based fault-tolerant scheduling for IEEE 802.1 Qbv time-sensitive networking," *J. Syst. Architect.*, vol. 123, Feb. 2022, Art. no. 102381.

[18] I. Álvarez, J. Proenza, M. Barranco, and M. Knezic, "Towards a time redundancy mechanism for critical frames in time-sensitive networking," in *Proc. 22nd IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, 2017, pp. 1–4.

[19] I. Álvarez, D. Čavka, J. Proenza, and M. Barranco, "Simulation of the proactive transmission of replicated frames mechanism over TSN," in *Proc. 24th IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, 2019, pp. 1375–1378.

[20] I. Álvarez, I. Furió, J. Proenza, and M. Barranco, "Design and experimental evaluation of the proactive transmission of replicated frames mechanism over time-sensitive networking," *Sensors*, vol. 21, no. 3, p. 756, 2021.

[21] A. A. Atallah, G. B. Hamad, and O. A. Mohamed, "Reliability-aware routing of AVB streams in TSN networks," in *Proc. Int. Conf. Ind. Eng. Appl. Appl. Intell. Syst.*, 2018, pp. 697–708.

[22] Z. Feng, M. Cai, and Q. Deng, "An efficient pro-active fault-tolerance scheduling of IEEE 802.1 Qbv time-sensitive network," *IEEE Internet Things J.*, vol. 9, no. 16, pp. 14501–14510, Aug. 2022.

[23] Y. Zhou, S. Samii, P. Eles, and Z. Peng, "Reliability-aware scheduling and routing for messages in time-sensitive networking," *ACM Trans. Embedded Comput. Syst.*, vol. 20, no. 5, pp. 1–24, 2021.

[24] F. Pozo, G. Rodriguez-Navas, and H. Hansson, "Schedule reparability: Enhancing time-triggered network recovery upon link failures," in *Proc. IEEE 24th Int. Conf. Embedded Real Time Comput. Syst. Appl. (RTCSA)*, 2018, pp. 147–156.

[25] W. Kong, M. Nabi, and K. Goossens, "Run-time recovery and failure analysis of time-triggered traffic in time sensitive networks," *IEEE Access*, vol. 9, pp. 91710–91722, 2021.

[26] Z. Zhou and G. Shou, "An efficient configuration scheme of OPC UA TSN in industrial Internet," in *Proc. Chin. Autom. Congr. (CAC)*, 2019, pp. 1548–1551.

[27] A. Kostrzewa and R. Ernst, "Achieving safety and performance with reconfiguration protocol for ethernet TSN in automotive systems," *J. Syst. Architect.*, vol. 118, Sep. 2021, Art. no. 102208.

[28] A. Berisa *et al.*, "AVB-aware routing and scheduling for critical traffic in time-sensitive networks with preemption," in *Proc. 30th Int. Conf. Real Time Netw. Syst.*, 2022, pp. 207–218.

[29] L. L. Bello, M. Ashjaei, G. Patti, and M. Behnam, "Schedulability analysis of time-sensitive networks with scheduled traffic and preemption support," *J. Parallel Distrib. Comput.*, vol. 144, pp. 153–171, Oct. 2020.

[30] D. Maxim and Y.-Q. Song, "Delay analysis of AVB traffic in time-sensitive networks (TSN)," in *Proc. 25th Int. Conf. Real Time Netw. Syst.*, 2017, pp. 18–27.

[31] V. Gavriluţ and P. Pop, "Scheduling in time sensitive networks (TSN) for mixed-criticality industrial applications," in *Proc. 14th IEEE Int. Workshop Fact. Commun. Syst. (WFCS)*, 2018, pp. 1–4.

[32] L. Zhao, P. Pop, Z. Zheng, and Q. Li, "Timing analysis of AVB traffic in TSN networks using network calculus," in *Proc. IEEE RTAS*, 2018, pp. 25–36.

[33] A. Alnajim, S. Salehi, and C.-C. Shen, "Incremental path-selection and scheduling for time-sensitive networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2019, pp. 1–6.

[34] M. Vlk, Z. Hanzálek, K. Brejchová, S. Tang, S. Bhattacharjee, and S. Fu, "Enhancing schedulability and throughput of time-triggered traffic in IEEE 802.1Qbv time-sensitive networks," *IEEE Trans. Commun.*, vol. 68, no. 11, pp. 7023–7038, Nov. 2020.