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

Link back to DTU Orbit

Citation (APA):
Fault-Tolerant Topology and Routing Synthesis for IEEE Time-Sensitive Networking

Voica Gavrilut, 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.

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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

RTNS ’17, October 4–6, 2017, Grenoble, France
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-5286-4/17/10...
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
Researches 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 proposed 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. Tâmâz-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 $G(V,E)$, where the vertices (or nodes) $V = ES \cup B$ denote the set of all End Systems (ESes) and network switches, usually denoted in TSN as Bridges (Bs), respectively. The edges $E$ 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, $B$ to be used and the physical interconnections. We assume that the system engineer provides a network component library $L$.
Fault-Tolerant Topology and Routing Synthesis for IEEE Time-Sensitive Networking

including a set of bridge types $BT$ and a set of physical link types $LT$. 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 $L = (BT \times LT \times BC)$, where $BT$ is a set of bridge types, and $LT$ 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 packed 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 $bt_{2s}$, which has 1 internal and 3 external ports.

![Diagram](image)

**Figure 1: Architecture model example**

<table>
<thead>
<tr>
<th>Id</th>
<th>No. int. ports</th>
<th>No. ext. ports</th>
</tr>
</thead>
<tbody>
<tr>
<td>$bt_1$</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>$bt_2$</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>$bt_3$</td>
<td>10</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Id</th>
<th>Cost (Mbit/s)</th>
<th>Speed (Mbit/s)</th>
<th>Int./Ext.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$lt_1$</td>
<td>7</td>
<td>100</td>
<td>Ext.</td>
</tr>
<tr>
<td>$lt_2$</td>
<td>1</td>
<td>1000</td>
<td>Int.</td>
</tr>
</tbody>
</table>

**Table 1: Example library $L$**

We use two functions to specify the type of the bridges and the physical links used within the architecture network. The first function $BT : B \mapsto BT$, specifies the type of a bridge, e.g., $BT(b_1) = bt_2$ in Fig. 2. The other function $LT : E \mapsto LT$, specifies the type for each link in the network topology, e.g., $LT(l_1) = l_{t1}$. 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 physical 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 $bt_1$ and $bt_1$, respectively.

We distinguish between two types of connections: internal links, which are between the MIs 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 $l_i, l_i \in E$, 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.

A dataflow link (DL) $dl_i$ 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 $[(ES_1 - B_1), (B_1 - B_2), (B_2 - ES_2)]$. The set of all DLs is denoted with $DL$ and the set of all DPs is $DP$.

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 through streams. A stream is denoted as $s_i$ and the set of all streams is denoted $S$. 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 $s_i \in S$ 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:
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 assigns the priority of each stream 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.

<table>
<thead>
<tr>
<th>Id</th>
<th>Src.</th>
<th>Dests.</th>
<th>B in B</th>
<th>R in ms</th>
<th>D in ms</th>
<th>rl</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>ES1</td>
<td>ES3, ES4</td>
<td>150</td>
<td>15</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>s2</td>
<td>ES2</td>
<td>ES3, ES4</td>
<td>100</td>
<td>10</td>
<td>4.5</td>
<td>1</td>
</tr>
<tr>
<td>s3</td>
<td>ES4</td>
<td>ES1, ES2</td>
<td>100</td>
<td>10</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 2: Application model example

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.

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

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 P802.1Qbr [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 most common types of permanent hardware failures are 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.
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 $q_{L1}^H$, $q_{L2}^H$, $q_{L1}^L$, $q_{L2}^L$, and $q_{L3}^L$, 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 $q_{L1}^H$, $q_{L2}^H$, and $q_{L3}^H$ 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 $q_{L1}^L$, $q_{L2}^L$, and $q_{L3}^L$ 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 $s_j$ 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 lowest 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 $s_j \in S^*$.  

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 $ES$, (2) the library of components $L$, (3) the set of streams $S$ for which we know the source, destination(s), and timing properties as well as the desired redundancy level $s_j, r_l$. We are interested in determining an optimized solution $Sol = (G, SR)$, where $G$ is the network architecture and $SR : S^* \rightarrow M T_D$ 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, $b_{TT}$ to $b_{CBS}$, with types properties presented in Table 1 and where all ESs 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.
we normalize the two values. For both, $\delta_1$ 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 $\psi_{sched}$ is significantly higher than the architecture penalty $\psi_{topo}$.

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

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

(2)

The degree of schedulability $\delta_1$ 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 $\delta_1(SR)$ as follows:

$$\delta_1(SR) = \begin{cases} \text{if at least one stream is not schedulable} & \sum_{s_i \in S^*, WCD(s_i) > s_i.D} WCD(s_i) - s_i.D \\ & \sum_{s_i \in S^*} WCD(s_i) - s_i.D \text{ otherwise} \end{cases}$$

(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 $E, S$, components library $L$ and streams $S$
Output: The TSN network $G(V, E)$ and the routes $SR$
1 $G_{init} \leftarrow \text{CreateInitialTopology}$
2 $E_{used} \leftarrow \emptyset$; $MT_x \leftarrow \emptyset$
3 for $s_i : S$ do
4     $G_{init} \leftarrow \text{ConvertGraph} (G_{init})$
5     for $j \leftarrow 1$ to $s_i, r_l$ do
6         $SR(s_i^j) \leftarrow \text{SearchRoute} (G_{init}, E_{used}, s_i^j, s_i, r_l - j)$
7         $G_{init} \leftarrow \text{RemoveEdges} (G_{init}, \text{Edges}(SR(s_i^j)))$
8         $E_{used} \leftarrow E_{used} \cup \text{Edges}(SR(s_i^j))$
9     $G_{init} \leftarrow \text{AssignBridgeTypes} (G_{init}, SR(s_i^j))$
10    end
11 $G \leftarrow \text{RemoveUnusedEdgesAndVertices} (G_{init}, E_{used})$
12 $\text{CheckSchedulability} (S, MT_x)$
13 return $(G, \cost_A(G), MT_x)$

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

TRH starts from a fully connected initial solution $G_{init}$, line 1 in Alg. 1 (similar to the solution discussed in Fig. 4a) onto which each stream in $S^*$ is routed, fulfilling the fault-tolerance requirements
<table>
<thead>
<tr>
<th>Id</th>
<th>Cost</th>
<th>No. int. ports</th>
<th>No. ext. ports</th>
</tr>
</thead>
<tbody>
<tr>
<td>bt1</td>
<td>8</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>bt2</td>
<td>10</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>bt3</td>
<td>16</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 3: Library for the TRH example

Table (the for loop in lines 3 to 11). Then, we remove from $G_{init}$ 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 $mt_i(s_i)$ for a stream $s_i$. The idea of our heuristic is to keep track of the physical links used by the routes in $E_{used}$, found for the already visited streams in Alg. 1. $E_{used}$ encourages 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.

$G_{init}$ 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 $L$ 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 $ES_5$, are available $bt_2$ and $bt_3$ and $ES_6$ can be connected to bridge of type $ES_1$. Then, the $G_{init}$ for this step is presented in Fig. 5a.

TRH iterates through $s_i \in S$ and determines the routes, lines 3-11 in Alg. 1. We sort the streams in $S$ 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 $s'_i$, we want to make sure that this route is disjoint to all the redundant routes established for $s_i$. Hence, we use the $G_{init}$ graph to keep track of already used links, removing from $G_{init}$ the edges involved in each $mt_i(s_i)$ 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 $G_{init}$, which for each original route $s_i \in S$ considered is initialized to $G_{init}$, 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 $E_{used}$. After we determine a route, we update the bridge type for bridges in $G_{init}$ 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 $G_{init}$ 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.

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 $s'_i$. 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 $E_{used}$) 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...
before is added the number of remaining redundant copies, \( r' \) 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 \( ES_1 \) 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 \( ES_2 \) to \( ES_2 \), \( ES_4 \) and \( ES_5 \). 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 \( s_i' \) are labeled as \( S_{ij} \).

6.3 GRASP

GRASP [7] is a meta-heuristic optimization, which searches for that solution which minimizes the cost 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 \( s_i \in S^* \)) 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 \( s_j \in S^* \) and remove the routes for its original stream and redundant copies, (5) select two streams \( s_j, s_j' \in S^* \) and remove the routes for their original streams and redundant copies and (6) select two original streams \( s_j, s_j' \in S \) 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 \( G \), the components library \( L \) and the set \( S \), and attempts to determine the optimal network architecture \( G \) and the routing \( SR \) according to the cost function introduced in Sect. 6.1. Moreover, it can take \( G \) 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) \( n = |ES| \) 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 \( A_{i,j} \) denote the adjacency matrix of the \( G_{max} (\mathcal{V}_{max}, \mathcal{E}_{max}) \) 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, \( \forall i,j \in I, a_{i,j} \in \{0,1\} \). 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 \( \forall i,j \in I, a_{i,j} = a_{j,i}. \) 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 \( B_{j} \) which is an integer vector to specify the type of the bridges in the network. For each bridge \( j \in J \), variable \( b_{j} \) specifies the type of the bridge \( b_{j} = \{1,...,|BT|\} \cup \{nil\}, b_{j} \in BT \). 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 $MT^s_{*,i}$ and $MT^w_{*,i}$. 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 ($|S^s|$), 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 $s \in S^s$), $MT^s_{*,i}$ is an integer vector that specifies a multicast tree for the stream. Each element of this vector ($\forall i \in I$, $MT^s_{*,i} \in I \cup \{nil\}$), 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 $MT^w_{*,i}$ holds the weight of the path from that vertex to the source of the stream ($s$), $\forall i \in I$, $MT^w_{*,i} \in \mathbb{R}$. The value of $MT^s_{*,i}$ for the source of the stream is set to the index of the source End System, $MT^s_{*,src} = s.src$ and $MT^w_{*,src} = 0$. If a bridge is not part of the multicast tree then $MT^s_{*,i} = nil$ and $MT^w_{*,i} = -1$.

Our mathematical model 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, $ES \in E\mathcal{S}$, must be connected with one bridge: $\forall k \in K, \sum_{j \in J}(ak_{k,j} \neq 0) = 1$
2. The specified bridge constraint $BC$ should be satisfied: $\forall k \in K, j \in J, ak_{k,j} \neq 0 \Rightarrow bt_{ba_{k,j}} \in BC(ES_k)$
3. The number of internal links connected to a bridge should not exceed the number of external ports supported by the bridge: $\forall j \in J, lct_j \leq \sum_{i \in I}(lct_i = Ext) \leq bt_{ba_{j,extPorts}}$
4. The number of external links connected to a bridge should not exceed the number of external ports supported by the bridge: $\forall j \in J, lct_j \leq \sum_{i \in I}(lct_i = Ext) \leq bt_{ba_{j,extPorts}}$
5. If a bridge is inactive, it cannot be connected to other bridges or end systems: $\forall j \in J, \sum_{i \in I}(ai_{j,i} = 0 \Rightarrow ba_{j,i} = nil)$
6. All the active bridges should be connected at least via two links (no bridge is an end point): $\forall j \in J, \sum_{i \in I}(ai_{j,i} < 0) \geq 2 \times (ba_{j,i} \neq nil)$

### 6.4.3 Routing Constraints

1. The successors of destinations of a stream cannot be nil: $\forall s \in S, \forall d \in \{s.dests\} \Rightarrow MT^s_{*,d} = nil$
2. The successors of nodes which are not the source should not point to themselves: $\forall i \in I, s \in S^s, i \neq src \Rightarrow MT^s_{*,i} = 1$
3. The successors of End Systems which are neither source nor destinations of a stream must be nil: $\forall s \in S, \forall k \in K/\{s.src\} \Rightarrow \sum_{i \in I}(MT^s_{*,i} = nil)$
4. Inactive bridges cannot be used within a multicast tree: $\forall j \in J, \sum_{s \in S^s}(MT^s_{*,j} = nil) \geq 0 \Rightarrow ba_{j,i} = nil$
5. The multicast trees must not contain cycles: $\forall s \in S^s, i \neq src \Rightarrow MT^s_{*,i} = MT^w_{*,i} + 1$
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: $\forall s \in S, i \neq src \Rightarrow \sum_{j \neq i \in J, j \neq src}(MT^s_{*,j} = nil \Rightarrow MT^s_{*,i} = nil)$
7. Any vertex and its successor should be connected: $\forall s \in S, \forall i \in I, \forall j \neq i \Rightarrow MT^s_{*,j} = 1$
8. All the multicast trees for the redundant copies of each stream should not have common links: $\forall s \in S, \forall v \in \{s.src\}\backslash s.dests, \forall i,j \neq i \wedge i \neq nil \wedge j \neq nil \Rightarrow MT^s_{*,i} \neq MT^s_{*,j}$

### 6.4.4 Search Strategy

1. Any End System, $ES \in E\mathcal{S}$, must be connected with one bridge: $\forall k \in K, \sum_{j \in J}(ak_{k,j} \neq 0) = 1$
2. Any vertex and its successor should be connected: $\forall s \in S, i \neq src \Rightarrow MT^s_{*,i} = 1$
3. All the multicast trees for the redundant copies of each stream should not have common links: $\forall s \in S, \forall v \in \{s.src\}\backslash s.dests, \forall i,j \neq i \wedge i \neq nil \wedge j \neq nil \Rightarrow MT^s_{*,i} \neq MT^s_{*,j}$

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, namely, tc1 to tc4, and GM. We ran experiments 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 1 shows the cost of the fully connected topology $G_{max}$ (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 schedulability solutions, for all test-cases. As can be observed in Table 4, our strategies are able to significantly reduce the architecture cost $cost_A$. Compared to $G_{max}$, 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...
The table shows the performance of GRASP and TRO in terms of execution time and cost. The primary focus is on the optimization of the architecture cost and time, with GRASP being able to obtain good quality results in a reasonable time. The table is followed by a section on conclusions, which highlights the benefits of using GRASP and its scalability with problem size. The references at the end of the page list various works related to network topology and routing optimization.