Fault-Tolerant Topology and Routing Synthesis for IEEE Time-Sensitive Networking

Gavrilut, Voica Maria; Zarrin, Bahram; Pop, Paul; Samii, Soheil

Published in:
Proceedings of the 25th International Conference on Real-Time Networks and Systems

Link to article, DOI:
10.1145/3139258.3139284

Publication date:
2017

Document Version
Peer reviewed version

Citation (APA):
Gavrilut, V. M., Zarrin, B., Pop, P., & Samii, S. (2017). Fault-Tolerant Topology and Routing Synthesis for IEEE Time-Sensitive Networking. In *Proceedings of the 25th International Conference on Real-Time Networks and Systems* Association for Computing Machinery. https://doi.org/10.1145/3139258.3139284

Fault-Tolerant Topology and Routing Synthesis for IEEE Time-Sensitive Networking

Voica Gavriluț, Bahram Zarrin and Paul Pop
DTU Compute, Technical University of Denmark
voga@dtu.dk

Soheil Samii
General Motors R&D, USA
Linköping University, Sweden

ABSTRACT

Time-Sensitive Networking (TSN) is a set of IEEE standards that extend Ethernet for safety-critical and real-time applications. TSN is envisioned to be widely used in several applications areas, from industrial automation to in-vehicle networking. A TSN network is composed of end systems interconnected by physical links and bridges (switches). The data in TSN is exchanged via streams. We address safety-critical real-time systems, and we consider that the streams use the Urgency-Based Scheduler (UBS) traffic-type, suitable for hard real-time traffic. We are interested in determining a fault-tolerant network topology, consisting of redundant physical links and bridges, the routing of each stream in the applications, such that the architecture cost is minimized, the applications are fault-tolerant (i.e., the critical streams have redundant disjoint routes), and the timing constraints of the applications are satisfied. We propose three approaches to solve this optimization problem: (1) a heuristic solution, (2) a Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristic, and (3) a Constraint Programming-based model. The approaches are evaluated on several test cases, including a test case from General Motors Company.

CCS CONCEPTS

• **Computer systems organization → Fault-tolerant network topologies**;

KEYWORDS

Safety-Critical Systems, TSN, Fault-Tolerant Architectures

1 INTRODUCTION

Many safety-critical real-time applications, following physical, modularity or safety constraints, are implemented using distributed architectures, composed of heterogeneous processing elements (PEs) embedded in "smart" devices which are interconnected in a network. A large number of communication protocols have been proposed for embedded systems. However, only a few protocols are suitable for safety-critical real-time applications [22]. In this paper, we are interested in the protocol colloquially known as Time-Sensitive Networking (TSN) [33]. TSN is used in several application areas, from industrial automation to automotive architectures. For example, in the automotive area, fault-tolerant TSN networks are envisioned in future autonomous driving architectures, since they have the bandwidth requirements to integrate traffic from multiple sensors and the dependability required for autonomous driving.

Ethernet [14], although it has low cost and high speed, is known to be unsuitable for real-time and safety-critical applications [6]. For example, in half-duplex implementations, frame collision is unavoidable, leading to unbounded transmission times. [6] presents the requirements for a real-time network and how Ethernet can be improved to comply with these requirements. Several real-time communication solutions based on Ethernet have been proposed. [23] and [5] describe and compare several of the proposed Ethernet-based real-time communication protocols.

TSN [33] is a set of sub-standards which extend the IEEE 802.1 standards (for switched Ethernet networks) for safety-critical and real-time applications. First, IEEE 802.1Q-2005[1] introduced support for prioritizing the Best-Effort (BE) traffic in order to improve Quality of Services (QoSs). Following this, the IEEE Audio-Video-Bridging (AVB) Task Group was formed to develop another set of enhancements, namely IEEE 802.1BA known as AVB. This standard introduces two new shaped AVB traffic-types, with bounded *Worst-Case end-to-end Delays* (WCDs). In 2012, the AVB Task Group was renamed to IEEE 802.1 Time-Sensitive Networking Task Group to reflect the shifted focus onto further extending the protocol towards safety-critical and time-sensitive transmissions, and has introduced new traffic types such as Time-Triggered (TT) [29] and Urgency-Based Scheduler (UBS) [26].

In this paper, we are interested in safety-critical real-time applications. We consider that the application messages use the Urgency-Based Scheduler (UBS) traffic-type IEEE 802.1Qcr [32]. UBS is an asynchronous traffic scheduling algorithm, which gives low delay guarantees while maintaining a low implementation complexity. It also provides a temporally-composable timing analysis, see Sect. 4 and [26] for more details. Compared to the TT traffic type, UBS does not require schedule tables, which can be difficult to create, and compared to AVB, UBS guarantees lower latencies and has a simpler and faster timing analysis. However, although we consider UBS in this paper, our approach can handle any combination of traffic types, as long as a timing analysis is available. The choice of traffic type depends on the characteristics of the applications, and the problem of determining the appropriate traffic types has been addressed in [8] for mixed-criticality traffic in TTEthernet.

TSN is highly suitable for applications of different safety criticality levels, as it offers spatial separation for mixed-criticality traffic through the concept of Virtual Local Area Network (VLAN), as well as temporal separation through the various traffic type mechanisms. A TSN network is composed of End Systems (ESes) interconnected by physical links and Network Switches, also known in TSN as

[1] We will not provide references for all standards, but these can be easily found based on their names.

RTNS '17, October 4–6, 2017, Grenoble, France

ACM ISBN 978-1-4503-5286-4/17/10…$15.00
https://doi.org/10.1145/3139258.3139284

Bridges (Bs). The links are full duplex, and the network can be multi-hop, see Sect. 2 for the architecture model. The data in TSN is exchanged via *streams*, see Sect. 3 for the application model. Because we target safety-critical applications, we consider that the routing of streams is determined statically at design time. However, non-critical streams can use dynamic route and bandwidth reservation mechanisms provided by TSN, see [13] and [30].

We are targeting safety-related systems, which have to be developed according to certification standards; for example, IEC 61508 is used in industrial applications, ISO 26262 is for the automotive area, whereas DO 178C refers to software for airborne systems. Considering the current certification practice, we assume that the engineer will specify for each application, depending on its criticality, the required *Redundancy Level* (RL). At the level of the network topology, this translates into requirements for redundant disjoint routes between the ESes involved in the communication. Thus, if a physical link or a bridge will fail, the other routes can still deliver the information by the deadlines. The current approach in such a situation is to use hardware redundancy at the network level and replicate the complete network, as discussed by [2] for an avionics network. Such a solution may not be scalable in terms of weight, space, and resource efficiency for application areas where functions have varying redundancy requirements.

In this paper, our focus is on determining a low-cost fault-tolerant network architecture, which can guarantee the safety and real-time requirements of the applications. We assume that the applications and ESes are given and that the designer has established the redundancy levels, depending on the criticality of the applications. We are interested in determining a fault-tolerant network topology, consisting of redundant physical links and bridges, the routing of each stream in the applications, such that the architecture cost is minimized, the applications are fault-tolerant (i.e., the critical streams have RL redundant disjoint routes), and the timing constraints of the applications are satisfied.

Contributions: This is the first time, to our knowledge, that the problems of (i) topology synthesis and (ii) routing of time-sensitive traffic have been addressed for TSN. We propose three strategies to solve these problems: (1) a fast heuristic solution, (2) a Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristic that finds good quality solutions in a reasonable time, and (3) a Constraint Programming-based model that searches for the optimal solution.

The paper is organized as follows. The next section presents the related work. Sect. 2 and Sect. 3 present the topology architecture and traffic models used in the paper. The concepts related to TSN relevant for our paper are presented in Sect. 4. The problem formulation is presented in Sect. 5 and illustrated with a motivational example in Sect. 5.1. The proposed optimization strategies are presented in Sect. 6, and Sect. 7 presents our experimental evaluation.

1.1 Related Work

Researchers have started to address the analysis and optimization of "Deterministic Ethernet" (DE) protocols, such as TTEthernet, Industrial Ethernet and TSN. The problem of determining the network topology, i.e., the number of bridges and their interconnection via physical links and to the end systems, is called *network planning and design*. This problem has been addressed for DE in the context of Industrial Ethernet [16] and TTEthernet in aerospace [27].

In the telecommunications area, there is a lot of work on network reliability and redundancy optimization. An annotated overview of system reliability optimization, which covers also network reliability is presented in [17]. In [15], the authors present the latest research results in network reliability optimization. Several network reliability measures have been proposed in the literature, such as connectivity, resilience and performability. Researchers have proposed several approaches to the optimization problem, including heuristics, metaheuristics and exact solutions based, for example, on mathematical programming [15].

However, these results cannot be applied directly to DE. One of the basic assumptions of earlier works on network reliability optimization is that once a fault is detected, the network will reconfigure itself to avoid the fault. That is, new routes will be found for messages. In the case of DE the routes for safety-critical applications are typically *static*: they are loaded into the end systems and network switches at design time, and it is not possible to change the routing dynamically, at runtime. In this context, researchers have prosed a fault-tolerant topology selection for TTEthernet [9]. However, for non-critical applications, runtime reconfiguration, including routing, is a relevant problem.

Routing optimization is a well-studied subject where Wang et al. [35] and Grammatikakis et al. [11] provide excellent overviews of the different centralized and distributed routing algorithms. Researchers have also addressed routing in safety-critical systems [12], [20]. For ARINC 664p7, Al Sheikh et al. [1] proposed an approach to find the optimal routes in ARINC 664p7 networks using Mixed Integer Linear Programming. Tămaş-Selicean et al. [28] have used a Tabu Search-based metaheuristic to, among other things, optimize the routing of the RC traffic type to minimize the WCDs in TTEthernet systems.

Regarding routing in TSN, AVB flows are typically established at runtime using the *Stream Reservation Protocol (SRP)* [30] where either the *Rapid Spanning Tree Protocol* (RSTP) or *Shortest Path Bridging* (SPB) are used to determine the routing. The future enhancements around TSN will support more sophisticated runtime routing algorithms, and the possibility to also determine the routes offline. Researchers have proposed an offline routing optimization approach for AVB in [18]. However, routing for time-sensitive traffic types such as TT and UBS has not been addressed previously.

2 ARCHITECTURE MODEL

We model the architecture, which is a TSN network as an undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where the vertices (or nodes) $\mathcal{V} = \mathcal{ES} \cup \mathcal{B}$ denote the set of all End Systems (ESes) and network switches, usually denoted in TSN as Bridges (Bs), respectively. The edges $\mathcal{E}$ are the full-duplex physical links interconnecting the ESes and Bs. An ES can be of several types. For example, in automotive architecture, an ES is typically an Electronic Control Unit (ECU) composed of a CPU, memory, and I/Os. However, an ES could also be an intelligent sensor such as video camera, radar, or LiDAR. All ESes regardless of their type have a Media-Independent Interface (MII) connector which is a full-duplex digital interface to connect the ES to the network. Fig. 1 shows an example network with 4 ESes and 4 Bs.

In this paper, for the given set of ESes, we determine the set of bridges, $\mathcal{B}$ to be used and the physical interconnections. We assume that the system engineer provides a network component library $\mathcal{L}$

including a set of bridge types $\mathcal{BT}$ and a set of physical link types $\mathcal{LT}$. Such a library will be defined based on the TSN bridges and physical links available on the market and suitable for the application area considered. For example, TTTech Computertechnik AG and Infineon Technologies AG provide TSN bridges, for the automotive area, with different number of ports and different physical layer technologies, such as the IEEE 802.3 standards for automotive 100 Mbit/s and 1 Gbit/s Ethernet. Our approach is general and can be applied in several areas from automotive to industrial automation.

The library is defined as $\mathcal{L} = (\mathcal{BT}, \mathcal{LT}, BC)$, where $\mathcal{BT}$ is a set of bridge types, and $\mathcal{LT}$ is the set of physical link types. In general, an ES can be connected to any bridges. However, in practice, there can be constraints that limit the type of bridges and physical links that can be used by an ES. For example, a video camera could impose a limit on the fit of the bridge (due to the fact that the ES and bridge are packaged into the same electrical component), such that it fits together with the camera in the desired location in the vehicle. Therefore, there are packaging constraints for some ESes that limit the network topology synthesis. We capture the bridge constraints with the function BC, which is a mapping from an End System ES_i to the limited set of the bridge and physical link types that can be used by ES_i. In Fig. 1 all bridges have assigned the bridge type bt_2, which has 1 internal and 3 external ports.

Id	Cost	No.int. ports	No.ext. ports
bt_1	2	1	2
bt_2	8	1	3
bt_3	10	2	2

Id	Cost	Speed Mbit/s	Int./Ext.
lt_1	7	100	Ext.
lt_2	1	1000	Int.

Table 1: Example library $\mathcal{L}$

We use two functions to specify the type of the bridges and the physical links used within the architecture network. The first function $BT : \mathcal{B} \twoheadrightarrow \mathcal{BT}$, specifies the type of a bridge, e.g., $BT(B_1) = bt_2$ in Fig. 2. The other function $LT : \mathcal{E} \twoheadrightarrow \mathcal{LT}$, specifies the type for each link in the network topology, e.g., $LT(l_1) = lt_1$. We represent the monetary cost of a bridge, ES, and physical link as *cost*, e.g., $B_1.cost = 8$. We denote the transmission rate of a physical link as *speed*, e.g., $l_1.speed = 100$ Mbit/s, the connectivity type of a physical link as *lct*, e.g., $l_1.lct =$ Ext, and the number of int. and ext. ports of a bridge as *noIntPorts* and *noExtPorts*, e.g., $B_1.noIntPorts = 1$.

Similar to the network engineering practice, we will allow the "chaining" of several bridges to construct a new type of bridge, that has more ports, hence supporting more connections. Fig. 2 shows an automotive ECU with a microcontroller ES_1 connected to a bridge that is built from chaining two bridges B_1 and B_2 of types bt_3 and bt_1, respectively.

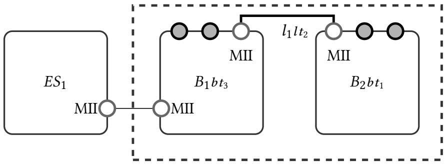


Figure 2: Chaining bridges

We distinguish between two types of connections: internal links, which are between the MIIs of ESes and bridges, and external links, which connects two bridges using a physical connector, colloquially known as PHY. A PHY consists of a physical digital to analog converter, as well as filters to support the bit rate with the required signal qualities within the operating environment, and a connector to the wiring, for example. Both internal and external links are physical links denoted as l_i, $l_i \in \mathcal{E}$, which are bidirectional. We call a link connectivity ESes to bridges an internal link because there are application areas, e.g., automotive, where the bridge is integrated with the ES on the same board, so the internal link is a connection on the PCB between the pins of the microcontroller/sensor and the pins of the TSN bridge. Such an internal link is more reliable compared to an external link, which is susceptible to PHY connector failures, see Sect. 3.1 for the fault model.

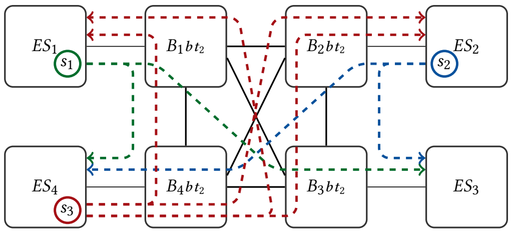


Figure 1: Architecture model example

A dataflow link (DL) dl_j represents a directed connection on a physical link l_i, $(ES_1 - B_1)$ from Fig. 1 for example. A dataflow path (DP) dp_k is a sequence of interconnected DLs. Such a path in Fig. 1 is $[(ES_1 - B_1), (B_1 - B_4), (B_4 - ES_4)]$. The set of all DLs is denoted with $\mathcal{DL}$ and the set of all DPs is $\mathcal{DP}$.

3 APPLICATION MODEL

The safety-critical real-time applications are modeled as periodic tasks distributed on the ESes. Our application model captures the communication among tasks via streams. A stream is denoted as s_i and the set of all streams is denoted $\mathcal{S}$. Streams may be multicast, so each stream s_i has a source $s_i.src$, which is an ES, and has one or multiple destinations ESes $s_i.dests$. The messages transmitted in a stream may be split into several packets, and each packet is wrapped in an Ethernet *frame*. The messages of a stream may have to be fragmented into several packets, if their length is larger than the Ethernet Maximum Transmission Unit (MTU) of 1,500 bytes. The problem of message fragmenting and frame packing is orthogonal to our problem, and has been addressed in the context of Deterministic Ethernet [28].

We use the leaky bucket traffic model in this paper, see [26] for more details, which means that each stream s_i is characterized by a burstiness $s_i.B$, which represents the maximum amount of data that can be transmitted at once, and a leak rate $s_i.R$. Our application model can accept any type of streams which satisfy the leaky bucket constraint, i.e., for a stream s_i the total amount of data w_i accumulated on a duration d is bounded by $w_i(d) \leq s_i.B + d \cdot s_i.R$. Each frame of a stream $s_i \in \mathcal{S}$ has a deadline $s_i.D$ by which the frame has to arrive at its destinations, relative to the releasing of each frame. The advantage of such a traffic model is its versatility: it can model strictly periodic streams with fixed size frames, sporadic streams, as well as variable sized frames useful for multimedia data and large data payloads that need to be transmitted in back-to-back frames, see [26] for more details. For example, a strictly periodic stream s_i, with a packet size $s_i.size$ a period $s_i.T$ and an absolute deadline $s_i.deadline$ by which the message need to be delivered, can be modeled with the leaky bucket model as:

Id	Src.	Dests.	B in B	R in ms	D in ms	rl
s_1	ES_1	ES_3, ES_4	150	15	7	1
s_2	ES_2	ES_3, ES_4	100	10	4.5	1
s_3	ES_4	ES_1, ES_2	100	10	4	2

Table 2: Application model example

$s_i.R = s_i.size/s_i.T$ and $s_i.B = s_i.size$ and $s_i.D = s_i.deadline$. An aperiodic stream with a maximum allowed amount of data $s_i.size$ exceeding MTU and a minimum inter-arrival time, which is denoted $s_i.T$, can be similarly modeled with all frames on which the streams is fragmented inheriting the stream's relative deadline $s_i.D$.

We model the routing of a stream as a Multicast Tree $mt_s(s_i)$, a direct structure with the source as root and destinations as leaves. $\mathcal{MT}_s$ is the set of all multicast trees. Fig. 1 shows 4 trees. For example, $mt_s(s_1)$ for the stream s_1 from ES_1 to ES_3 and ES_4, has the route $ES_1 - B_1 - [[B_3 - ES_3], [B_4 - ES_4]]$ depicted with a thick green dashed arrow. For each original stream $s_i \in \mathcal{S}$ we denote with $s_i^j, 1 \leq j \leq s_i.rl$ its j^{th} redundant copy, s_i^1 being s_i itself, and $s_i.rl$ is the stream's redundancy level, see Sect. 3.1. The set $\mathcal{S}^\star$ denotes the set of all streams and their redundant copies. Table 2 shows an example application model, with s_3 having a redundancy level of 2 and the other two streams not being fault-tolerant. The routes for the streams listed in Table 2 are depicted in Fig. 1.

Due to the delays implied by path recovery in case of a physical fault, in this paper we proposed a network configuration where the routes are static: they are determined and loaded into the end systems and bridges at design time. For the non-critical streams the routes can be determined also dynamically, e.g., using the TSN sub-standards as 802.1Qat or 802.1Qcc.

As discussed in the introduction, we assume that each stream uses the UBS traffic-type. UBS allows the definition of a non-unique stream priority $s_i.priority$, which can change at each hop. The assignment of priority is an interesting optimization problem. However, in this paper we assume that the priorities are given as input by the system engineer, and without loss of generality we assume that the priority is fixed for a stream. For example, the priority for a stream $s_i \in \mathcal{S}$ could be defined by the ratio $s_i.B/s_i.R/s_i.deadline$, thus the stream with higher burstiness, lower leak rate and lower deadline has a higher priority. In our example from Table 2 we consider that all streams have the same priority.

3.1 Fault Model

Critical streams have to deliver their data even in the case of permanent failures. As mentioned, we assume that the system engineer provides for each stream s_i the required redundancy level (RL) $s_i.rl$, which means that the stream s_i has to be routed on $s_i.rl$ disjoint routes, such that the failure of any $RL - 1$ routes does not result in communication failure of the stream.

Our model is complementary to common probabilistic models of diagnostic and reliability requirements, such as MTTF targets established for various safety integrity levels in the automotive functional safety standard ISO 26262. New application areas, such as fail-operational autonomous driving systems [19], have functional safety requirements that are not currently addressed by ISO 26262. For example, some systems may require that there is no single point failure; absence of dual point failures are also seen in some highly critical applications. This is evident in the failure models considered in recent work in industry standardization (redundant communication paths in Ethernet [31]) and research work on dependable real-time Ethernet [4] to synthesize robust time-triggered schedules for a given number of maximum link failures. Similar requirements can be seen in some avionics and industrial control applications.

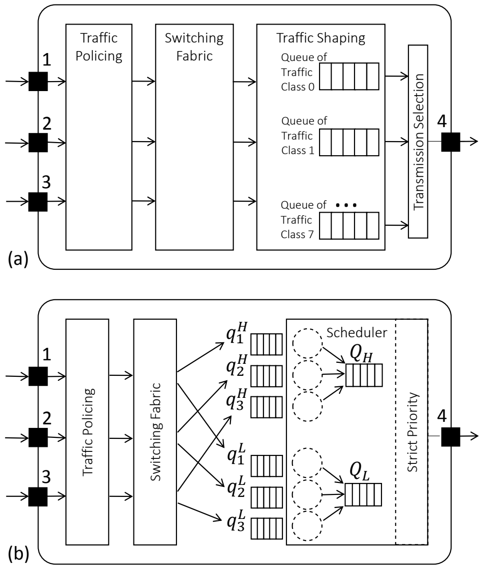


Figure 3: (a) Structure of a TSN-aware bridge and (b) UBS shaping for (a)

The most common types of permanent hardware failures is the physical connector (PHY) failures [10, 25], i.e., the cable terminals are corroded due to vibration and thermal fluctuations. End Systems (microcontrollers, smart sensors) and Bridges (network switches) are less likely to fail [25]. The internal links (MII) are on the PCB (microcontroller and switch are all on the same board), hence an internal link failure would result in an ES failure, from a system perspective.

4 TSN PROTOCOL AND UBS

In this paper, we consider that the streams are scheduled using the UBS traffic type. UBS has been proposed in [26] and, due to its advantages, it is currently being standardized by the TSN Task Group as IEEE P802.1Qcr [32]. UBS is a Rate-Constrained (RC) class, which means it does not rely on the availability of network clock synchronization (required for the TT traffic-type) or on offline synthesis and coordination of schedule tables. Moreover, due to the leaky bucket traffic model used in UBS (see Sect. 3), it does not impose any constraints on the burstiness or leak rate of streams.

The TSN sub-standards are amendments to IEEE 802.1Q, which is the standard for Bridged Virtual Local Area Networks using full-duplex IEEE 802.3 Ethernet. 802.1Q introduced additional content in the Ethernet frame header, including a 3-bit Priority Code Point (PCP) identifying up to 8 priority levels.

In Fig. 3a, we show the general structure of a 4-port TSN-aware bridge with the following main functionality: traffic policing, switching, traffic shaping and transmission selection. For presentation purposes, without loss of generality, we show only the ingress portion for the left 3 ports and only the egress portion of the right hand port. On ingress, frames go through a policing engine, which can be used to limit the allowed traffic and its bandwidth (TSN standard P802.1Qci). The switching is aware of each stream's route and forwards incoming frames to one (unicast) or more (multicast) egress ports based on the Address Resolution Logic (ARL) table, which is one of the decision variables of our routing synthesis problem. Each egress port contains a number of queues, each of which is configured to support a traffic type—for example, TT, Credit-Based Shaping (CBS) as in AVB, UBS, or FIFO-like for best-effort traffic. Each queue is configured at a fixed priority (with 8 priority levels available). The transmission selection algorithm selects frames for transmission based on the priority levels and based on whether or not a queue has a frame available for transmission. The traffic shaping, queuing, and transmission selection mechanisms are also implemented by each TSN-aware end system.

Incoming streams to the bridges shown in Fig. 3a and Fig. 3b are forwarded to the appropriate egress queue based on the stream identifier (typically the destination MAC address and optionally also VLAN identifier) and the frame priority value (i.e., the PCP). For UBS, each queue is shaped to satisfy the cumulative leaky bucket constraint of the streams mapped to that queue. The transmission selection algorithm then prioritizes the traffic based on queue priorities (discussed in Sect. 3 and in detail in [26]).

Fig. 3b shows a detailed view of UBS shaping at the egress port, for the same bridge shown in Fig. 3a. Incoming UBS streams are statically mapped to the UBS queues q_1^H, q_2^H, q_3^H, q_1^L, q_2^L, and q_3^L, which could be the first six queues in Fig. 3a (the last two could, for example, be dedicated to noncritical, best-effort traffic), with the rule that frames on different ingress ports are mapped to different queues (for fault isolation purposes). Frames in each UBS queue are shaped to satisfy the leaky-bucket constraint; the shaper is shown with a dashed circle. The purpose of the shaper is to establish whether or not the frame at the head of the queue is eligible for transmission, based on leaky-bucket constraints. Each queue has a fixed priority and it is possible that two or more queues have the same priority, for flow aggregation purposes. We assume that the order of priority levels is preserved through each hop along the route; see Sect. 3 for a more detailed explanation and [26] for the structure of a general purpose bridge. In our example, the bridge is aware of two priority levels, high (H) and low (L). Queues q_1^H, q_2^H, and q_3^H have the same priority (H) and are, after shaping, therefore merged into the logical FIFO queue Q_H (called *pseudo queue* in [26]). Similarly, queues q_1^L, q_2^L, and q_3^L have priority L and are merged into the logical FIFO queue Q_L. Note that Q_H, Q_L, and the strict priority scheduler in Fig. 3b correspond to the Transmission Selection block of Fig. 3a. In case the queue priorities are unique, there is no merging into logical queues after traffic shaping.

In the worst-case, in the scheduler of the sending node, a frame of stream s_i is delayed by all streams of higher priorities H, all streams of the same priority $C(i)$ and by the frame of maximum size of a lower priority stream $size_L$. On the receiver side, the frame is delayed, in the worst-case, by the slowest stream (i.e., the stream with highest burstiness). We use the analysis method from [26] to check the schedulability of each frame of a stream $s_i \in \mathcal{S}^\star$.

5 PROBLEM FORMULATION

The problem we are addressing in this paper can be formulated as follows. As an input we have (1) the set of end systems $\mathcal{ES}$, (2) the library of components $\mathcal{L}$, (3) the set of streams $\mathcal{S}$ for which we know the source, destination(s), and timing properties as well as the desired redundancy level $s_i.rl$. We are interested in determining an optimized solution $Sol = (\mathcal{G}, SR)$, where $\mathcal{G}$ is the network architecture and $SR : \mathcal{S}^\star \longmapsto \mathcal{MT}_s$ is a function that specifies the routing expressed as multicast trees for all the streams and their redundant copies, such that the architecture cost is minimized, the applications are fault-tolerant, considering the specified redundancy levels, and the timing constraints of all streams are satisfied.

5.1 Motivational Example

Let us consider the example from Fig. 4 where we have 4 ESs , ES_1 to ES_4 and the applications from Table 2. As library components we have 3 bridge types, bt_1 to bt_3, with types properties presented in Table 1 and where all ESes can be directly connected to all types of bridges. For these examples the physical links have a speed of 100 KBps and a cost of one monetary unit. In Fig. 4 the gray lines represent the internal links, the thicker black lines the external ones and the colored directed arrows are used for showing the stream routes. We present for each stream its WCD calculated by the analysis in Sect. 4. The streams in this example are schedulable if the WCDs are smaller or equal to the relative deadline D from Table 2.

The topology that maximizes redundancy without concern for cost is shown in Fig. 4a. To obtain this topology, we have connected each ES to its own bridge, and we have introduced full connectivity among the bridges: each bridge is connected to all other bridges. The bridge type is selected from the library such that it accommodates the required ports. The cost of such topology in Fig. 4a is 42 monetary units. As expected, we can find disjoint redundant routes for s_3, which is fault-tolerant, and all streams are schedulable.

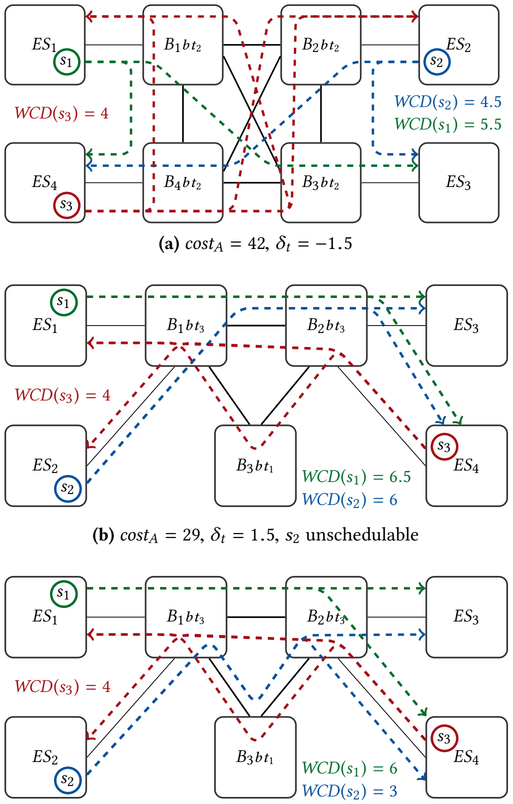
We can reduce the cost to 29 monetary units if we use the topology from Fig. 4b, which uses 3 bridges (although their individual cost is higher) and fewer physical links. We are able to find disjoint redundant routes for s_3. To determine the routes, in Fig. 4b we use the shortest path approach. However with this routing, s_2 is not schedulable. Because we are routing both s_1 and s_2 through link $(B_1 - B_2)$, in the worst case s_2 is delayed by frames of s_1 such that the $s_2.D$ is not satisfied.

Our approach optimizes both the physical topology and the routing of streams. Fig. 4c shows the same topology from Fig. 4b, but where the routes are optimized; counterintuitively, they may now take longer route compared to Fig. 4b. Here we can see that by routing s_2 through a longer route, namely $ES_2 - B_1 - B_3 - B_2 - [ES_3, ES_4]$ the routes for redundant copies are fully disjoint and all streams are now schedulable.

As we can see from this example, by only optimizing the topology and routing we are able to minimize the cost and guarantee the fault-tolerance and timing constraints of streams.

6 SYNTHESIS STRATEGIES

The optimization problem described in the previous section is NP-hard. According to [35], any routing problem subject to at least two additive or multiplicative tree constraints is an NP-hard problem.



(a) $cost_A = 42$, $\delta_t = -1.5$

(b) $cost_A = 29$, $\delta_t = 1.5$, s_2 unschedulable

(c) $cost_A = 29$, $\delta_t = -2.5$, **all schedulable**

Figure 4: Motivational example

Following the classification from [35], our problem can be expressed as a graph optimization problem subject to: (1) link constraint: the capacity of the links should not be exceeded, (2) the number of links adjacent to a vertex should not exceed the node number of ports, (3) the routes of redundant streams are link-disjoint[2] interrelated tree constraint and (4) all streams should be schedulable. Consequently, based on constraints (3) and (4), our optimization problem is NP-hard. To solve this problem, we propose three strategies, (1) a heuristic-based approach, further called Topology and Routing Heuristic (TRH), see Sect. 6.2, (2) a Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristic, see Sect. 6.3, and (3) a Constraint Programming-based strategy, further referred as Topology and Routing Optimization (TRO), see Sect. 6.4. In order to evaluate the visited solutions, all strategies use the cost function defined in Sect. 6.1.

6.1 Cost Function

A solution is evaluated using the following cost function:

$$cost_T(Sol(\mathcal{G}, SR)) = \wp_{sched} * \delta_t(SR) + \wp_{topo} * cost_A(\mathcal{G}) \qquad (1)$$

Where the first term is used to check if the solution is schedulable, the second term captures the architecture cost, and $\wp_{sched}$ and $\wp_{topo}$ are constant weights. In order to be able to aggregate the two terms, we normalize the two values. For both, δ_t and $cost_A$, the minimum and maximum values are computed and the actual values scaled in the range $(0, 1]$. To increase the probability of finding a solution we relaxed the schedulability constraint adding it as a soft constraint, i.e., as a highly penalized part of the cost function. In order to distinguish among the topologies of similar costs we are going for those solutions which: (a) are schedulable and (b) once they are schedulable, they should reduce the WCDs, see Fig. 4c. Therefore, the weighted penalty for the first term $\wp_{sched}$ is significantly higher than the architecture penalty $\wp_{topo}$.

The monetary cost of the network architecture is the sum over the cost of all bridges and all physical links in the topology $\mathcal{G}$:

$$cost_A(\mathcal{G}(\mathcal{V}, \mathcal{E})) = \sum_{v \in \mathcal{V}} v.cost + \sum_{e \in \mathcal{E}} e.cost \qquad (2)$$

The degree of schedulability δ_t represents the amount of tardiness with which all streams are arriving after their relative deadline, having a negative tardiness for a schedulable stream. We define $\delta_t(SR)$ as follows:

$$\delta_t(SR) = \begin{cases} \text{if at least one stream is not schedulable} \\ \displaystyle\sum_{s_i \in \mathcal{S}^\star,\, WCD(s_i) > s_i.D} WCD(s_i) - s_i.D \\ \displaystyle\sum_{s_i \in \mathcal{S}^\star} WCD(s_i) - s_i.D \qquad \text{otherwise} \end{cases} \qquad (3)$$

Where $WCD(s_i)$ is the WCD of a frame transmitted by a stream s_i having the UBS traffic class. We can then check if the frame is received by the deadline $s_i.D$. Such an analysis has been proposed in [26].

6.2 Heuristic Strategy

Algorithm 1: TRH

```
Input: End systems $\mathcal{ES}$, components library $\mathcal{L}$ and streams $\mathcal{S}$
Output: The TSN network $\mathcal{G}(\mathcal{V}, \mathcal{E})$ and the routes $SR$
1  $\mathcal{G}_{init} \leftarrow CreateInitialTopology$
2  $\mathcal{E}_{used} \leftarrow \emptyset$ ; $\mathcal{MT}_s \leftarrow \emptyset$
3  for $s_i : \mathcal{S}$ do
4      $\mathcal{G}_{ft} \leftarrow ConvertGraph(\mathcal{G}_{init})$
5      for $j \leftarrow 1$ to $s_i.rl$ do
6          $SR(s_i^j) \leftarrow SearchRoute(\mathcal{G}_{ft}, \mathcal{E}_{used}, s_i^j, s_i.rl - j)$
7          $\mathcal{G}_{ft} \leftarrow RemoveEdges(\mathcal{G}_{ft}, Edges(SR(s_i^j)))$
8          $\mathcal{E}_{used} \leftarrow \mathcal{E}_{used} \cup Edges(SR(s_i^j))$
9          $\mathcal{G}_{init} \leftarrow AssignBridgeTypes(\mathcal{G}_{init}, SR(s_i^j))$
10     end
11 end
12 $\mathcal{G} \leftarrow RemoveUnusedEdgesAndVertices(\mathcal{G}_{init}, \mathcal{E}_{used})$
13 $CheckSchedulability(\mathcal{S}, \mathcal{MT}_s)$
14 return $(\mathcal{G}, cost_A(\mathcal{G}), \mathcal{MT}_s)$
```

The Topology and Routing Heuristic (TRH) is a strategy which takes as input the set of end systems $\mathcal{ES}$, the components library $\mathcal{L} = (\mathcal{BT}, \mathcal{LT}, BC)$ and the set of streams $\mathcal{S}$, and returns the network topology $\mathcal{G}$ and the routing SR, see Alg. 1.

TRH starts from a fully connected initial solution $\mathcal{G}_{init}$, line 1 in Alg. 1 (similar to the solution discussed in Fig. 4a) onto which each stream in $\mathcal{S}^\star$ is routed, fulfilling the fault-tolerance requirements

[2] The number of commonly used links should be 0. For example, if $R_1 = G(V_1, E_1)$ and $R_2 = G(V_2, E_2)$ represent the multicast trees of two redundant copies of the same stream, then $E_1 \cap E_2 = \emptyset$

Id	Cost	No. int. ports	No. ext. ports
bt_1	8	2	3
bt_2	10	1	4
bt_3	16	2	5

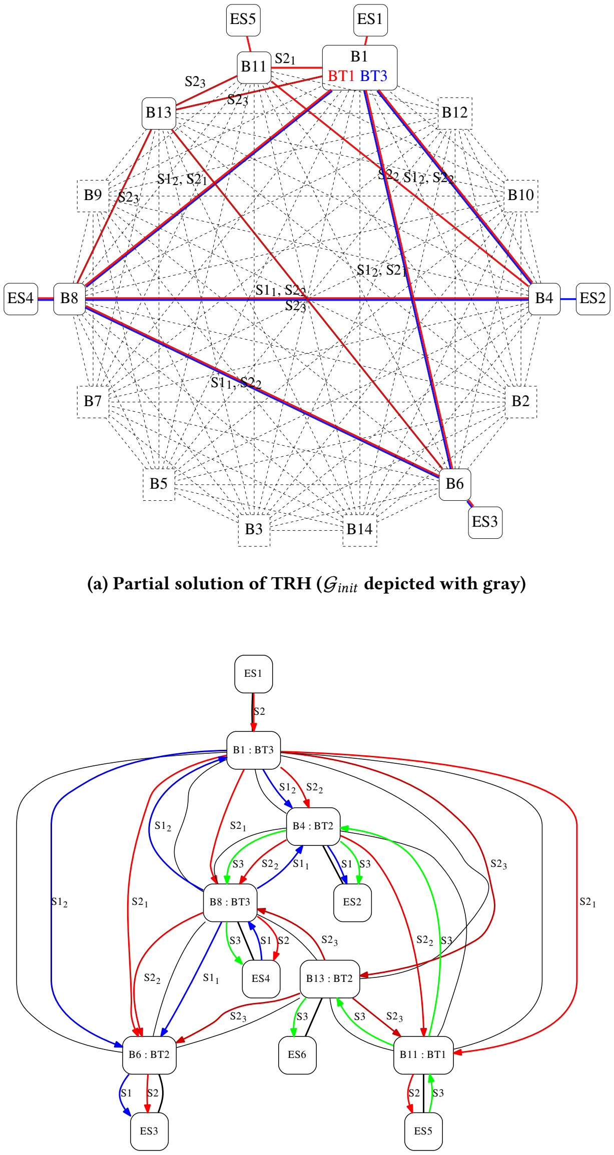
Table 3: Library for the TRH example

(the for loop in lines 3 to 11). Then, we remove from $\mathcal{G}_{init}$ the physical links and the bridges which are not used by the routing (line 12). TRH is a heuristic that does not guarantee finding the optimal solution, and it may terminate without finding a solution, even if one exists. The function *SearchRoute* returns a route for $mt_s(s_i)$ for a stream s_i. The idea of our heuristic is to keep track of the physical links used by the routes in $\mathcal{E}_{used}$, found for the already visited streams in Alg. 1. $\mathcal{E}_{used}$ encourage subsequent calls to *SearchRoute* to reuse already used physical links as long as the fault-tolerance constraints are satisfied. TRH does not directly attempt to reduce the architecture cost $cost_A$ during this process, and it does not check the schedulability. Schedulability is checked in line 13 and $cost_A$ is reported for the constructed solution at the end.

$\mathcal{G}_{init}$ is obtained as explained in Sect. 5.1 for Fig. 4a. Note that when assuming the bridge type for each bridge such that it accommodates the required ports, we also use bridge chaining if necessary, assigning the lowest-cost chain we can construct (see Sect. 3 and Fig. 2). Let us consider an example input with 6 ESes and the library $\mathcal{L}$ from Table 3, with the same link types presented in Table 1. Let us assume that the ESes ES_1 and ES_4 can be connected only to bridges of types bt_1 and bt_3, for ES_2, ES_3 and ES_6 are available bt_2 and bt_3 and ES_5 can be connected to bridge of type ES_1. Then, the $\mathcal{G}_{init}$ for this step is presented in Fig. 5a.

TRH iterates through $s_i \in \mathcal{S}$ and determines the routes, lines 3-11 in Alg. 1. We sort the streams in $\mathcal{S}$ based on the timing properties and RL (the aim is to route the most critical streams first). Moreover, when searching for a route for a redundant copy of a stream s_i^j we want to make sure that this route is disjoint to all the redundant routes established for s_i. Hence, we use the $\mathcal{G}_{ft}$ graph to keep track of already used links, removing from $\mathcal{G}_{ft}$ the edges involved in each $mt_s(s_i)$ determined so far (line 7). Thus, these edges will not be used in subsequent redundant routes for s_i. The route for s_i is searched on $\mathcal{G}_{ft}$, which for each original route $s_i \in \mathcal{S}$ considered is initialized to $\mathcal{G}_{init}$, where each undirected physical link is converted to two directed data flow links.

The routes are found using the *SearchRoute* function, which is an adapted Breadth-First Search (BFS) algorithm, presented in Alg. 2. The *SearchRoute* function attempts to reuse as much as possible the edges used by previously determined routes, hence we keep track of the edges used so far in $\mathcal{E}_{used}$. After we determine a route, we update the bridge type for bridges in $\mathcal{G}_{init}$ by selecting from the library the bridge type of minimum cost which has the required number of ports. Fig. 5a shows a partial solution where we iterated over streams s_1 and s_2 and than determined the links and the stream's routes. The final step of TRH is to remove from $\mathcal{G}_{init}$ the edges not used for routes and to remove any vertices that became thus isolated in the topology (unused bridges), line 12 in Alg. 1.



(a) Partial solution of TRH ($\mathcal{G}_{init}$ depicted with gray)

(b) Final solution of TRH

Figure 5: TRH example

We modified BFS in *SearchRoute* function such that we visit a dataflow link during search only if *IsVisitable* returns true. *IsVisitable* returns true, if at least one of the following conditions holds: (1) the link was already used for the already determined routes or (2) the source and target bridges have enough free ports to support the addition of this link and of the next possible redundant copies. *IsVisitable* will return false if the dataflow link will exceed its capacity by routing s_i^j. In Alg. 2 the function *Target* applied on a dataflow link gives the vertex on which the link enters and the elements of queue q are dataflow paths, therefore we used the function *LastVertex* to retrieve end of a dataflow path. The number of ports for a bridge is determined by summing up the number of physical links incident to that bridge that were used for already determined routings (stored in $\mathcal{E}_{used}$) and that are used for the current routing. If the stream for which we are searching the route is not the last one from the set of its redundant copies to the number determined

Algorithm 2: SearchRoute

```
Input: $\mathcal{G}_{ft}$, the set of $\mathcal{E}_{used}$, the stream $s_i^j$, the remaining redundancy $rl'$
Output: The route $mt_s$
$q \leftarrow \{s_i^j.src\}$
$dests \leftarrow s_i^j.dests$
$visited \leftarrow \emptyset$ ; $\mathcal{G}_{current} \leftarrow \emptyset$ ; $paths \leftarrow \emptyset$
while $q \neq \emptyset$ AND $dests \neq \emptyset$ do
    choose first element of q as $current$
    $successors \leftarrow Successors(\mathcal{G}_{ft}, LastVertex(current))$
    for $succ : successors$ do
        if $Target(succ) \notin visited$ AND
        $IsVisitable(succ, \mathcal{E}_{used}, \mathcal{G}_{current}, rl', \mathcal{G}_{ft})$ then
            $newPath \leftarrow current + succ$
            if $Target(succ) \notin \mathcal{ES}$ then
                $q \leftarrow q \cup \{newPath\}$
            else if $Target(succ) \in dests$ then
                $dests \leftarrow dests \setminus \{Target(succ)\}$
                $paths \leftarrow paths \cup \{newPath\}$
                $\mathcal{G}_{current} \leftarrow \mathcal{G}_{current} \cup Edges(newPath)$
            end
        end
    end
    $visited \leftarrow visited \cup \{Target(succ)$
end
return $ConvertToTree(paths)$
```

before is added the number of remaining redundant copies, rl' in Alg. 2.

Let us consider that for the TRH example considered earlier we have three streams, s_1 to s_3. Furthermore, let us assume that s_1 is sent from ES_4 to ES_2 and ES_3, s_2 from ES_1 to ES_3, ES_4 and ES_5 and s_3 is sent from ES_5 to ES_2, ES_4 and ES_6. For this example we consider that only first two streams are fault-tolerant, with a redundancy level of 2 and 3 for s_1 and s_2, respectively. Fig. 5b shows the final solution of TRH for our example, i.e., the routes and the network from which the unused physical links and bridges are removed. The physical links are depicted with solid black lines (the internal links use thicker lines) and the routes are depicted with colored thin arrows. We used blue arrows for routes of s_1, red for s_2 and green for the route of s_3. The networks in Fig. 5 have been generated by our tool, and redundant streams s_i^j are labeled as $\mathrm{S_{ij}}$.

6.3 GRASP

GRASP [7] is a meta-heuristic optimization, which searches for that solution which minimizes the $cost_T$ function. GRASP is an iterative algorithm, where each iteration consists of two phases: Phase (i) constructs an initial solution (a topology and a route for each stream $s_i \in \mathcal{S}^\star$) based on a randomized greedy algorithm and Phase (ii) performs a local search on the constructed solution to reach the local minimum. At the end of each iteration, if the cost of the local minimum found is less than the cost of the best solution, found so far, the solution is stored as the "best-so-far". The termination condition for the strategy is based on a given time limit. We implemented GRASP with Google OR-Tools.

To construct the solutions in Phase (i), we have adapted our TRH strategy as follows. First, we create initial solutions by creating a random ordering of streams at the start of Alg. 1 (with TRH, the streams are ordered based on their "criticality" of timing and RL). We use the same *SearchRoute* function (Alg. 2). Then, we also create initial solutions which do not use the routes returned by *SearchRoute*, but instead use random routes, in the hope of providing a better coverage of the search space.

In Phase (ii), starting from each such initial solution, we search for a local minimum using the Large Neighborhood Search (LNS) algorithm [24], which improves the initial solutions by iteratively "destroying" and "repairing" the solution.

For the destroy part we use 6 operators which remove links from the topology (removing all routes routed over that links) or remove routes. The operators are as follows: (1) remove a link, (2) remove two routes, (3) remove a route with the containing links and routes routed over these links, (4) select one stream $s_i \in \mathcal{S}^\star$ and remove the routes for its original stream and redundant copies, (5) select two streams $s_i, s_j \in \mathcal{S}^\star$ and remove the routes for their original streams and redundant copies and (6) select two original streams $s_i, s_j \in \mathcal{S}$ and for each stream remove the route for a randomly chosen redundant copy. The *degree of destruction* is such that we are able to explore the neighborhood in a reasonable time, but we do not degrade into a full optimization.

Several options are possible for the "repair". We have decided to use an "optimal repair", i.e., the best possible full solution is constructed from the partial solution. To find the solution for the repair we have used the CP-Strategy presented in the next section.

6.4 Constraint Programming-Based Strategy

The Topology and Routing Optimization (TRO) strategy is a Constraint Programming (CP) [3] model. TRO takes as input the set $\mathcal{ES}$, the components library $\mathcal{L}$ and the set $\mathcal{S}$, and attempts to determine the optimal network architecture $\mathcal{G}$ and the routing *SR* according to the cost function introduced in Sect. 6.1. Moreover, it can take $\mathcal{G}$ as an additional input and attempts to only determine the optimal routing *SR* for the given architecture. In the following, we present a CP model for the problem described in Sect. 5.

6.4.1 CP Model. We use two parameters: (1) $n = |\mathcal{ES}|$ the number of End Systems. (2) n_{max} = the maximum number of bridges, which can be given or computed as explained in Sect. 6.2 for Fig. 5a. Based on these parameters we define the following sets:

- K End Systems index set, $K = \{1, .., n\}$
- J bridges index set, $J = \{n + 1, n + 2, .., n_{max}\}$
- I End Systems and bridges index set, $I = K \cup J$

To model the topology of the network let $\mathcal{A}_{i1,i2}$ denote the adjacency matrix of the $\mathcal{G}_{max}(\mathcal{V}_{\max}, \mathcal{E}_{\max})$ which has $n + n_{max}$ vertices. Each element of this matrix is an integer variable that represents the type of the link between nodes of this graph, $\forall\ i1, i2 \in I,\ a_{i1,i2} \in \{0, .., |\mathcal{LT}|\}, lt_{a_{i1,i2}} \in \mathcal{LT}$ (0 = no link). Since the topology graph is undirected, we only initialize new variables for the right-upper half of the matrix elements. The elements on the other half of the matrix point to their symmetric elements, therefore $\forall\ i1, i2 \in I,\ a_{i1,i2} = a_{i2,i1}$. Since the graph should not contain self-loop links, we set the elements on the main diagonal of the matrix to 0, $\forall\ i \in I,\ a_{i,i} = 0$. We also define $\mathcal{BA}_j$ which is an integer vector to specify the type of the bridges in the network. For each bridge $j \in J$, variable ba_j specifies the type of the bridge ($ba_j \in \{1, .., |\mathcal{BT}|\} \cup \{nil\}, bt_{ba_j} \in \mathcal{BT}$). If this value is *nil* for a bridge j, it means that the bridge is not active and it is not included in the network topology.

In order to model the stream's routes, we define two integer matrices $\mathcal{MT}^{s}_{s^\star,i}$ and $\mathcal{MT}^{w}_{s^\star,i}$. Both matrices have the same size and dimensions. The size of the first dimension is the size of all the streams including their redundant copies ($|\mathcal{S}^\star|$), and the size of the second dimension is the size of all the vertices including End Systems and bridges ($|I|$). For each stream (including the redundant copies $s \in \mathcal{S}^\star$), $\mathcal{MT}^{s}_{s}$ is an integer vector that specifies a multicast tree for the stream. Each element of this vector ($\forall\ i \in I, \mathcal{MT}^{s}_{s,i} \in I \cup \{nil\}$), holds the successor vertex on the path from that vertex (i) to the source of the stream (s). In the same manner, each element of $\mathcal{MT}^{w}_{s,i}$ holds the weight of the path from that vertex (i) to the source of the stream (s), $\forall\ i \in I, \mathcal{MT}^{w}_{s,i} \in \mathbb{R}$. The value of $\mathcal{MT}^{s}_{s}$ for the source of the stream is set to the index of the source End System, $\mathcal{MT}^{s}_{s,s.src} = s.src$ and $\mathcal{MT}^{w}_{s,s.src} = 0$. If a bridge is not part of the multicast tree then $\mathcal{MT}^{s}_{s,i} = nil$ and $\mathcal{MT}^{w}_{s,i} = -1$. Our mathematical model and constraints for multicast tree are an extension of the models presented in [21].

According to these variables, we define the following constraints.

6.4.2 *Topology Constraints.*

(1) Any End System, $ES \in \mathcal{ES}$, must be connected with one bridge: $\forall\ k \in K, \sum_{j \in J}(a_{k,j} \neq 0) = 1$
(2) The specified bridge constraint BC should be satisfied: $\forall\ k \in K, j \in J,\ a_{k,j} \neq 0 \Rightarrow bt_{ba_j}, l_{a_{k,j}} \in BC(ES_k)$
(3) The number of internal links connected to a bridge should not exceed the number of internal ports supported by the bridge: $\forall j \in J \sum_{i \in I, a_{i,j} \neq 0}(l_{a_{i,j}}.lct = Int) \leq bt_{ba_j}.noIntPorts$
(4) The number of external links connected to a bridge should not exceed the number of external ports supported by the bridge: $\forall j \in J \sum_{i \in I, a_{i,j} \neq 0}(l_{a_{i,j}}.lct = Ext) \leq bt_{ba_j}.noExtPorts$
(5) If a bridge is inactive, it cannot be connected to other bridges or end systems: $\forall j \in J, \sum_{i \in I} a_{i,j} = 0 \Leftrightarrow ba_j = nil$
(6) All the active bridges should be connected at least via two links (no bridge is an end point): $\forall j \in J \sum_{i \in I, i \neq j}(0 < a_{i,j}) \geq 2 \times (ba_j \neq nil)$

6.4.3 *Routing Constraints.*

(1) The successors of destinations of a stream cannot be *nil*: $\forall s \in \mathcal{S}, \forall d \in s.dests\ \mathcal{MT}^{s}_{s,d} \neq nil$
(2) The successors of nodes which are not the source should not point to themselves: $\forall_{i \in I}, \forall_{s \in \mathcal{S}^\star, i \neq s.src} \mathcal{MT}^{s}_{s,i} \neq i$
(3) The successors of End Systems which are neither source nor destinations of a stream must be *nil*: $\forall\ s \in \mathcal{S}, \forall\ k \in K \backslash \{s.src\} \backslash s.dests\ \mathcal{MT}^{s}_{s,k} = nil$
(4) Inactive bridges cannot be used within a multicast tree: $\forall j \in J \sum_{s \in \mathcal{S}^\star} \left(\mathcal{MT}^{s}_{s,j} \neq nil\right) \neq 0 \Leftrightarrow ba_j \neq nil$
(5) The multicast trees must not contain cycles: $\forall_{s \in \mathcal{S}, i \in I \backslash \{s.src\}}$ $\mathcal{MT}^{s}_{s,i} \neq nil \Rightarrow \mathcal{MT}^{w}_{s,i} = \mathcal{MT}^{w}_{s, \mathcal{MT}^{s}_{s,i}}) + 1$
(6) All the bridges should be transient, which means that if a stream enters to a bridge through a link it should exit from another link connected to the bridge: $\forall\ s \in \mathcal{S}, i \in I, \exists j \in J, i \neq j$ s.t. $\mathcal{MT}^{s}_{s,i} \neq nil \Leftrightarrow \mathcal{MT}^{s}_{s,j} = i$
(7) Any vertex and its successor should be connected: $\forall\ s \in \mathcal{S}, \forall i, j \in I\ \mathcal{MT}^{s}_{s,i} = j \Rightarrow a_{i,j} \neq 0$
(8) All the multicast trees for the redundant copies of each stream should not have common links: $\forall\ s \in \mathcal{S}, \forall v \in I \backslash \{s.src_s\} \backslash s.dests,\ \forall i, j = 1, \ldots, s.rl\ i \neq j \wedge i \neq nil \wedge j \neq nil \Rightarrow \mathcal{MT}^{s}_{s^i,v} \neq \mathcal{MT}^{s}_{s^j,v}$
(9) For all physical links, the capacity of the links should not be exceeded: $\forall_{i,j \in I} \sum\limits_{s \in \mathcal{S}^\star} \left((\mathcal{MT}^{s}_{s,j} = i) \times s.R\right) \leq l_{a_{i,j}}.speed$

6.4.4 Search Strategy. We define $\mathcal{MT}^{s}_{s^\star,i}$ and $\mathcal{BA}_j$ as the main decision variables. Consequently, the assignments of $\mathcal{A}_{i,i}$ and $\mathcal{MT}^{w}_{s^\star,i}$ will be determined by propagating the constraints (8) and (6). In the case that the architecture is given as input, we initialize the values of $\mathcal{A}_{i,i}$ and $\mathcal{BA}_j$ according to the given architecture. Thus, the solver will do the exhaustive search only for $\mathcal{MT}^{s}_{s^\star,i}$.

Two strategies should be specified for the CP solver to perform the search. The first is the order of selecting the variables for assignment. The other strategy is the order of selecting the values from the variable's domain for assignment. Based on the results obtained for small case studies, we decide to use First-Unbound-Variable and Assign-Min-Value strategies for the decision variables.

To validate the schedulability constraint and guide the solver to find the optimal solution, we implemented a Search-Monitor (the term used in OR-Tools) that will be triggered whenever the CP solver finds a solution (which satisfies all the constraints). The search-monitor computes the degree of schedulability δ_t and the architecture cost $cost_A$ of the obtained solution. If the degree of schedulability is less or equal to zero, it will consider the solution as a feasible solution, and if the total cost $cost_T$ of the solution is less than the cost of the earlier solutions, it will consider the solution as the best solution obtained so far. At the end of the search process, we will return the best solution found by converting the assignments of $\mathcal{A}_{i1,i2}$ and $\mathcal{BA}_j$ into the network architecture $\mathcal{G}$, and $\mathcal{MT}^{s}_{s^\star,i}$ into the routing *SR* obtained for the given set of streams.

7 EXPERIMENTAL RESULTS

For the evaluation of our strategies, namely the Topology and Routing Heuristic (TRH), which is a heuristic approach, GRASP (a metaheuristic strategy), and the Topology and Routing Optimization (TRO), which is a CP-based strategy, we used five synthetic test-cases, *motiv*, *tc1* to *tc4* and *GM*, which is a real-life case study from General Motors. For all experiments, as the components library for the link types we used those presented in Table 1, but for bridges, we have extended the library to contain 6 bridge types. For these experiments we considered that all streams have the same priority. In Table 4 the first 3 columns describe the test-case, i.e., name, number of end systems, and the number of streams and their redundant copies. The strategies, TRH, GRASP and TRO were implemented in Java and all experiments were run on Intel Core i7-2600 machines at 3.4 GHz. For TRO and GRASP we have used OR-Tools [34], which is a CP library introduced by Google.

The results of our experiments are shown in Table 4. Column 4 shows the cost of the fully connected topology $\mathcal{G}_{init}$ (see Sect. 6.2), which is an upper bound on the architecture cost. For all strategies, we present the architecture cost and the execution time. These strategies were able to find schedulable solutions, for all test-cases.

As it can be observed in Table 4, our strategies are able to significantly reduce the architecture cost $cost_A$. Compared to $\mathcal{G}_{init}$ TRH is able to decrease the architecture cost, in average with 37%, with a maximum decrease in cost for *GM*, where the cost is reduced by 80%. Although TRH is able to decrease architecture cost compared

Name	$\lvert\mathcal{ES}\rvert$	$\lvert\mathcal{S}\rvert$, $\lvert\mathcal{S}^{\star}\rvert$	$\mathcal{G}_{init}$	*TRH*		*GRASP*		*TRO*	
			$cost_A$	$cost_A$	Exec. time (s)	$cost_A$	Exec. time (s)	$cost_A$	Exec. time (s)
motiv	4	3, 4	78	63	0.15	*43	0.67	*43	1.32
tc1	4	4, 6	78	78	0.09	50	0.40	*41	11.84
tc2	4	8, 11	78	71	0.08	52	0.60	*41	32.6
tc3	6	6, 8	176	106	0.1	76	0.95	73	48 h
tc4	15	30, 34	1893	392	8.05	336	3.35 min	357	48 h
GM	20	27, 38	2230	432	130	402	9.3 min	410	48 h

Table 4: Experimental results

to $\mathcal{G}_{init}$ and scales well with the problem size, it obtains solutions with a higher cost compared to GRASP and TRO. For example, TRO improves on average by 27% on the TRH architecture cost, and GRASP improves on average by 23% compared to TRH.

TRO is able to find the optimum solution, marked by * in the table. However, TRO finds the optimum solution only for the smaller test-cases *motiv*, *tc1* and *tc2*, and it does not scale well with the problem size. For the other test-cases, TRO did not find the optimum solution and we list in the table the architecture cost found after a time limit (execution time) of 48 hours that we impose on the search.

Finally, our proposed GRASP strategy is able to obtain good quality results in a reasonable time. As we can see, GRASP obtains optimum solution for *motiv* test-case, and for the other test-cases, it obtains results comparatively close to the TRO with the relative gap of 5% on average. The main advantage of GRASP is that it can explore the design space much faster. For example, for the realistic test case GM, GRASP has obtained an architecture cost of 402 in 9.3 minutes, compared to TRO, which has actually obtained a larger cost of 410 in the 48 hours we let it run.

8 CONCLUSIONS

In this paper, we have considered safety-critical real-time applications implemented using TSN-based distributed architectures. Our focus was on the synthesis of the network topology and streams routing such that the real-time and redundancy requirements of the applications are satisfied, and the cost of the architecture is minimized. We have proposed three strategies, a heuristic approach, called Topology and Routing Heuristic, a GRASP metaheuristic and an approach based on CP, namely Topology and Routing Optimization. The experimental results show that by using our strategies we are able to significantly reduce the cost of architecture, obtaining architectures which are at the same time fault-tolerant and meet the timing requirements of the streams. In particular, the proposed GRASP metaheuristic is able to obtain good quality results in a reasonable time, and scales well with the problem size.

ACKNOWLEDGMENTS

This work has been conducted within the ENABLE-S3 project that has received funding from the ECSEL Joint Undertaking under grant agreement No 692455.

REFERENCES

[1] Ahmad Al Sheikh, Olivier Brun, Maxime Chéramy, and Pierre-Emmanuel Hladik. 2013. Optimal design of virtual links in AFDX networks. *Real-Time Systems* 49, 3 (2013), 308–336.

[2] Bjoern Annighoefer, Caroline Reif, and Frank Thieleck. 2014. Network topology optimization for distributed integrated modular avionics. In *Digital Avionics Systems Conference.* 4A1–1.

[3] Krzysztof Apt. 2003. *Principles of constraint programming.* Cambridge U. Press.

[4] Guy Avni, Shibashis Guha, and Guillermo Rodriguez-Navas. 2016. Synthesizing Time-Triggered Schedules for Switched Networks with Faulty Links. In *Proceedings of the 13th International Conference on Embedded Software.* 1–10.

[5] Rodney Cummings, Kai Richter, Rolf Ernst, Jonas Diemer, and Arkadeb Ghosal. 2012. Exploring Use of Ethernet for In-Vehicle Control Applications: AFDX, TTEthernet, EtherCAT, and AVB. *SAE International Journal of Passenger Cars - Electronic and Electrical Systems* 5, 1 (2012), 72–88.

[6] Jean-Dominique Decotignie. 2005. Ethernet-based real-time and industrial communications. *Proc. IEEE* 93, 6 (2005), 1102–1117.

[7] Thomas A Feo and Mauricio G. C Resende. 1989. A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem. *O.R. Letters* (1989).

[8] Voica Gavrilut and Paul Pop. 2016. Traffic class assignment for mixed-criticality frames in TTEthernet. *ACM Sigbed Review* 13, 4 (2016), 31–36.

[9] Voica Gavrilut, Domiţian Tămaş-Selicean, and Paul Pop. 2015. Fault-Tolerant Topology Selection for TTEthernet Networks. In *Proceedings of the Safety and Reliability of Complex Engineered Systems Conference.* 4001–4009.

[10] Tibebu Gissila. 2013. *Connectors and Vibrations–Damages in Different Electrical Environments.* Master’s thesis. Blekinge Institute of Technology.

[11] Miltos D. Grammatikakis, D. Frank Hsu, Miro Kraetzl, and Jop F. Sibeyn. 1998. Packet Routing in Fixed-Connection Networks: A Survey. *J. Parallel and Distrib. Comput.* 54, 2 (1998), 77–132.

[12] Thomas Herpel, Bernhard Kloiber, Reinhard German, and Steffen Fey. 2009. Routing of Safety-Relevant Messages in Automotive ECU Networks. In *Vehicular Technology Conference Fall.* 1–5.

[13] IEEE. 2010. *IEEE 802.1Qat - IEEE Standard for Local and Metropolitan Area Networks - Virtual Bridged Local Area Networks Amendment 14: Stream Reservation Protocol.* The Institute of Electrical and Electronics Engineers, Inc.

[14] IEEE. 2012. *IEEE 802.3 - IEEE Standard for Ethernet.* The Institute of Electrical and Electronics Engineers, Inc.

[15] Abdullah Konak and Alice E Smith. 2006. Network reliability optimization. In *Handbook of Optimization in Telecommunications.* Springer, 735–760.

[16] Nicolas Krommenacker, Eric Rondeau, and Thierry Divoux. 2002. Genetic algorithms for industrial Ethernet network design. In *Proc. of IEEE International Workshop on Factory Communication Systems.* 149–156.

[17] Way Kuo and V Rajendra Prasad. 2000. An annotated overview of system-reliability optimization. *Reliability, IEEE Transactions on* 49, 2 (2000), 176–187.

[18] Sune Mølgaard Laursen, Paul Pop, and Wilfried Steiner. 2016. Routing Optimization of AVB Streams in TSN Networks. *SIGBED Review* (2016).

[19] On-road Automated Vehicle Standards Committee. 2014. *SAE J3016: Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems.* Technical Report. SAE International.

[20] Paulo Pedreiras and Luis Almeida. 2004. Message routing in multi-segment FTT networks: the isochronous approach. In *Proc. of Parallel and Distributed Processing Symposium.* 122–129.

[21] Quang Dung Pham and Yves Deville. 2012. Solving the quorumcast routing problem by constraint programming. *Constraints* 17, 4 (2012), 409–431.

[22] John Rushby. 2001. *A comparison of bus architectures for safety-critical embedded systems.* Technical Report. Computer Science Laboratory, SRI International.

[23] Stefan Schneele and Fabien Geyer. 2012. Comparison of IEEE AVB and AFDX. In *Proceedings of the Digital Avionics Systems Conference.*

[24] Paul Shaw. 1998. Using constraint programming and local search methods to solve vehicle routing problems. In *International Conference on Principles and Practice of Constraint Programming.* Springer, 417–431.

[25] Siemens AG. 2004. *SN 29500-5: Failure rates of components expected values. Section 5. Expected values for electrical connections, connectors and sockets.* Technical Report SN 29500-5. Siemens AG. https://www.isograph.com/software/reliability-workbench/reliability-prediction/siemens-sn-29500/

[26] Johannes Specht and Soheil Samii. 2016. Urgency-based scheduler for time-sensitive switched ethernet networks. In *2016 28th Euromicro Conference on Real-time Systems (ECRTS).* IEEE, 75–85.

[27] Domiţian Tămaş-Selicean, Paul Pop, and Jan Madsen. 2014. *Design of Mixed-Criticality Applications on Distributed Real-Time Systems.* Ph.D. Dissertation. Technical University of Denmark.

[28] D. Tămaş-Selicean, Paul Pop, and Wilfried Steiner. 2014. Design optimization of TTEthernet-based distributed real-time systems. *Real-Time Systems* (2014).

[29] TSN Task Group. 2015. IEEE 802.1Qbv/D3.1 : Enhancements for Scheduled Traffic. (2015). http://www.ieee802.org/1/pages/802.1bv.html

[30] TSN Task Group. 2016. IEEE 802.1Qcc/D0.5 : Stream Reservation Protocol (SRP) Enhancements and Performance Improvements. (2016). http://www.ieee802.org/1/pages/802.1cc.html

[31] TSN Task Group. 2017. IEEE 802.1CB/D2.8: Frame Replication and Elimination for Reliability. (2017). http://www.ieee802.org/1/pages/802.1CB.html

[32] TSN Task Group. 2017. IEEE 802.1Qcr/D0.0 : Bridges and Bridged Networks Amendment: Asynchronous Traffic Shaping. (2017). http://www.ieee802.org/1/pages/802.1cr.html

[33] TSN Task Group. 2017. Time-Sensitive Networking. (2017). http://www.ieee802.org/1/pages/tsn.html

[34] Nikolaj van Omme, Laurent Perron, and Vincent Furnon. 2016. *OR-Tools user’s manual.* Technical Report. Google.

[35] Bin Wang and J. C. Hou. 2000. Multicast Routing and its QoS Extension: Problems, Algorithms, and Protocols. *IEEE Network* 14, 1 (2000), 22–36.