IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier xxxxxx/ACCESS.xxxx.DOI

Run-time Recovery and Failure Analysis of Time-Triggered Traffic in Time Sensitive Networks

WEIJIANG KONG, MAJID NABI, and KEES GOOSSENS.
Eindhoven University of Technology, Eindhoven, 5612 AZ Netherlands (e-mail: {w.kong, m.nabi, k.g.w.goossens}@tue.nl)

Corresponding author: Weijiang Kong

This research was supported through PENTA project HIPER 181004 on high performance vehicle computer (HPVC) and communication system for autonomous driving.

ABSTRACT Reliability is one of the major concerns of Time Sensitive Networking (TSN). Current systems mostly rely on static redundancy to protect functionality from permanent component failures. This greatly increases the cost of Time-Triggered (TT) flows. Instead, Software Defined Networking (SDN) enables dynamic redundancy. Disrupted traffic can be rerouted by a centralized controller to reduce the cost while maintaining reliability. This paper presents an approach to compute alternative paths at run-time and analyze their impact on reliability. We define a novel three-mode recovery scheme, which includes full functionality, reduced functionality, and emergency halt modes. Run-time recovery for TT flows is explored using Integer Linear Programming (ILP) and a heuristic algorithm. Then, a Markov chain-based design-time reliability analysis is developed to evaluate the Mean Time to Reduced Functionality Mode (MTTRF) and Mean Time to Failure (MTTF) of run-time recoverable systems. Our experiments show that run-time recovery provides better protection against multi-point failures than static redundancy. Compared with the state of the art, our proposed ILP has better routing efficiency. The proposed heuristic algorithm can perform routing and scheduling in polynomial time, but it tends to route multicast flows to longer paths than ILP. Furthermore, when applied to realistic recovery scenarios, our proposed ILP improves the MTTF by up to $2\times$ and the average execution time by up to $20\times$ than the raw ILP of the state of the art. Although less efficient with multicast flows, the heuristic algorithm achieves similar reliability as the ILP, and its worst-case recovery time is below $100ms$ on an embedded ARM processor.

INDEX TERMS Network reliability, Run-time recovery, Times-sensitive networking.

I. INTRODUCTION

The communication bandwidth demand of the emerging autonomous driving technology has encouraged innovations in next-generation vehicle networks. While switched Ethernet is considered as a promising solution, it is not originally designed for real-time safety-critical systems and requires enhancements for bounded latency and reliability. This results in a set of amendments of the Ethernet standard named Time Sensitive Networking (TSN) [1]. To support safety critical control applications, TSN strictly specifies that recovery time must be less than $100ms$ [2].

TSN supports seamless redundancy with Frame Replication and Elimination for Reliability (FRER) [3]. According to FRER, frames are replicated at source and transmitted through separate paths; duplicates are eliminated at destinations. While single-point failures are eliminated, FRER introduces a significant overhead since it requires at least twice the bandwidth. Instead, earlier practices in Software Defined Networking (SDN) [4] point out a more efficient solution [5], [6]. If network has a logically centralized controller possessing knowledge of its flows, it will be able to compute alternative configuration at run-time to recover flows from failures. Therefore, bandwidth of the backup paths does not have to be reserved but can be assigned to non-safety critical traffic until needed. If designed properly, recovery also tolerates more multi-point failures than static protection.

Today's intelligent vehicles employ cross-layer approaches [7] to enhance reliability, which means fault handling mechanisms on different layers (e.g., TSN, computa-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3092572, IEEE Access

tion hardware, applications, etc.) can collaborate to perform graceful degradation. Therefore, TSN has the option to inform its applications when facing unrecoverable failures and cooperate with their fault handling mechanisms to operate or shutdown the system safely.

Problems remain to be solved to bring run-time recoverable TSN into reality: 1) the recovery scheme in which TSN gracefully propagate failures to its applications is yet to be defined; 2) run-time recovery requires computation of routing and scheduling within a *100ms* deadline, which existing solutions still cannot satisfy; 3) a system-level reliability analysis at design-time to evaluate the impact of specific recovery algorithms on reliability is missing.

In this paper, our exploration of run-time recovery is limited to Time-Triggered (TT) traffic on TSN. We target a Time Sensitive Software Defined Networks (TSSDN) architecture [8], which provides the necessary software-defined features for run-time recovery. It has been formally verified that, with careful optimization, TSN configuration latency lower than a millisecond is possible [9]. Thus, we assume that the real-time configuration is provided. This paper focus on the prior steps of building an integrated run-time recoverable system, which is to prove the feasibility of run-time routing and scheduling within the deadline as well as analyzing the reliability gain of run-time recovery. We specifically target TSN used in in-vehicle networks, where the network scale and the number of flows to be supported is relatively limited. The contributions of this paper are as follows.

- We define a novel recovery approach for TT traffic. It consists of three modes: full functionality mode, reduced functionality mode, and emergency halt mode. Run-time recovery attempts to maintain the current functionality mode. When failures are not recoverable, system-wide functionality degradation is performed through mode switching. (§3-4)
- We develop faster methods to compute alternative routes and schedules in response to failures. Our Integer Linear Programming (ILP) solution produces near optimal results at design-time. We minimize its number of variables and linear constraints to substantially reduce the solving time and introduce slots prioritization as well as destination forwarding to increase it routing capacity. But it may take longer than the recovery deadline ($100ms$) making it unsafe for use at runtime. (It does provide a baseline to which the run-time implementation can be compared). Our polynomial time heuristic algorithm instead finds feasible routes and schedules within the deadline using resources available on embedded platforms. But it tends to use more links than ILP when routing multicast flows. (§5)
- We develop a system-level reliability analysis for our recovery and degradation process. Given system specifications and recovery approaches, the system's Mean Time to Reduced Functionality Mode (MTTRF) and Mean Time to Failure (MTTF) are calculated by evaluating the recovery behavior for possible failure sequences. (§6)
- We perform a comparative study of the proposed recovery approaches, a state of the art routing & scheduling approach [10], and FRER using randomized testing and the proposed reliability analysis tool. Experiments are conducted on 40 synthetic topologies to demonstrate that run-time recovery is more reliable against multi-point failures than FRER. Compared with the state of the art, our proposed ILP can setup more flows on a same network. The proposed heuristic algorithm achieves similar efficiency with ILP for unicast flows. But it is less efficient in handling multicast flows. Using additional 120 more realistic test cases, our experiment shows that the proposed recovery approaches can result up to twice the mean time to failure than the state of the art; the proposed ILP by its average can setup flows within a second; and the heuristic algorithm meets the $100ms$ worst-case execution time requirement. Finally, to demonstrate that the proposed approaches are feasible in realistic network configurations, we perform a case study of an automotive TSN to show that the execution time of the proposed heuristic algorithm is well below the recovery deadline. (§7)

II. RELATED WORK

Various techniques for fault recovery have been studied on switched Ethernet. SDN enables segment protection which creates static redundancy [6], [11]. It features at low response time and can be combined with dynamic path configuration which is not real-time to improve the average recovery delay. However, such techniques cannot be applied to TSN which requires worst-case recovery time guarantee. Recovery is also studied in OpenFlow [5], [12]. These research focus on the design of network architecture and use build-in routing algorithm from the network controller. Because OpenFlow networks require nationwide scalability, they conclude that carrier grade recovery is hard to achieve. However, TSN is a scenario where scalability is of less concern. Moreover, high redundancy cost, high determinism of traffic and powerful computation resources on-board together makes TSN a feasible scenario for run-time recovery.

Flow recovery connects to the well-researched routing and scheduling problem of TSN. Here we consider only TT traffics. TSN scheduling focuses on assigning the transmission duration to flows that traverse determined paths. Satisfiability Modulo Theories (SMT) is a well-developed and widely used approach for this problem [13]–[16]. Latest research [17] proposes window-based heuristic scheduling which relaxes flow isolation but still provides real-time guarantee for flows. However, flow routes must be provided for scheduling whereas recovering link failures requires finding both the alternative routes and schedules. Therefore, a scheduling mechanism that relies on additional routing methods alone is not sufficient for solving the run-time recovery problems. A joint routing and scheduling technique solves flow routing and scheduling as a single optimization problem, for which ILP [18], [19] is a widely-adopted design-time approach. It offers better routing quality (e.g., lower latencies) compared

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3092572, IEEE Access

with scheduling on fixed paths [18] but also demands significant run-time and resources. As an example, the ILP in [18] takes more than a minute to route 30 flows on a server-grade processor. To solve the joint routing and scheduling problem for online usage, incremental approaches [10] have been developed. Routing and scheduling for online usage is the closest scenario to run-time recovery. Thus, the work in [10] is the state of the art with which our solutions will be compared. However, since it is not strictly real-time, the state of the art is not efficient enough for recovery: their raw ILP formulation takes more than 10s per flow; and to achieve sub-seconds average execution time, they introduce optimizations involving reattempts on which recovery cannot rely. Instead, our ILP solution focuses on directly optimizing the raw ILP formulation to achieve sub-seconds execution time. Recent research also addresses machine learning as an accurate approach for the schedulability analysis and verification of TSN [20], [21]. However, issues such as "false positives" [20] still remain to be solved before it can be applied to run-time recovery.

A survey on functional safety of the Ethernet-based communication solutions can be found in [22]. Existing reliability analysis for self-recoverable networking has limitations to be applied to self-recoverable TSN. [23] uses Markov state reward model for SONET mesh networks by considering link restoration with a measurable restoration rate. However, flow recovery requires more in-depth analysis of the systems. [24] uses discrete-event simulation to analyze a binary sibling tree network. But it has the risk to over-estimate system's reliability and is thus not suitable for safety critical systems. [25] considers the reliability of self-healing networks from the perspective of tasks. It is based on the general assumptions that the alternative path can be found if a route exists, which omits the impact of recovery mechanism and might significantly overestimate the system reliability.

III. SYSTEM OVERVIEW

In this section, we describe our run-time recoverable system, following a detailed introduction of the TSSDN's scheduling model. We model a TSN-based system as a directed graph $G \equiv (V, E)$. The set of vertices V contains end stations (V_{ES}) such as Electronic Control Units (ECU), sensors, actuators, and switches (V_{SW}), i.e. $V = V_{ES} \cup V_{SW}$. E represents simplex links connecting network vertices. Since TSN requires bidirectional communication, each pair of connected vertices is connected by at least two opposite edges. Direct links between end stations are allowed. *Both switches and end stations can forward flows while end stations additionally can generate and consume flows.*

A. SCHEDULING MODEL OF TSSDN

According to IEEE 802.1Qbv [26], TT flows are cyclic and shaped by Time Aware Shapers (TAS) on egression ports. TAS synchronously grant transmission to traffic queues based on a predefined schedule in Gate Control Lists (GCL), which consist of ordered operations to open or close transmission gates for each port. TSSDN uses the same hardware basis but imposes an extra non-queuing transmission constraint to ensure timing isolation as well as ultimate low delay and jitter [8]. Non-queuing transmission requires to reserve consecutive links so that flows can be forwarded from sources to destinations without blocking. To avoid collisions, TSSDN uniformly divides large TAS periods into smaller synchronized time slots. As shown in Fig. 1, different TT flows are not allowed to be scheduled on the same slot of the same link. As a result, each egress port will receive only one frame during each time slot. And that frame will be transmitted before the time slot ends. Such specification dispel worries about timing interference, such as cyclic dependency caused during routing [27].

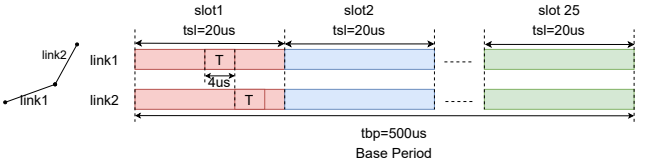


FIGURE 1. Schedule of TSSDN. Durations with 'T' are for TT flows. The rest of each slot is assigned to other flow types.

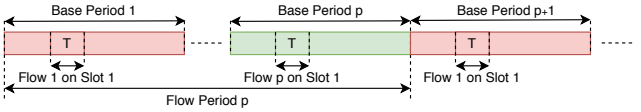


FIGURE 2. Example of p flows sharing the same slot

In TSSDN, the period of TAS is referred to as Base Period (BP). Its duration t_{bp} must be smaller than the minimum transmission period of TT flows so that flow periods can be rounded down to its nearest integer multiple for scheduling. Meanwhile, the duration of time slot t_{sl} must be larger than the non-queuing end-to-end delay of a maximum-sized frame. b represents the number of slots in one BP, i.e. $t_{bp} = b \times t_{sl}$. And $B = \{1, ..., b\}$ represents the set of all time slots in one BP for all links available for scheduling. TSSDN scheduling only involves specifying the time slot. Reserving links and generating GCL are automated [28].

F represents the set of flows to be supported at a specific moment. A flow $f \in F$ consists of a specification and an implementation, i.e. $f = (sp, ip)$. The specification sp defines single-frame transmissions from a source $s \in V_{ES}$ to a set of destination $D \subseteq V_{ES}$ with period pt. Since pt is rounded to $p = \left\lfloor \frac{pt}{t_{bp}} \right\rfloor$ during scheduling, we consider p instead of pt in flow specifications, i.e. $sp = (s, D, p)$.

Phase is the transmission offsets that allow flows to overlap on the same slot of the same link. Since a flow with period p transmits every p BPs, it can share slot sc with at most $p - 1$ other flows of the same period marked by different phases [10]. This is shown in Fig. 2. We represent the set of phases for flows whose periods are p by $P = \{1, ..., p\}$. The implementation $ip = (RO, sc, ph)$ consists of the route, time

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3092572, IEEE Access

slot and phase of a flow. The route $RO \subseteq E$ is a cycle-free set of links. $sc \in B$ is the slot of transmissions and ph is the phase. If a flow is currently under disruption and does not have a valid implementation, it is referred to as $ip = \emptyset$.

TSSDN conforms to the Control Data Traffic (CDT) specification of TSN in [29]. Frames can have maximum size of 128 bytes. The minimum interval between frames is $500\mu s$. Flows can traverse at most 5 hops and require end-to-end delay of less than $100\mu s$. Thus, network diameter must be limited, e.g., to seven hops as required by IEEE 802.1AS time synchronization protocol [30].

B. MODEL OF SELF-RECOVERABLE SYSTEMS

Today's intelligent in-vehicle networks are designed with domain-based pattern where different applications are supported in different domains. Due to their strict timing requirement, TT flows must be constrained within the control domain fully equipped with TSN-capable switches. Therefore, recovery can only be performed within a fraction of the system. In fact, both the standard [30] and TSSDN [10] suggest the network diameter to be limited within seven hops. Due to expensive bandwidth overhead, network designers also tend to minimize the amount of TT flows. The realistic systems evaluated in recent studies [17], [29], [31] indicate that there can be less than twenty TT flows in relatively small TSN use cases.

Centralized network control: The network control in TSSDN is centralized. When the recovery controller receives notification on failures, it interprets the disrupted flows and computes their alternative routes and schedules based on the global knowledge of the network. Then, it sends update messages to either recover the disrupted flows without changing running flows or initiates a system degradation by switching to a mode with less functionality. Previous study [9] has formally verified that the network configuration latency is well below the required recovery deadline. For instance, with the explicit flow configuration protocol where SDN traffic are mapped to highest priority and number of SDN messages per switch is limited to one, the upper bound of configuration latency can be reduced to $0.6ms$ which is independent from the size of the configuration message [9].

Failures: Run-time recovery targets long-lived or permanent failures of the network components. Transient failures such as packet loss due to noise should be handled by other fault tolerant mechanisms. When failures are detected, the messages sent to the recovery controller only signal link failures. The vertex failures can be taken as the failure of all its attached links. Upon arrival of the first failure message, the recovery controller keeps monitoring messages given a worst-case timeout. Thus, all failure messages can be received in case of vertex failures or multi-link failures.

Functionality: The growing complexity of today's autonomous driving systems demands cross-layer reliability approaches. Different from the traditional fail-operational networks that must handle failures within their own scope, TSN has the option to propagate failures to the application layer when necessary and collaborate with its failure handling mechanisms. For example, a flow carrying 60 fps video stream of a camera might not be recoverable from a link failure because the redundant links in the network do not have enough bandwidth. Therefore, the network controller informs the camera to reduce its frame rate to 30 fps for a successful reroute. Furthermore, the vehicle now must limit its speed and other applications may also change their flows accordingly. Similar but more complex examples of such design can be found in [32].

Negotiating system-wide flow requirement upon degradation is very likely to cause a recovery delay violation. Instead, a more suitable solution would be specifying the degradation behavior as a part of the Service Level Agreement (SLA), which allows pre-configuration. We generalize such system behavior as *functionality modes*. Each functionality mode is a set of flows to be supported. We refer to the mode in which full designed functionality is performed as the *full functionality mode*, and the mode to which the system switches when full functionality cannot be supported as *reduced functionality mode*. F^{fm} is the set of flows for full functionality mode, and F^{rm} is the set of flows for reduced functionality modes. When the reduced functionality mode cannot be supported due to failures, the system enters *emergency halt mode* to avoid catastrophic behavior. The emergency halt mode does not require setting up new flows, i.e. applications should utilize the surviving flows to perform safe shutdown.

While designing flows in different functionality modes is a multi-discipline question involving co-design of TSN and safety-critical applications, we focus on the recovery of TSN in this paper. Generally, TSN does not impose any constraints on the flows of these modes. But to make the protection valid, F^{rm} should require fewer network resources than F^{fm} (fewer flows and/or lower frame rate per flow). Like fast failover mechanisms in OpenFlow [33], our mode-switching degradation requires the reduced functionality mode configuration to be pre-installed in switches. If switches can support multiple pre-installed configurations, the network can have multiple reduced functionality modes. However, for simplicity and w.l.o.g., we consider only one reduced functionality mode in this paper.

FRER: Run-time recovery requires flows to tolerate a recovery delay during which packet will be lost. Such packet losses need to be captured by the application-level fault tolerance mechanisms. For flows that cannot tolerant recovery delay or packet losses, FRER must be applied to guarantee reliability. In this scenario, run-time recovery can be a supplementary protection for the redundant paths. These flows will not fail if at least one of their replicas is still running, regardless of whether the disrupted replicas are recoverable. However, if the last running replica is disrupted, or all replicas are disrupted simultaneously, the system must degrade immediately without recovery attempt.

Given a set of flows F, we refer to its subset of FRER flows as F^f and its subset of non-FRER flows as F^n. For every FRER flow $f \in F^f$, its replica is identified as

$f' \in F^f$. f and f' are identical in specification but different in implementation. To avoid being disrupted by the same link failure, their routes cannot share links. Additionally, the maximum tolerable jitter jt between two replicated frames must be specified in their specification, i.e. $\forall f \in F^f$, $sp = (s, D, p, jt)$.

IV. PROPOSED RECOVERY PROCESS

In this section, we describe the forwarding and management operations of the network controller and vertices to perform a reliable recovery.

Initialization: On startup, the recovery controller queries each network vertex for the topology. Based on the given time slot and flow specification, it computes two GCL configurations for each network vertex, one for the full functionality mode and one for the reduced functionality mode. Both configurations are then transmitted to the corresponding network vertices. Full functionality configuration is immediately set up on regarded ports and becomes ready for incoming traffic. Meanwhile, reduced functionality configuration is stored in a fallback GCL. It has no effect on traffic, but the ports can switch to fallback GCL following reconfiguration messages.

Usually TT flows are static. Configurations for dynamic and transient flows can be added to the network incrementally by the network controller. Resources for dynamic flows must be reserved based on their worst-case assumptions. Before transmission, transient flows send transient requests for which a timeout is specified to free the resources. Before the timeout is reached, transient flows are served as static flows in the current functionality for recovery. To avoid the transient requests disrupting the recovery process, they must have lower priority in transmission and processing, resulting in potentially higher startup delay of the transient flows compared to the recovery process.

Failure processing: When link failures are identified, network vertices block the affected ports and send failure messages to the network controller. Delayed frames cannot be used by the safety-critical applications due to their timing violation. But they still carry information about the failures and thus are valuable for reliability mechanisms. For instance, if ECUs detect that an actuator is not responding, knowing that packets are blocked due to network failures indicates that the actuator might still be functioning leading to different handling procedure. Hence, the blocked frames can be redirected to nearby ports to be forwarded to their source, destinations, or the network controller depending on their applications. They can be forwarded as Best-Effort (BE) or Audio Video Bridging (AVB)-A traffic depending on the exact reliability mechanisms.

Reconfiguration: The recovery controller collects all failed links $E_f \subset E$ including those involved in earlier recoveries, and computes alternative implementations for affected flows F_D whose routes have at least one failed link (i.e., $F_D = \{f \in F \mid RO \cap E_f \neq \emptyset\}$). Based on the alternative implementation, reconfiguration messages are generated. There are two scenarios.

- When maintaining current mode, the reconfiguration message contains instructions to remove invalid GCL entries on old routes and add alternative entries to new routes. During this process, non-disrupted flows are forwarded normally.
- When changing mode, the reconfiguration message for a network vertex contains instructions to remove invalid fallback GCL entries on old routes and add alternative entries to new routes. During this process, the ongoing mode is supported in a fail-operational manner. As the fallback GCL is updated, an activate instruction within reconfiguration message causes the network vertex to switch to the fallback GCL for forwarding.

Note that emergency halt does not involve any GCL switching on network vertices. The network controller informs tasks about the existing failures. And it is up to the tasks to shutdown using their surviving flows.

Verification: Before reconfiguring the network, the generated implementation must be verified. We implement a simple rule-based verification for our recovery controller which checks: 1) if each route exceeds the maximum of hops, 2) if each route is connected and cycle-free, 3) if the schedule has conflicts between flows. Implementation that fails to be verified signals a failure of the recovery controller and results in instant system halt.

V. ROUTING AND SCHEDULING FOR RECOVERY

Upon reception of the failed links E_f, the recovery controller marks the implementation of all disrupted flows F_D as invalid. It then processes flows in F_D one by one. We use a deterministic processing order to ensure analyzability. Attempting recovery with non-deterministic order, e.g., randomly selecting the next flow to be processed, will cause additional complexity for reliability analysis. In this section, we introduce an ILP and a heuristic algorithm to compute alternative routes and schedules.

To recover a disrupted flow, the following inputs are used.

- Network topology $G = (V, E)$
- Set of all time slots $B = \{1, ..., b\}$
- Set of all failed links E_f so far
- Selected flow to be recovered $f \in F$
- Set of phases that f can use $P = \{1, ..., p\}$
- Specification and implementation of all flows in F

A. ILP SOLUTION

ILP has already been used in TSSDN to set up flows on a fixed topology [10]. Since it produces near optimal results, we develop an ILP solution to indicate the maximum reliability achievable by recovery. Compared with ILP in [10], our proposed solution features at handling an extended routing and scheduling problem with dynamic failures in the topology as well as FRER flows. *We propose to express the problem using fewer variables. And the correlations between these variables can be mostly resolved as upper and lower bounds of the variables instead of linear constraints.* Thus,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3092572, IEEE Access

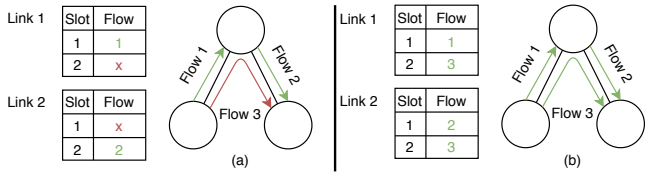


FIGURE 3. An example of slot prioritization.

the solving time of our ILP formulation is greatly reduced. To increase the routing capability of our ILP, we propose *slots prioritization* and *destination forwarding*. Our main objective is the same as [10] which is to minimize flows' route length, because it has been shown that the shortest routing path utilizes network resources more efficiently.

Here are the variables used in our ILP solution.

- Flow counter $c_i \in \mathbb{N}$, for $i \in E$. c_i indicates the number of destinations flow f reaches through link i. If i is not used by f, then $c_i = 0$.
- Slot and phase selection $h_{j,k} \in \{0, 1\}$, for $j \in B, k \in P$. $h_{j,k} = 1$ if and only if f uses slot j and phase k.
- Link-wise schedule $l_{i,j,k} \in \{0, 1\}$, for $i \in E, j \in B, k \in P$. $l_{i,j,k} = 1$ if and only if f is scheduled on link i, slot j and phase k.

The objective function is shown in Eqn. 1. pi_j is the priority for slots $j, \forall j \in B$.

$$Min \left\{ \sum_{\forall i \in E, j \in B, k \in P} l_{i,j,k} + \sum_{\forall j \in B, k \in P} \frac{pi_j \cdot h_{j,k}}{b+1} \right\} \quad (1)$$

The primary objective (first term) minimizes the length of path for f to minimize its network load. The secondary objective (second term) specifies slots prioritization in which each time slot is assigned a fixed priority as its preference of being selected for scheduling (slots with lower priority first). Slots prioritization urges the solution to concentrate flow scheduling to avoid false blocking. An example of such false blocking is shown in Fig. 3. In Fig. 3(a), flows 1 and 2 are scheduled on the shortest path but the randomly picked slots block flow 3. Prioritization forces flows 1 and 2 to use slot 1 in Fig. 3(b), so flow 3 can use slot 2. We define $pi_j = j$ so slots represented by smaller number are preferred. Since selected slot and phase are unique, $\sum_{\forall j \in B, k \in P} pi_j \times h_{j,k} \leqslant b$. Thus, the secondary objective divided by $b + 1$ will be effective only when the primary objective is minimized.

General constraints: Constraints for all flows are:

1. A flow selects a single slot and a single phase.

$$\sum_{\forall j \in B, k \in P} h_{j,k} = 1 \quad (2)$$

2. Flow conservation constraints ensure the validity of the route. A valid route must be continuous starting from the source and passing all destinations free of cycles. $i(v)$ and $o(v)$ stand for input and output links for vertex v. Eq. 4 allows destination forwarding in which route could pass a destination to each other destinations. It makes maximum use of the redundant links connecting end stations to route flows. Note that the network interface of these end stations must be designed with forwarding capability to enable destination forwarding.

$$\sum_{\forall i \in i(s)} c_i = 0\,, \quad \sum_{\forall i \in o(s)} c_i = |D| \quad (3)$$

$$\forall\, v \in D: \quad \sum_{\forall i \in i(v)} c_i - \sum_{\forall i \in o(v)} c_i = 1 \quad (4)$$

$$\forall\, v \notin \{s\} \cup D: \quad \sum_{\forall i \in i(v)} c_i - \sum_{\forall i \in o(v)} c_i = 0 \quad (5)$$

3. Link-wise route and schedule constraints ensure $h_{j,k} = 1$ if at least one link is used and $l_{i,j,k}$ is not all 0.

$$\forall i \in E, j \in B, k \in P: \quad l_{i,j,k} - h_{j,k} \leqslant 0 \quad (6)$$

$$\forall i \in E: \quad c_i - \sum_{\forall j \in B, k \in P} |D| \cdot l_{i,j,k} \leqslant 0 \quad (7)$$

4. Collision avoidance constraints for every flow $f_n \in F$ with period p_n and valid $ip_n = (RO_n, sc_n, ph_n)$ are as follows.

$$if\ p = p_n \rightarrow\ \forall i \in RO_n: \quad l_{i,sc_n,ph_n} = 0; \quad (8)$$

$$if\ p \neq p_n \rightarrow\ \forall i \in RO_n, k \in P: \quad l_{i,sc_n,k} = 0; \quad (9)$$

5. Failed links must be avoided.

$$\forall i \in E_f, j \in B, k \in P: \quad l_{i,j,k} = 0 \quad (10)$$

FRER constraints: Constraints for FRER flows are:

6. If $f \in F^f$ with replica f', ensure that their paths are disjoint.

$$\forall i \in RO', j \in B, k \in P: \quad l_{i,j,k} = 0 \quad (11)$$

7. If $f \in F^f$ with replica f', ensure replicas arrive in tolerated jitter range, where JC is the feasible slot and phase pairs $(j \in B, k \in P)$ that satisfy Eq. 13.

$$\forall (j,k) \notin JC: \quad h_{j,k} = 0 \quad (12)$$

$$|(k - ph') \cdot b + (j - sc')| \leqslant \left\lfloor \frac{jt}{t_{sl}} \right\rfloor \quad (13)$$

Constraints 4, 5, 6, and 7 are encoded into the upper and lower bound constraints of the solver (both upper and lower bounds of the variables are 0). As a result, FRER, existing flows, and failures do not change the number of variables and constraints but require extra efforts to form the upper and lower bounds. Compared to using linear constraints, it reduces the search effort and helps to quickly identify the existence of feasible solutions. The rest of the constraints are linear constraints. Our formulation uses $|E| \times |B| \times |P| + 2 \times |E| + |V| + 2$ linear constraints. In comparison, the ILP in [10] uses $4 \times |E| \times |B| \times |P| + |V| + |D| + |E| + 2$ linear constraints. Since $|E| \times |B| \times |P|$ is magnitudes larger than remaining factors, the # of constraints is reduced by approximately 75%.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3092572, IEEE Access

Algorithm 1 Heuristic Routing & Scheduling

for all j in B **do**
  for all k in P **do**
    $Gr_{j,k} = residualnw(G, ST, PT, E_f, j, k)$
    $ro_{j,k} = shortestpath(Gr_{j,k}, s, D)$
    if $ro_{j,k}$ is currently shortest **then**
      $ip = (ro_{j,k}, j, k)$
    end if
  end for
end for
return ip if found

B. HEURISTIC SOLUTION

Instead of solving routing and scheduling as a uniform optimization problem, they can also be viewed as separated steps. Consider a given schedule (j, k) where $j \in B, k \in P$, the problem is reduced to routing a multi-destination flow among the free links $Er_{j,k} \subseteq E$ (links that are neither scheduled on (j, k) nor failed). This is the well-known Steiner tree problem [34]. Although the problem is NP-complete [35], there are polynomial-time heuristic algorithms to find near optimal solutions.

Besides inputs defined earlier, our heuristic algorithm requires keeping track of two additional variables along with the flow implementation, which avoids checking flows to identify link availability. However, they must be updated every time a flow implementation is changed.

- Slot table $ST = \{st_{i,j}\}$, for $i \in E$, $j \in B$. $st_{i,j}$ is the period of the flows scheduled on slot j of link i. Only flows with the same period can share slots. $st_{i,j} = 0$ if it is unused.
- Phase table $PT = \{pt_{i,j,k}\}$, for $i \in E$, $j \in B$, $k \in \{1, ..., st_{i,j}\}$. $pt_{i,j,k} = 1$ means phase k is occupied on slot j of link i.

Our heuristic algorithm iterates through all pairs of slots and phases to find the one that minimizes the route length. To ensure fast execution, we compute the shortest path tree from source to all destinations as an approximate solution to the Steiner tree problem. This is shown in Algorithm 1.

In Algorithm 1, *residualnw()* computes the sub-network $Gr_{j,k} = (V, Er_{j,k})$ of G formed by free links. A link $i \in Er_{j,k}$ if it satisfies Eq. 14. $Gr_{j,k}$ and G always have the same vertices. It is up to the routing algorithm to handle vertices isolated by failures and occupied links if they exist.

$$(i \notin E_f) \wedge \{st_{i,j} = 0 \vee [(st_{i,j} = p) \wedge (pt_{i,j,k} = 0)]\} \quad (14)$$

Procedure *shortestpath()* computes shortest path tree connecting source and all destinations of f on $Gr_{j,k}$. Instead of finding the shortest tree that connects the source and all the destinations, this algorithm finds the shortest route from source to each of the destination. So, it can be performed by the Dijkstra algorithm with $O(|Er_{j,k}| + |V|\log|V|)$ complexity [36]. Thus, the overall complexity for routing and scheduling is $O(|B| \times |P| \times (|E \setminus E_f| + |V|\log|V|))$.

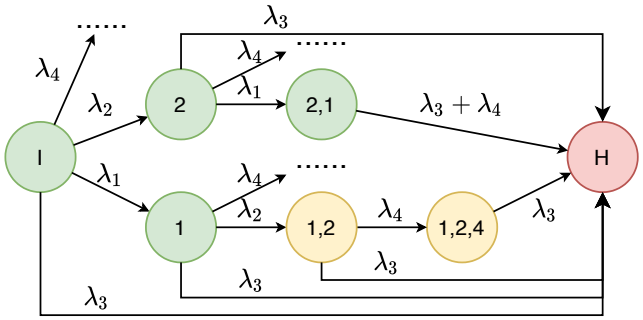


FIGURE 4. CTMC of a 4-link network. Colors are: green for full, yellow for reduced, red for halted functionality modes. λ_i is the failure intensity of link i. Node labels indicate the failure sequences.

This polynomial complexity comes at the cost of its solution quality. While it still provides shortest route for unicast flows, its solution will potentially cost more links than an optimal Steiner tree for multicast flows.

Slot prioritization can be implemented when deciding if the route found is currently shortest. First, the recorded solution ip is compared with $ro_{j,k}$ on the length of the route. If they are the same, the selected time slots are compared. The solution whose time slot has higher priority is recorded as the best solution.

Algorithm 1 only requires slight modification to be applied to FRER flows. If its replica f' has a valid implementation, (j, k) must be selected from JC, which is calculated as in constraint 7 in §5.*a*. Also, RO' must be excluded from G_r. Thus, the overall complexity for processing an FRER flow is $O(\lfloor jt/t_{sl} \rfloor \times (|E \setminus \{E_f \cup RO'\}| + |V|\log|V|))$.

VI. CTMC-BASED RELIABILITY ANALYSIS

The life cycle of run-time recoverable TSN can be modeled by a Continuous Time Markov Chain (CTMC) since the flow implementation after recovery only depends on the failure and flow implementation in the previous state of the network.

Failure events: Long-lived or permanent component failures that can possibly occur to the system can be modeled as failure events. Each failure event consists of a set of failed components associated with a distribution function. To make CTMC analysis valid, all failure events must be independent. The coverage of the failure events determines the precision of the analysis.

State space generation: Given a system and all its failure events, state space generation takes the failure events and generates a CTMC exploring all sequences of successive failures events. Fig. 4 shows an example of our CTMC considering only link failures. For a given failure sequence, it starts from the initial full functionality state, removes the first failed link and recovers the system into a new state. *The analysis applies the recovery approach under evaluation to determine the functionality of the new state based on whether the previous failures are recoverable.* This process is repeated until the system enters the halt mode (when reduced

functionality cannot be sustained). The system cannot be restored from the halt mode by itself, as it is shut down for maintenance in the real life.

System reliability: Given the CTMC generated above, the distribution of states can be calculated using an analytical approach [37] with outputs:

- $\pi_f(t)$ is the probability that the system is in full functionality mode at time t.
- $\pi_r(t)$ is the probability that the system is in full or reduced functionality mode at time t.

We evaluate different recovery approaches with two metrics, MTTRF and MTTF. MTTRF is the mean time that the system stays in the full functionality mode, i.e. the average time since system start up till the first functionality degradation happens. It indicates the frequency which the network requires for maintenance. MTTF is the mean time before the emergency halt, which reflects the lifetime of the system. They can indicate the reliability of the systems as well as the maintenance frequency (cost). Therefore, both metrics reflect the quality of recovery approaches. While our work mainly focuses on finding recovery approaches that meets the execution time requirement, improvement of recovery quality is preferred. And significant loss of reliability must be avoided. MTTRF and MTTF are calculated by Eq. 15 and 16.

$$MTTRF = \int_0^\infty t \cdot (1 - \pi_f(t))' d_t \tag{15}$$

$$MTTF = \int_0^\infty t \cdot (1 - \pi_r(t))' d_t \tag{16}$$

Optimization: For large or highly redundant networks, the CTMC generated can be huge. Thus, we apply following optimization to reduce the cost of computation:

- The sequences are enumerated in a depth-first order to reduce the memory consumption, e.g., $\{1, 2\}$ is evaluated after $\{1\}$ in Fig. 4, since it requires knowledge of $\{1\}$. After all states reachable from $\{1\}$ are evaluated, the implementation of $\{1\}$ can be deleted from memory.
- High order failures can be assumed to result in the halt mode. This is a conservative approximation as it provides a lower bound of the system reliability. Depending on the timing and precision requirement, designers can chose the maximum number of failures evaluated.
- Merge all halt states into a single absorbing state. Parallel edges created are also merged, e.g., halt is reachable from $\{2, 1\}$ by failure of link 3 or 4 in Fig. 4, and thus can be encoded as a single edge $\lambda_3 + \lambda_4 = 2\lambda$.

VII. EVALUATION

To evaluate the proposed approaches, we implement our proposed ILP (pILP, §5.a) and analysis (§6) in MATLAB which runs on a desktop computer equipped with an Intel(R) Core(TM) i7-9700F 3.0GHz CPU. For comparison, we re-implement the state of the art ILP (SOA) [10] using the same setup. To show that our heuristic algorithm (HRS, §5.b) is ready for embedded deployment, it is implemented in C++ together with the state space generation (§6) and runs on a Cortex-A9 dual-core 650MHz processor on a PYNQ board [38]. The result is post-processed by MATLAB to calculate MTTRF and MTTF. Note that in CDT specification, TT traffic can transmit in maximum 5 hops. Thus, for larger networks, links and nodes beyond the reach of 5 hops can be pruned for each source node to reduce the execution time.

In this section, we comparatively evaluate our proposed approaches with the existing approaches. We compare their routing capability on networks with/without failures using 40 randomly generated topologies to which a limited number of time slots are allocated (§7.a). Then, we generate 120 test cases based on 3 typical redundant topologies with 30-100 flows setup by pILP and analyze the impact of the recovery approaches as well as topologies on reliability (§7.b). The execution time of different approaches captured in this process is evaluated (§7.c). Finally, a case study on an automotive TSN is performed (§7.d).

A. RECOVERY EVALUATION

The network designers tend to allocate just enough time slots for meeting the scheduling and redundancy requirement so that more bandwidth guarantee can be provided to other traffic types. The routing and scheduling approaches that can probably setup more flows given the same amount of time slots are preferred. Randomized testing can be used to evaluate the efficiency of the proposed recovery approaches in comparison with the SOA and FRER. We generate 40 topologies with redundancy using random regular graph, each with 8 end stations. Every end station is attached to 3 links. To show how the algorithms perform when time slots are just enough or even insufficient, the number of time slots per link is set to only 4 (TSSDN can have 40 time slots if necessary [10]). Allocating more time slots results in more flows to be successfully setup in the network. But it does not change the trend demonstrated in the results. On each topology, we carry our experiment with randomly generated flows whose source and destinations are selected by uniform distribution. The period of the flows is set to the BP of the networks.

First, we compare the three approaches (pILP, HRS and SOA) in their capability of setting up flows on failure-free networks. The capacity of the algorithms is indicated by the *acceptance rate* which is *the percentage of different topologies in which the corresponding set of flows can be successfully setup.* Note that even there is only one single flow in the tested set of flows cannot be setup, the topology cannot be counted as accepted. Because FRER relies on these routing and scheduling approaches to be setup, it is excluded from this comparison.

Flow capacity: To test the routing capacity of the algorithms, evaluation is conducted by generating 15 - 50 unicast flows by steps of 5 on each topology, resulting 40×8=320 cases. The quality of the algorithms is measured by the acceptance rate. The result is shown in Fig. 5.(a).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3092572, IEEE Access

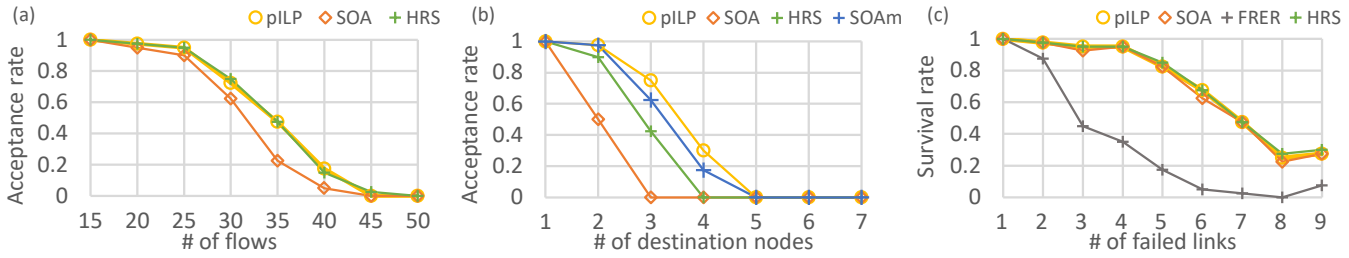


FIGURE 5. Evaluation of different recovery approaches. (a) Acceptance rate versus the number of flows for unicast flows. (b) Acceptance rate versus the number of destinations of 15 multicast flows. (c) Survival rate versus the order of failures.

Generally for all three algorithms, the acceptance rate reduces when the number of flows increases. Compared with SOA, the acceptance rate of pILP reduces more smoothly due to slot prioritizing. For example, when there are 35 flows in the network, pILP can successfully setup 47.5% cases while SOA can only setup 22.5%. In scenarios of unicasting, the Steiner tree problem is reduced to the single destination routing problem. Thus, the acceptance rate of HRS is very close to that of pILP. Notice that pILP and HRS are slightly different as they may chose different shortest path when multiple options exist. Thus, there is a minor variation between the acceptance rate of HRS and pILP in Fig. 5.a. The acceptance rate of all approaches reduces to zero for scenarios with 50 flows. Since there are only 4 time slots on each link, flows exceed the capacity of the network. More flows could be setup if more time slots are allocated. However, the relative behavior of the algorithms will not change.

Multicast flows: To demonstrate how these algorithms behave when setting up multicast flows, evaluation is conducted by generating 15 multicast flows with 1-7 destinations on the topologies, resulting 40×7=280 cases. The quality of the algorithms is again measured by the acceptance rate. The result acquired is shown in Fig. 5.(b). Note that adding more flows to the networks will not change the trend observed in this experiment.

When the flows have more destinations, their paths generally occupy more links. Thus, the acceptance rate decreases. As indicated by the result, pILP again achieves highest acceptance rate, which indicates slot prioritizing is still effective for multicast flows. Due to the approximation in HRS, it finds longer paths than pILP. Hence, the networks are more likely to be saturated, resulting in lower acceptance rate. The original design of SOA does not allow flows to be forwarded through its destinations. To ensure a fair comparison, we run the same test with the original SOA and a modified SOA (SOAm) which has constraints allowing forwarding through destinations. Note that in scenarios of unicasting, flows only need to reach one destination, which means destinations do not forward flows anyway. So, the ILP formulation of SOA and SOAm are exactly the same for unicast flows. Despite the lack of slot prioritizing, SOAm still finds shorter paths for multicasting, thus is more efficient than HRS. In contrast, SOA has the lowest acceptance rate among all the solutions, since it cannot utilize the redundant links at destinations. However, whether forwarding is allowed through destinations is more of a question for the network design than routing, since end stations needs efficient network interfaces. Our experiment show that in networks with redundancy, enabling forwarding in end stations can enhance the capacity of the networks by up to 60% (# of destination nodes=3). Note that multicast flows with more destinations potentially require more links to be setup. When flows have 5 or more destinations, the flows exceed the capacity of the network in which 4 slots are allocated per link. Hence, the acceptance rate drops to 0, although part of the flows with more than 5 or more destinations can be successfully setup.

Then, we compare run-time recovery using different algorithms and FRER under different failure scenarios.

Order of failures: To demonstrate the effectiveness of different approaches in handling failures, we randomly generate sequential failures of 1-9 links on the topologies, resulting 40×9=360 cases. For each case, 10 random unicast flows are setup on the failure-free network. The quality of the approaches is indicated by *survival rate* which is *the percent of cases with the same number of failed links but on different topologies where all disrupted flows can be successfully recovered.* Note that since FRER does not involve recovery, we measure its *percent of cases where all flows have at least one surviving replica.* The result is shown in Fig. 5.(c).

Similar to the conclusion in Fig. 5.(a), the proposed recovery approaches and SOA achieves similar quality in the unicasting scenarios. While FRER can provides 100% protection for single link failures, it is vulnerable to multi-link failures. Its survival rate drops to 87.5% for 2-link failures and 45% for 3-link failures, while survival rate remains higher than 95% for the recovery approaches.

B. SYSTEM ANALYSIS

To compare the impact of topological redundancy on the reliability of run-time recoverable systems, evaluation is performed on three redundant topologies: 8 nodes ring (R08), 10 nodes ring (R10), and 4x2 mesh (M08) with all vertices as end stations, and each link has 16 transmission slots. For simplicity, we only consider link failures with exponential distribution, whose failure rate is $\lambda = 3\times10^{-9}h^{-1}$ for every link as required by Automotive Safety Integrity Level D [39].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3092572, IEEE Access

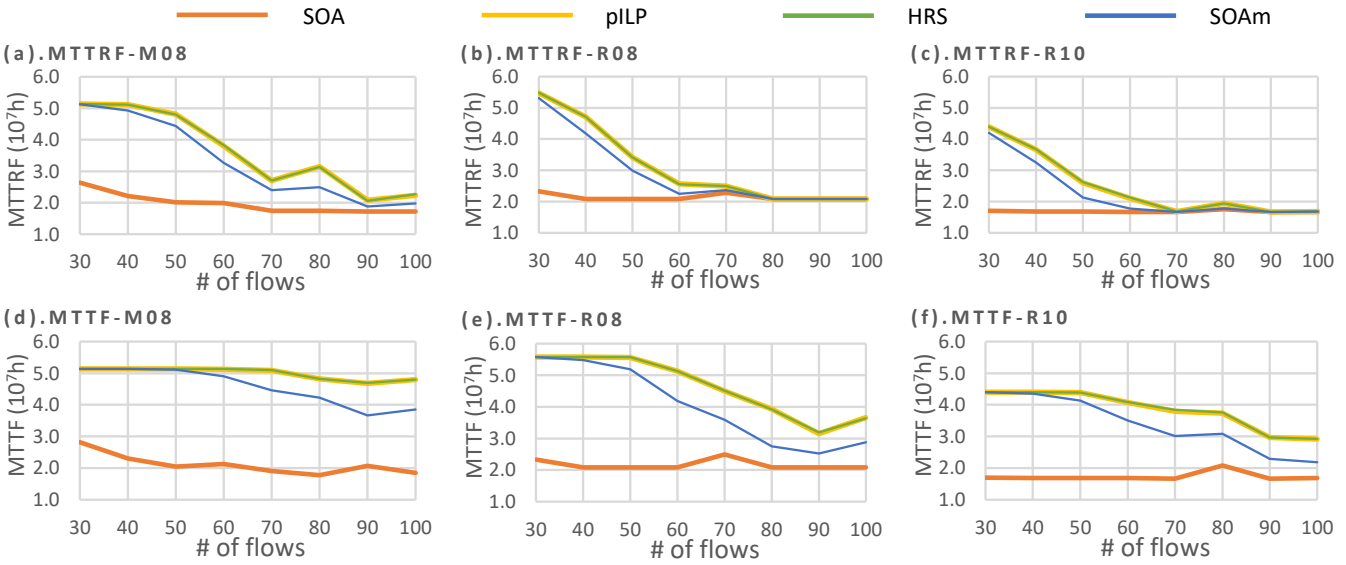


FIGURE 6. Reliability achieved by different recovery approaches on various topology and flow setups.

On each topology, evaluation is conducted by generating 30-100 flows which randomly have 1-3 destinations resulting $3 \times 5 \times 8 = 120$ (5 cases for each number of flows). Flows in F^{fm} and F^{rm} are identical except their period, i.e. F^{fm} consists of these flows with period set to two times the base period while F^{rm} consists of these flows with period set to 3 times the base period. We analyze the reliability of those systems under different recovery approaches compared above (SOA, pILP, HRS, SOAm) using the proposed reliability analysis. *The average of MTTRF and MTTF for every 5 test cases with the same number of flows* are shown in Fig. 6. Note that flows are initialized using pILP for all cases. And to reduce the state space, reliability analysis assumes that failure of more than 3 links will in result emergency halt.

As already indicated by previous experiments, loading more flows to the network will reduce the effect of recovery because they occupy time slots and links as well as increasing the number of disrupted flows on failures. Additionally, due to phase sharing, flows with lower periods are also harder to recover. Thus, reduced functionality mode tends to survive even after the failure of full functionality mode. For example, in M08 with 60 flows and pILP recovery, on average, MTTRF is $3.807 \times 10^7 h$ while MTTF is $5.128 \times 10^7 h$. This means that the systems are expected to survive an additional $1.321 \times 10^7 h$ with reduced functionality after leaving full functionality mode. Compared with SOA which is effective in fewer scenarios causing reduced functionality mode to fail soon after degrading from full functionality mode, we thereby conclude that the proposed recovery process significantly enhances the ability of reduced functionality to protect full functionality.

Recovery Approaches: pILP and HRS can have MTTRF and MTTF up to 2x larger than SOA which does not allow forwarding using the destinations, e.g., on M08 with 30 flows, average MTTRF for pILP is $5.137 \times 10^7 h$ while it is 2.637×10^7 for SOA. In contrast, SOAm which is the SOA with modified routing constraint to allow forwarding through destinations can achieve similar MTTRF and MTTF as pILP when the network is not heavily loaded. However, when the network load increases, it becomes less effective than pILP due to the lack of slot prioritizing.

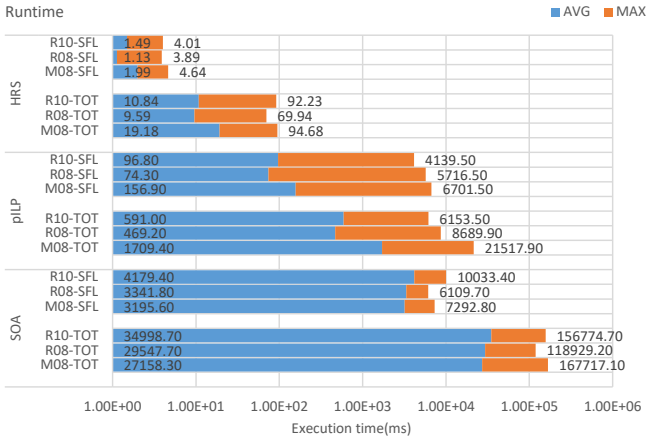
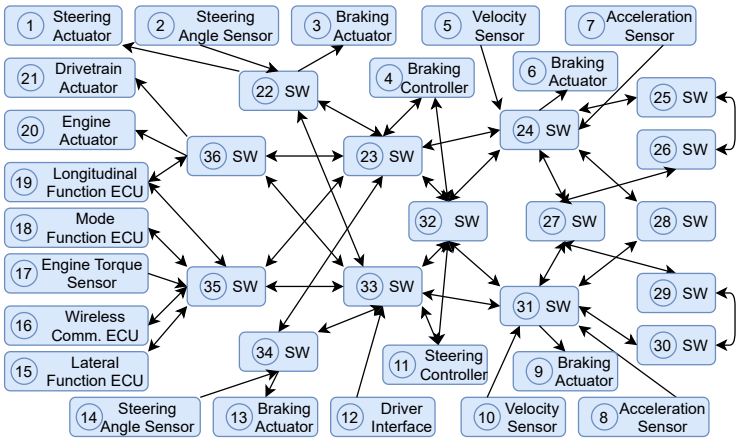


FIGURE 7. Average and maximum runtime measured for pILP and SOA on i7-9700F, and HRS on Cortex-A9. SFL: time to process a single flow, TOT: total time to process a failure.

The fact that HRS finds longer path for multicast flows does not cause major reliability impact in this realistic setup. The reason is that multicast flows are sensitive to the integrity of the topologies. So, they are more likely to cause system halt due to loss of connectivity which is not recoverable by any approach than saturating the network. In fact, although only 1/3 of the flows has 3 destinations, we found more than 50% of the degradation is caused by flows with 3 destinations no matter what recovery approach is used. Therefore, the reliability achieved by a recovery approach is more affected

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3092572, IEEE Access



Flow	From	To	Task
1	12	11	Driver steering
2	12	4	Driver braking
3	12	19	Driver accelerating
4	11	1	Steering actuating
5	4	3,6,9,13	Braking actuating
6	11	18	Steering interference
7	4	18	Braking interference
8	18	19	Longitudinal mode
9	18	15	Lateral mode
10	7	15	Accelerator sensor 1
11	8	15	Accelerator sensor 2
12	5	15	Velocity sensor 1
13	10	15	Velocity sensor 1
14	17	19	Engine torque sensor
15	19	20,21	Powertrain control
16	16	15	Road detection
17	19	11	Controller steering
18	19	4	Controller braking
19	19	16	V2V Longitudinal
20	15	16	V2V Lateral

FIGURE 8. Network and flows reconstructed from Orion [31] and EcoTwin [40]. SW represents switches.

Approach	MTTR/h	MTTF/h	MAX/ms	AVG/ms
pILP	9.75×10^6	9.76×10^6	2.49×10^4	432.75
SOA	9.75×10^6	9.76×10^6	5.87×10^4	5.32×10^3
HRS	9.75×10^6	9.76×10^6	29.69	0.524

TABLE 1. Experiment results for the case study. MAX and AVG stands for the maximum and average time observed to process a failure (TOT).

by its capability of handling unicast flows. In this experiment, since HRS is also enforced with slot prioritizing, it even achieves better MTTRF and MTTF than SOAm which has be proven to handle multicast flows more efficiently.

Topology: By virtue of the cross-links, meshes are more redundant and can safely carry more flows. Thus, reliability of R08 is more sensitive to the number of flows compared with M08. The capability to reliably carry flows also reduces when rings scale up. Because limited redundancy can be easily congested during recovery, plus the fact that the failure probability increases when there are more links.

C. RUN-TIME EXECUTION

While performing reliability analysis to the 120 test cases above, our program also records the execution time to process each failure and flows. They are shown in Fig. 7. Since the execution time of HRS and ILPs have different magnitude, we apply logarithmic scale on x-axis to ensure readability.

Even in the worst case, HRS can process a single flow within $5ms$. Thus, recovering a single link failure without degradation requires no more than $b \times 5ms = 80ms$. Compared with SOA, the per-flow average execution time is $156.90/1.99 \simeq 79\times$ faster on M08. The worst-case total recovery time is observed during degradation, where up to $2 \times b$ flows need to be processed. But this can be avoided by evaluating reduced functionality mode in parallel (on 2 cores) with full functionality mode. According to the experiment, HRS is capable of computing alternative flow implementations within $100ms$. Considering that today's vehicles are equipped with much more powerful processors than a 650MHz Cortex-A9, e.g., 2.0GHz Cortex-A72 on NXP BlueBox [41], we conclude that HRS is capable of dynamically computing flow configuration in response to failures well within required $100ms$ deadline.

Compared with SOA, the per-flow average execution time of pILP is improved by $3195/157 \simeq 20\times$ on M08 and more than $40\times$ on both rings. Thus, pILP has more advantages while being used to setup networks or benchmark topologies. However, both ILP-based approaches require more than $10s$ to process a single failure, thus are not suitable for run-time recovery.

D. AN AUTOMOTIVE CASE STUDY

To demonstrate the proposed approaches are suitable for realistic automotive networks, we perform a case study on the scenario shown in Fig. 8. The network consisting of 15 switches and 21 end stations is modified from the architecture of the Orion crew exploration vehicle [31]. Although this topology is not initially built for automotive applications, it features at larger scale and more redundancy than automotive networks. Thus, it demonstrates that our approaches scale to realistic automotive use cases. We keep the switch network of Orion and remap its end stations to vehicle components. Unmapped end stations are considered non-safety-critical and thus omitted. Since the applications of Orion only use up to 5 TT flows, we reconstruct an automotive application based on a level 2+ truck platooning architecture EcoTwin [40]. The safety-critical functions of the EcoTwin is initially distributed on four processors. To maximize the amount of flows, we map each function to an individual processor resulting six processors connected by TT flows. Audio and video streams in the EcoTwin are also omitted because they are forwarded

as AVB flows. Additionally, the original topology does not have redundant links for end stations. So, we add redundant links for end stations related to the basic control functions (e.g., braking and steering).

The number of time slots per link in our case study network is set to 8 as it is the minimum number of time slots for the network to reach maximum reliability. Link failure rate is $\lambda = 3 \times 10^{-9}h^{-1}$ as in the previous experiments. While the application performs automatic truck platooning in full functionality mode, its reduced functionality mode can be designed to provide only the basic driving interface for drivers. A driver take-over is requested and the platooning application is still executed to provide route suggestion instead of directly controlling the vehicle. Thus, flows in full and reduced functionality mode have different period, other specifications remain the same for simplicity. The flow period equals to base period in F^{fm} and two times the base period in F^{rm}. We compare pILP, SOA, and HRS.

The reliability of the system and execution time for each approach is shown in Table 1. As there are plenty of time slots allocated for recovery, all three approaches compared reaches the maximum MTTRF and MTTF. This is consistent with our previous experiment that when there are plenty of redundancy and relatively fewer flows, network reliability achieved by different recovery approaches is similar. Complex topology introduces significant variation of the execution time. Although the average execution time of pILP is 12x lower than SOA, its worst-case execution time is only 2x better. While both pILP and SOA cannot meet the $100ms$ deadline for run-time recovery, the maximum execution time of HRS is $29.69ms$ which is well below the recovery deadline.

VIII. CONCLUSION

This paper presents a design to realize run-time recovery of TSN. We develop an improved design-time ILP approach and a fast run-time heuristic algorithm based on a novel multi-mode recovery scheme, which we evaluate with a CTMC-based reliability analysis. Our evaluation shows that recovery improves MTTF of the system by up to $2\times$ depending on the redundancy, topology, and recovery algorithm. The execution time of the heuristic algorithm on an embedded processor is within the $100ms$ requirement of recovery. We compare run-time recovery with the standardized FRER, as well. The result shows that run-time recovery which requires less redundancy in bandwidth achieves better protection against multi-point failures. In the future, we will implement efficient network architecture that supports fast configuration and mode switching to bring run-time recovery to reality. We are also interested in developing mechanisms in which ECUs collaborate with the spatial and temporal redundancy of TSN to ensure a more reliable system.

REFERENCES

[1] *Time-Sensitive Networking (TSN) Task Group*, (accessed March 3, 2021). [Online]. Available: https://1.ieee802.org/tsn/#TSN_Standards

[2] J. T. et al, "Requirements for automotive AVB system profiles," *AVnu Alliance Whitepaper*, mar 2011.

[3] "IEEE/ISO/IEC international standard - local and metropolitan area networks - specific requirements - part 1cb: Frame replication and elimination for reliability," *ISO/IEC/IEEE 8802-1CB:2019(E)*, pp. 1–106, 2019.

[4] K. Kirkpatrick, "Software-defined networking," *Communications of the ACM*, vol. 56, no. 9, pp. 16–19, 2013.

[5] D. Staessens, S. Sharma, D. Colle, M. Pickavet, and P. Demeester, "Software defined networking: Meeting carrier grade requirements," in *2011 18th IEEE Workshop on Local Metropolitan Area Networks (LANMAN)*, 2011, pp. 1–6.

[6] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "Openflow: Meeting carrier-grade recovery requirements," *Computer Communications*, vol. 36, no. 6, pp. 656 – 665, 2013, reliable Network-based Services.

[7] J. Schlatow, M. Moostl, R. Ernst, M. Nolte, I. Jatzkowski, M. Maurer, C. Herber, and A. Herkersdorf, "Self-awareness in autonomous automotive systems," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, 2017, pp. 1050–1055.

[8] N. G. Nayak, F. Dürr, and K. Rothermel, "Time-Sensitive Software-Defined Network (TSSDN) for real-time applications," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, ser. RTNS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 193–202.

[9] D. Thiele and R. Ernst, "Formal analysis based evaluation of software defined networking for time-sensitive ethernet," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2016, pp. 31–36.

[10] N. G. Nayak, F. Dürr, and K. Rothermel, "Incremental flow scheduling and routing in time-sensitive software-defined networks," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 5, pp. 2066–2075, May 2018.

[11] A. Sgambelluri, A. Giorgetti, F. Cugini, F. Paolucci, and P. Castoldi, "Openflow-based segment protection in Ethernet networks," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 5, no. 9, pp. 1066–1075, 2013.

[12] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "Enabling fast failure recovery in openflow networks," in *2011 8th International Workshop on the Design of Reliable Communication Networks (DRCN)*, 2011, pp. 164–171.

[13] W. Steiner, "An evaluation of SMT-based schedule synthesis for time-triggered multi-hop networks," in *2010 31st IEEE Real-Time Systems Symposium*, Nov 2010, pp. 375–384.

[14] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, "Scheduling real-time communication in IEEE 802.1qbv time sensitive networks," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, ser. RTNS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 183–192.

[15] W. Steiner, S. S. Craciunas, and R. S. Oliver, "Traffic planning for time-sensitive communication," *IEEE Communications Standards Magazine*, vol. 2, no. 2, pp. 42–47, 2018.

[16] S. S. Craciunas, R. S. Oliver, and T. AG, "An overview of scheduling mechanisms for time-sensitive networks," *Proceedings of the Real-time summer school LÉcole dÉté Temps Réel (ETR)*, pp. 1551–3203, 2017.

[17] N. Reusch, L. Zhao, S. S. Craciunas, and P. Pop, "Window-based schedule synthesis for industrial ieee 802.1qbv tsn networks," in *2020 16th IEEE International Conference on Factory Communication Systems (WFCS)*, 2020, pp. 1–4.

[18] E. Schweissguth, P. Danielis, D. Timmermann, H. Parzyjegla, and G. Mühl, "ILP-based joint routing and scheduling for time-triggered networks," in *Proceedings of the 25th International Conference on Real-Time Networks and Systems*, ser. RTNS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 8–17.

[19] F. Smirnov, M. Glaß, F. Reimann, and J. Teich, "Optimizing message routing and scheduling in automotive mixed-criticality time-triggered networks," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2017, pp. 1–6.

[20] T. L. Mai, N. Navet, and J. Migge, "On the use of supervised machine learning for assessing schedulability: Application to Ethernet TSN," in *Proceedings of the 27th International Conference on Real-Time Networks and Systems*, ser. RTNS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 143–153.

[21] T. L. Mai, N. Navet, and J. Migge, "A hybrid machine learning and schedulability analysis method for the verification of TSN networks," in *2019 15th IEEE International Workshop on Factory Communication Systems (WFCS)*, 2019, pp. 1–8.

[22] A. Elia, L. Ferrarini, and C. Veber, "Analysis of Ethernet-based safe automation networks according to IEC 61508," in *2006 IEEE Conference on Emerging Technologies and Factory Automation*, 2006, pp. 333–340.
[23] H. C. Cankaya and V. S. S. Nair, "Accelerated reliability analysis for self-healing sonet networks," *SIGCOMM Comput. Commun. Rev.*, vol. 28, no. 4, p. 268–277, Oct. 1998.
[24] T. Angskun, G. Bosilca, G. Fagg, J. Pješivac-Grbovi, and J. J. Dongarra, "Reliability analysis of self-healing network using discrete-event simulation," in *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid '07)*, 2007, pp. 437–444.
[25] M. Glaß, M. Lukasiewycz, F. Reimann, C. Haubelt, and J. Teich, "Symbolic reliability analysis of self-healing networked embedded systems," in *Computer Safety, Reliability, and Security*, M. D. Harrison and M.-A. Sujan, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 139–152.
[26] "ISO/IEC/IEEE international standard - local and metropolitan area networks - part 1q: Bridges and bridged networks amendment 3: Enhancements for scheduled traffic," *ISO/IEC/IEEE 8802-1Q:2016/Amd.3:2017(E)*, pp. 1–62, 2018.
[27] A. Finzi and S. S. Craciunas, "Breaking vs. solving: Analysis and routing of real-time networks with cyclic dependencies using network calculus," in *Proceedings of the 27th International Conference on Real-Time Networks and Systems*, ser. RTNS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 101–111.
[28] F. Dürr and N. G. Nayak, "No-wait packet scheduling for IEEE time-sensitive networks (TSN)," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, ser. RTNS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 203–212.
[29] S. Thangamuthu, N. Concer, P. J. L. Cuijpers, and J. J. Lukkien, "Analysis of ethernet-switch traffic shapers for in-vehicle networking applications," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2015, pp. 55–60.
[30] "IEEE standard for local and metropolitan area networks - timing and synchronization for time-sensitive applications in bridged local area networks," *IEEE Std 802.1AS-2011*, pp. 1–292, 2011.
[31] S. M. Laursen, P. Pop, and W. Steiner, "Routing optimization of AVB streams in TSN networks," *SIGBED Rev.*, vol. 13, no. 4, p. 43–48, Nov. 2016.
[32] T. Ishigooka, S. Otsuka, K. Serizawa, R. Tsuchiya, and F. Narisawa, "Graceful degradation design process for autonomous driving system," in *Computer Safety, Reliability, and Security*, A. Romanovsky, E. Troubitsyna, and F. Bitsch, Eds. Cham: Springer International Publishing, 2019, pp. 19–34.
[33] B. Stephens, A. L. Cox, and S. Rixner, "Scalable multi-failure fast failover via forwarding table compression," in *Proceedings of the Symposium on SDN Research*, ser. SOSR '16. New York, NY, USA: Association for Computing Machinery, 2016.
[34] C. A. Oliveira and P. M. Pardalos, "A survey of combinatorial optimization problems in multicast routing," *Computers & Operations Research*, vol. 32, no. 8, pp. 1953 – 1981, 2005.
[35] R. M. Karp, *Reducibility among Combinatorial Problems*. Boston, MA: Springer US, 1972, pp. 85–103.
[36] M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *J. ACM*, vol. 34, no. 3, p. 596–615, Jul. 1987.
[37] T. Yuge and S. Yanagi, "Quantitative analysis of a fault tree with priority and gates," *Reliability Engineering & System Safety*, vol. 93, no. 11, pp. 1577 – 1583, 2008.
[38] "TUL PYNQ-Z2 board," 2020. [Online]. Available: https://www.tul.com.tw/ProductsPYNQ-Z2.html
[39] "Road vehicles-functional safety-product development at the hardware level," *ISO 26262-5:2011*, Nov 2011.
[40] T. Bijlsma and T. Hendriks, "A fail-operational truck platooning architecture," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, 2017, pp. 1819–1826.
[41] "NXP BlueBox: Autonomous Driving Development Platform," 2020. [Online]. Available: https://www.nxp.com/design/development-boards/automotive-development-platforms/nxp-bluebox-autonomous-driving-development-platform:BLBX

WEIJIANG KONG received the B.Sc. in Electronic and Computer Engineering from Shanghai Jiao Tong University, Shanghai, China (2018) and the M.Sc. in Embedded Systems from Royal Institute of Technology, Stockholm, Sweden (2019). He is currently pursuing Ph.D. at Eindhoven University of Technology, Eindhoven, Netherlands. His current research interests includes time sensitive networking and network reliability.

MAJID NABI (S08-M13) received the B.Sc. and M.Sc. degrees both in computer engineering from Isfahan University of Technology and Tehran University, Respectively. He received the Ph.D. degree in electrical and computer engineering from Eindhoven University of Technology, Eindhoven, the Netherlands in 2013. He is currently an assistant professor with the Department of Electrical Engineering at TU/e, and Isfahan University of Technology. His research interests include efficient and reliable networked embedded systems, low-power wireless sensor networks, and internet-of-things. He is a member of IEEE.

KEES GOOSSENS has a BSc in computer science from the University of Wales (1988), and a PhD from the University of Edinburgh (1993). In his thesis he investigated the formal verification of hardware, in particular by using semi-automated proof systems in conjunction with formal semantics of hardware description languages such as ELLA and VHDL. He continued this work at several other universities before joining Philips Research in the Netherlands in 1995. At Philips he worked on behavioural synthesis for high-throughput video processing, then on on-chip communication protocols and memory management. Until 2010, at Philips/NXP Semiconductors Research he led the team that defined the Aethereal network on chip for consumer electronics, where real-time performance and low cost are major constraints. He was also part-time full professor at the Delft university of technology from 2007 to 2010, and is currently full professor at the Eindhoven university of technology, where his research focusses on composable (virtualised), predictable (real-time), low-power embedded systems, supporting multiple models of computation. He was editorial board member for the ACM Transactions on Design Automation of Electronic Systems (TODAES), associate editor for the Springer Journal of Design Automation of Embedded Systems (DAEM), and guest editor for several special issues on networks on chip. He is author on 17 patents, and published four books, 200+ articles, with four paper awards. His 2003 paper was selected as one of the 30 most influential papers of 10 years of the DATE conference. His Google Scholar H index is 50. He is or was steering committee member of ACSD, NOCS, MPSOC, and TPC member for many conferences including CODES+ISSS, CRTS, DATE, DSD, ECRTS, FPL, ICPP, ReConFig, RTAS, SAMOS, and VLSI-SOC.