Fault-Tolerant Topology Selection for TTEthernet Networks

V. Gavriluţ, D. Tămaş–Selicean & P. Pop
Technical University of Denmark
Kongens Lyngby, Denmark

ABSTRACT: Many safety-critical real-time applications are implemented using distributed architectures, composed of heterogeneous processing elements (PEs) interconnected in a network. In this paper, we are interested in the TTEthernet protocol, which is a deterministic, synchronized and congestion-free network protocol based on the IEEE 802.3 Ethernet standard and compliant with ARINC 664p7. TTEthernet supports three types of traffic: static time-triggered (TT) traffic and dynamic traffic, which is further subdivided into Rate Constrained (RC) traffic that has bounded end-to-end latencies, and Best-Effort (BE) traffic, for which no timing guarantees are provided. TTEthernet offers spatial separation through the concept of virtual links (VLs), and temporal separation, through schedule tables for TT messages and bandwidth allocation for RC messages. Given a set of PEs, we are interested to determine a fault-tolerant network topology, consisting of redundant physical links and network switches, such that the architecture cost is minimized, the applications are fault-tolerant to a given number of permanent faults occurring in the communication network, and the timing constraints of the TT and RC messages are satisfied. Deciding on a fault-tolerant topology means (i) deciding on the number of network switches, (ii) the physical links and the network topology, (iii) the routing of VLs on top of the physical network, (iv) the assignment of frames to VLs and (v) the schedule tables for the TT frames. We propose a Simulated Annealing meta-heuristic to solve this optimization problem. The proposed approach has been evaluated using a synthetic benchmark and a space case study, based on the Orion Crew Exploration Vehicle.

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), 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 (Rushby 2001). In this paper, we are interested in the TTEthernet protocol (SAE 2011).

Ethernet (IEEE 2012), although it is low cost and high speed, is known to be unsuitable for real-time and safety-critical applications (Decotignie 2005). For example, in half-duplex implementations, frame collision is unavoidable, leading to unbounded transmission times. Decotignie (2005) 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. Schneele and Geyer (2012) and Cummings et al. (2012) describe and compare several of the proposed Ethernet-based real-time communication protocols.

TTEthernet (SAE 2011) is a deterministic, synchronized and congestion-free network protocol based on the IEEE 802.3 Ethernet (IEEE 2012) standard and compliant with the ARINC 664p7 specification (ARINC 2009). ARINC 664p7 is a full-duplex Ethernet network, which emulates point-to-point connectivity over the network by defining *virtual links*, tree structures with one sender and one or several receivers (see Section 2). TTEthernet supports applications with mixed-criticality requirements in the temporal domain, providing three types of traffic: static time-triggered (TT) traffic and dynamic traffic, which is further subdivided into Rate Constrained (RC) traffic that has bounded end-to-end latencies, and Best-Effort (BE) traffic, for which no timing guarantees are provided. TT messages are transmitted based on static schedule tables and have the highest priority. RC messages are transmitted if there are no TT messages in transmission, and BE traffic has the lowest priority. TTEthernet is highly suitable for applications of different safety criticality levels, as it offers spatial separation for mixed-criticality messages through the concept of virtual links. A TTEthernet network is composed of a set of clusters. Each cluster consists of End Systems (ESes) interconnected by physical links and

Network Switches (NSes). The links are full duplex, allowing thus communication in both directions, and the networks can be multi-hop.

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 (Kuo & Prasad 2000). Konak and Smith (2006) 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 (Konak & Smith 2006).

However, these results cannot be applied directly in our case. 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 TTEthernet the routes of the virtual links for the messages are *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. TTEthernet is intended for real-time applications where message worst-case latencies have to be guaranteed.

Moreover, 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 178B refers to software for airborne systems. During the engineering of a safety-critical system, the hazards are identified and their severity is analyzed, the risks are assessed and the appropriate risk control measures are introduced to reduce the risk to an acceptable level. A *Safety-Integrity Level* (SIL) captures the required level of risk reduction. SIL allocation is typically a manual process, which is done after performing hazard and risk analysis. Depending on the standard and the SIL of the application, certain levels of redundancy are mandated for the system safety functions.

Considering the current certification practice, we assume that the engineer will specify for each application, depending on its SIL, the required *Redundancy Level*. At the level of the network topology this translates into requirements for redundant disjoint channels for communication between the ESes involved in the communication. Thus, if a physical link or a NS will fail, the other channels can still deliver the messages by their 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 (Annighoefer et al. 2014) for an avionics network. However, such a solution is too costly for most application areas, which do not require this level of redundancy for all the safety functions. Researchers have also proposed redundancy optimization solutions, e.g., (Kim & Yum 1993), but these works also assume dynamic routing. In addition, for TTEthernet, we also have to guarantee the schedulability of the messages, which depends on the configuration of the network, e.g., on the VL routes, the TT schedule tables and RC bandwidth allocation.

In (Tămaş-Selicean et al. 2014) we have considered that the network topology is given, and we have proposed a Tabu Search-based metaheuristic that optimizes the TTEthernet network configuration, such that the TT and RC messages are schedulable, and the end-to-end delay of RC messages is minimized. The configuration consists of deciding the routing of virtual links (VLs), the packing and fragmenting of messages into frames, the assignment of frames to VLs, the bandwidth of the RC VLs and the schedules for the TT messages. 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 application. We assume that the applications and ESes are given and that the designer has partitioned the messages into TT, RC or BE traffic classes, depending on the particularities of the application. We are interested to determine a fault-tolerant network topology, consisting of redundant physical links and NSes, such that the architecture cost is minimized, the applications are fault-tolerant to a given number of permanent faults occurring in the communication network, and the timing constraints of the TT and RC messages are satisfied.

2 SYSTEM MODEL

As mentioned, a TTEthernet network is composed of a set of clusters, consisting of ESes interconnected by links and NSes. An example cluster is presented in Fig. 1, where we have 4 ESes, ES_1 to ES_4, and 3 NSes, NS_1 to NS_3. The design problem addressed in this paper is performed at the cluster-level. Each ES consists of a processing element containing a CPU, memory, and a network interface card.

We model a TTEthernet cluster as an undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \mathcal{ES} \cup \mathcal{NS}$ is the set of end systems ($\mathcal{ES}$) and network switches ($\mathcal{NS}$) and $\mathcal{E}$ is the set of physical links. For Fig. 1, $\mathcal{V} = \mathcal{ES} \cup \mathcal{NS} = \{ES_1, ES_2, ES_3, ES_4\} \cup \{NS_1, NS_2, NS_3\}$, and the physical links $\mathcal{E}$ are depicted with thick, black, double arrows. Both ESes and NSes have a specified maximum number n_i of ports that can be used for connecting physical links. We define the cost of a network architecture as the total monetary cost of NSes and physical links $Cost(\mathcal{G}) = \sum_i Cost_{dl_i} + \sum_j Cost_{NS_j}$. These costs are specified by the engineer, and may include not only unit costs, but also costs related to the installation, for example.

A *dataflow path* $dp_i \in \mathcal{DP}$ is an ordered sequence of dataflow links connecting one sender to one re-

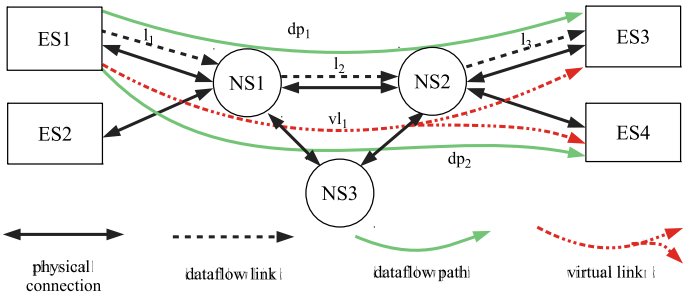


Figure 1: TTEthernet cluster example

ceiver. For example, in Fig. 1, dp_1 connects ES_1 to ES_3, while dp_2 connects ES_1 to ES_4 (the dataflow paths are depicted with green arrows). A *dataflow link* $l_i = [\nu_j, \nu_k] \in \mathcal{L}$, where $\mathcal{L}$ is the set of dataflow links in a cluster, is a directed communication connection from ν_j to ν_k, where ν_j and $\nu_k \in \mathcal{V}$ can be ESes or NSes. Using this notation, a dataflow path such as dp_1 in Fig. 1 can be denoted as [[ES_1, NS_1], [NS_1, NS_2], [NS_2, ES_3]].

The set of all messages in the system is denoted with $\mathcal{M}$. For each message we know its source ES and destination ESes. For example, let us assume that in Fig. 1 we have message m_1 transmitted from the source ES_1 to the destinations ES_3 and ES_4. The space partitioning between messages of different criticality transmitted over physical links and network switches is achieved through the concept of *virtual link*. Thus, each message $m_i \in \mathcal{M}$ is transmitted using a virtual link. The assignment of messages to virtual links is captured by the function $\mathcal{M}_F$.

We denote the set of virtual links in a cluster with $\mathcal{VL}$. A virtual link $vl_i \in \mathcal{VL}$ is a directed tree, with the sender as the root and the receivers as leafs. In our example, message m_1 is transmitted using vl_1, depicted in Fig. 1 using dot-dash red arrows, which is a tree with the root ES_1 and the leafs ES_3 and ES_4. Each virtual link is composed of a set of dataflow paths, one such dataflow path for each root-leaf connection. More formally, we denote with $\mathcal{R}_{VL}(vl_i) = \{\forall dp_j \in \mathcal{DP} | dp_j \in vl_i\}$ the routing of virtual link vl_i. For example, in Fig. 1, $\mathcal{R}_{VL}(vl_1) = \{dp_1, dp_2\}$.

TTEthernet transmits data using *frames*. The TTEthernet frame format fully complies with the ARINC 664p7 Messages are transmitted in the payload of frames. We know the size $m_i.size$ for each message $m_i \in \mathcal{M}$. Regarding real-time properties, for the TT and RC messages we know their periods and deadlines, $m_i.period$ and $m_i.deadline$, respectively. RC messages are not necessarily periodic, but have a minimum inter-arrival time. We define the rate of an RC message m_i as $m_i.rate = 1/m_i.period$. Regarding safety related properties, we assume that the engineer specifies for each critical message m_i a *redundancy level* RL_i. Thus each critical message requires that there are RL_i virtual links $VL_i^j, j \in 1..RL_i$, which have alternative paths through the topology, i.e., they do not share physical links and switches. In case of permanent failure of physical links or switches, the critical message can still be transmitted successfully, as long as no more than $RL_i - 1$ VLs are affected.

We define the sets $\mathcal{F}^{TT}$ and $\mathcal{F}^{RC}$ of TT and RC frames, respectively, and $\mathcal{F} = \mathcal{F}^{TT} \cup \mathcal{F}^{RC}$ represents all the frames in the system. In this paper we ignore the BE traffic, as it does not have any real-time or safety related properties. However, we have discussed how to take BE traffic into account during the design, such that its quality-of-service is maximized (Tamas-Selicean & Pop 2014). We explained in (Tămaş-Selicean, Pop, & Steiner 2014) how the TTEthernet protocol works, and for space reasons we will not reproduce here the details. TTEthernet has three traffic integration policies: shuffling, preemption and timely block. In this paper we consider the *timely block* policy: a low priority frame cannot be transmitted if it interferes with the transmission of a scheduled TT frame.

3 PROBLEM FORMULATION

We assume that the applications and ESes are given and that the engineer has partitioned the messages into TT, RC or BE traffic classes, depending on the particularities of the application. We also assume that each message is packed in a frame, i.e., we are not concerned in this paper with frame packing and fragmenting, which we have discussed in (Tămaş-Selicean, Pop, & Steiner 2014).

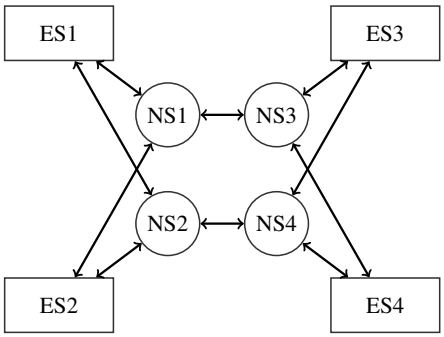
The problem we are addressing in this paper can be formulated as follows: given (1) the set of ESes in the system and the cost and the maximum number n_i of ports for both ESes and NSes, (2) the set of TT and RC frames $\mathcal{F}^{TT} \cup \mathcal{F}^{RC}$ and (3) for each message m_i the source and destination ESes, the size $m_i.size$, deadline $m_i.deadline$, period $m_i.period$ and redundancy level RL_i, we are interested to determine an optimized implementation Υ such that the architecture cost $Cost(\Upsilon)$ is minimized, the applications are fault-tolerant, considering the specified redundancy levels, and the timing constraints of all frames, both TT and RC are satisfied. Determining an implementation means deciding on the (i) network *topology* $\mathcal{G}$, i.e., the number of NSes, the physical links and how the ESes and NSes are interconnected, and the network *configuration*, which consists of (ii) the assignment $\mathcal{M}_F$ of frames to virtual links, (iii) the routing $\mathcal{R}_{VL}$ of virtual links, (iv) the bandwidth for each RC virtual link and (v) the set of TT schedule tables $\mathcal{S}$.

This paper focuses on determining (i) the network topology $\mathcal{G}$, which is motivated in the next subsection. Section 3.2 discusses how a network configuration, i.e., deciding on (ii) to (v), is determined. In Section 4 we discuss our proposed metaheuristic solution to determining the topology, further referred as Redundant Architecture Selection (RAS); to determine a network configuration within RAS, we have modified and extended the approach we have presented in (Tămaş-Selicean et al. 2014).

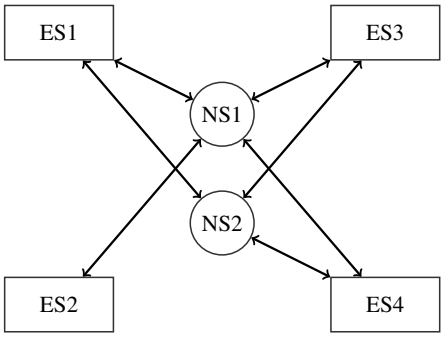
3.1 *Motivational Example*

Let us consider the example in Fig. 2, where we have four ESes, ES_1 to ES_4 and three messages, m_1 from ES_1 to ES_4, m_2 from ES_1 to ES_3 and m_3 from ES_2 to ES_4. We specify a redundancy level $RL_1 = 2$ for m_1; the rest of messages are not safety relevant.

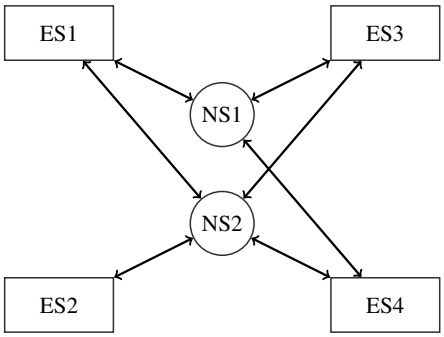
As mentioned, the redundancy levels of the messages impose redundant channels between the ESes involved in the communication. For example, if message m_1 is sent from ES_1 to ES_4, then we need two redundant channels between ES_1 and ES_4, such that if one of the channels fails (a physical link or a switch has a permanent failure), the other channel can deliver the message to the destination. Thus, a first possible fault-tolerant network topology for our system is depicted in Fig. 2(a), where we have a topology $\mathcal{G}^a$ consisting of a cascaded cluster with two redundant channels; there are four NSes, each connected to two ESes. Let us assume that m_1 and m_3 are RC and m_2 is TT and that their timing properties (size, which determines the transmission time, period, deadline) are such that the messages are schedulable on $\mathcal{G}^a$. Although the example is fault-tolerant and schedulable, the cost is high. Considering a cost of 20 units for each NS and of 10 units for each physical link, we have a total cost of $Cost(\mathcal{G}^a) = 4 \cdot 20 + 10 \cdot 10 = 180$ units.



(a) Fault-tolerant; sched.; **Cost = 180**



(b) Fault-tolerant; **not sched.**; $Cost = 110$



(c) Fault-tolerant; sched.; $Cost = 110$

Figure 2: Motivational Example

By optimizing the cost of the topology, we are able to obtain the solution $\mathcal{G}^b$ depicted in Fig. 2(b), which has a cost of 110 units. This solution is fault-tolerant because critical message m_1 has two alternative non-overlapping VLs for its transmission, that can tolerate 1 permanent failure (i.e., $RL_1 - 1$) in any of the physical links or NSes involved in the VLs. m_1 has two channels between ES_1 and ES_3 and two channels in between ES_1 and ES_4. However, the solution is not schedulable. The problem is that all messages, m_1 to m_3 have to pass through NS_1. Let us assume that their timing properties are such that they create a worst-case situation where m_3 has to wait for m_1 and m_2 such that it misses its deadline (we discuss how the end-to-end worst-case delays are determined in the next subsection).

However, by carefully deciding both the topology and the configuration of the network, we are able to obtain the solution in Fig. 2(c) which has the same low cost as the topology $\mathcal{G}^b$ (110 units), and it is both fault-tolerant and schedulable. In this solution, message m_3 reaches ES_4 via NS_2 and not the congested NS_1, thus it is able to meet its deadline. This shows that it is very important to optimize the network topology and its configuration in order to obtain low-cost fault-tolerant schedulable solutions.

3.2 *TTEthernet Configuration Example*

For a given network architecture, we need to determine a network configuration, which consists of the assignment of frames to VLs and their routing on top of the physical network, the schedule tables for the TT frames and the bandwidth for the RC frames. The configuration has to be determined such that the redundant physical channels for the critical messages are disjoint, and the frames are schedulable.

Let us consider the example in Fig. 1 where we have frame f_1 transmitted from the source ES_1 to the destinations ES_3 and ES_4. One possible assignment for f_1 to VLs is $\mathcal{M}_F(f_1) = vl_1$, with the routing $\mathcal{R}_{VL}(vl_1)$ as depicted in the figure. Note that the shortest route is not necessarily the best: it may create congestion, as discussed in the motivational example, or it may intersect with another redundant route, which is not allowed. If we assume a redundancy level $RL_1 = 2$ for f_1, the configuration in Fig. 1 is invalid, since it is not possible to have two disjoint redundant VLs for f_1.

Given a network topology $\mathcal{G}$, an assignment of frames $\mathcal{M}_F$ and their routing $\mathcal{R}_{VL}$, let us illustrate the derivation of the TT schedule tables $\mathcal{S}$, and the calculation of the worst-case end-to-end frame delays. We use the setup from Fig. 3, where we have an architecture model for a cluster composed of three ESes, ES_1 to ES_3 and a network switch NS_1 (see Fig. 3a) and

an application model with three frames, see the table in Fig. 3b. We have three virtual links, vl_1, vl_2 and vl_3 one for each frame, f_1, f_2 and f_3, respectively, as captured by the function $\mathcal{M}_F$ in the table. The periods $f_i.period$, deadlines $f_i.deadline$ and transmission times C_i on a dataflow link are given in the table for each frame. The dataflow links have the same speed, hence the C_i of a frame f_i is the same for each link.

We are interested to determine the TT schedules $\mathcal{S}$ such that all the TT and RC frames are schedulable. The schedulability of a TT frame f_i is easy to determine: we just have to check the schedules $\mathcal{S}$ to see if the times are synthesized such that the TT frame f_i is received before its deadline $f_i.deadline$. To determine the schedulability of an RC frame f_j we have to compute its worst-case end-to-end delay, from the moment it is sent to the moment it is received, which can then be compared to the deadline $f_j.deadline$ to determine if the RC frame f_j is schedulable. We denote this worst-case delay with R_{f_j}.

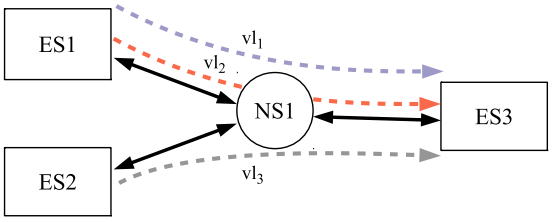
The worst-case end-to-end delay R_{f_i} of an RC frame $f_i \in \mathcal{F}^{RC}$ sent on a virtual link $vl_i = \mathcal{M}_F(f_i)$ is the sum of the worst-case queueing delays $Q_{f_i}^{[\nu_j,\nu_k]}$ on each network node (ES or NS) $\nu_j \in \mathcal{V}$ (which is the source of a dataflow link $[\nu_j, \nu_k] \in vl_i$) and the transmission duration $C_{f_i}^{[\nu_j,\nu_k]}$ for each dataflow link $[\nu_j, \nu_k] \in vl_i$ the frame transits:

$$R_{f_i} = \sum_{\substack{\nu_j,\nu_k \in \mathcal{V} \\ [\nu_j,\nu_k] \in vl_i}} (Q_{f_i}^{[\nu_j,\nu_k]} + C_{f_1}^{[\nu_j,\nu_k]}) \quad (1)$$

The worst-case queueing delay $Q_{f_i}^{[\nu_j,\nu_k]}$ of frame $f_i \in \mathcal{F}^{RC}$ transmitted on dataflow link $l_l = [\nu_j, \nu_k]$ is given by the following equation:

$$Q_{f_i}^{[\nu_j,\nu_k]} = Q_{f_i,[\nu_j,\nu_k]}^{TT} + Q_{f_i,[\nu_j,\nu_k]}^{RC} + Q_{\nu_j}^{TL} \quad (2)$$

where $Q_{f_i,[\nu_j,\nu_k]}^{TT}$ is the queueing delay due to the transmission of TT frames scheduled to be sent between the moment f_i arrives at the network node ν_j and the moment the frame instance is sent, $Q_{f_i,[\nu_j,\nu_k]}^{RC}$ is the delay caused by the RC frames that can arrive, in the



(a) Example architecture model

	period (μs)	deadline (μs)	C_i (μs)	$\mathcal{M}_F$
$f_1 \in \mathcal{F}^{RC}$	300	300	75	vl_1
$f_2 \in \mathcal{F}^{TT}$	200	200	50	vl_2
$f_3 \in \mathcal{F}^{TT}$	300	300	50	vl_3

(b) Example application model

Figure 3: Example system model

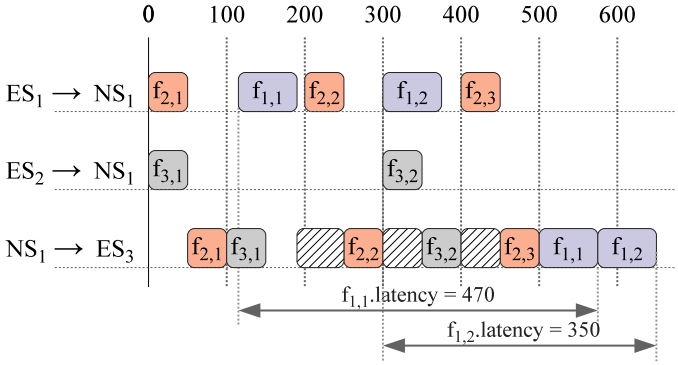


Figure 4: Worst-case scenario for RC frame f_1

worst-case, before f_i at the node and thus are placed before f_i into the outgoing queue. $Q_{\nu_j}^{TL}$ is the technical latency introduced by the network node for frame f_i, due to the hardware tasks implementing the TTEthernet protocol functionality, other than the latency resulting from queueing effects.

In (Tamas-Selicean et al. 2015) we have proposed a timing analysis for TTEthernet RC frames. However, the analysis was to slow to be integrated in our proposed network topology optimization. Hence, we have extended this analysis such that many situations that cannot occur in the worst-case are eliminated, speeding up the analysis. Since the focus of this paper is on the optimization work and for space reasons, we have decided to report this result in a future publication.

Fig. 4 presents a possible solution for synthesizing the TT schedules $\mathcal{S}$. Instead of presenting the actual schedule tables, we show a Gantt chart, which shows on a timeline from 0 to 600 μs what happens on the three dataflow links, $[ES_1, NS_1]$, $[ES_2, NS_1]$ and $[NS_1, ES_3]$. For the TT frames f_2 and f_3 the Gantt chart captures their sending times (the left edge of the rectangle) and transmission duration (the length of the rectangle).

Since the transmission of RC frames is not synchronized with the TT frames, there are many scenarios that can be depicted for f_1, depending on when f_1 is sent in relation to the schedule tables. Because we are interested in the schedulability of RC frames, for the RC frame f_1 we show in Fig. 4 the worst-case scenario, i.e., the situation which has generated the largest (worst-case) end-to-end delay. The two TT frames are schedulable. In Fig. 4 the TT schedules are constructed such that the end-to-end delay of TT frames is minimized, i.e., the TT frames arrive at their destination as soon as possible. In this case, the worst-case end-to-end delay of the RC frame f_1, namely R_{f_1}, is 470 μs, which is greater than its deadline of 300 μs, hence f_1 is not schedulable. This worst-case for f_1 happens for the first frame instance $f_{1,1}$, see Fig. 4, when $f_{1,1}$ happens to be sent by ES_1 at 105 μs. In this case, as the network implements the *timely block* integration algorithm, the frame cannot be forwarded by NS_1 to ES_3 until there is a big enough time interval to transmit the frame without

disturbing the scheduled TT frames. We denote these "blocked" time intervals with hatched boxes. The first big enough interval starts only at time 500, right after $f_{2,3}$ is received by ES_3, which is too late. However, if we instead schedule the TT frame f_3 such that its second instance $f_{3,2}$ will be sent by ES2 to NS1 at 350 μs, the worst case end-to-end delay for f_1 is reduced to 275, hence f_1 is schedulable. This example shows that by considering the RC traffic when scheduling the TT frames, the impact of the TT schedule on the latency of the RC frames can be greatly reduced.

4 OPTIMIZATION STRATEGY

To solve the optimization problem presented in the previous section we propose a Redundant Architecture Selection (RAS) algorithm, based on Simulated Annealing (SA) metaheuristic (Reeves 1993). SA is searching for that solution which minimizes the objective function. Thus, each potential solution Υ visited is evaluated using the following function:

$$Objective(\Upsilon) = WP_\delta \cdot \delta + Cost(\Upsilon) \quad (3)$$

where the second term is the cost of the architecture, and the first term is a schedulability constraint.

δ is a "degree of schedulability" defined as sum, for all frames, of positive delays between the worst-case frame response times R_i and their deadlines $f_i.deadline$, $\delta = \sum \max(0, R_i - f_i.deadline)$. R_i is calculated as discussed in Section 3.2. When all the frames are schedulable, δ is zero, and the first term is ignored. However, when one or more frames are not schedulable, δ is a positive number that is multiplied with the penalty weight WP_δ, which is a large number. Thus, we allow the search to visit unschedulable solutions, but we penalize them in the hope of pushing the search towards schedulable implementation.

The SA algorithm is a variant of the neighborhood search technique, where the local search space is explored by moving from the current solution to a neighbor solution. The neighbor solution is determined using *design transformations*, or moves, which are applied to the current solution, see the next subsection. Each neighbor solution thus generated is evaluated using the objective function. Before evaluating a topology solution $\mathcal{G}$, we have to derive the corresponding network configuration. As mentioned, we have modified and extended the approach we have presented in (Tămaş-Selicean et al. 2014) for the configuration synthesis and we have improved the runtime of the RC schedulability analysis from (Tamas-Selicean et al. 2015). For example, we have used a List Scheduling heuristic to a quickly produce the TT schedule tables, instead of relying on design transformations.

In general, the new solution is accepted if it is an improved one. However, in the case of SA, a worse solution can also be accepted with a certain probability that depends on the deterioration of the objective function and on a control parameter called temperature, which is analog to the temperature concept of the physical annealing process.

4.1 *Design Transformations*

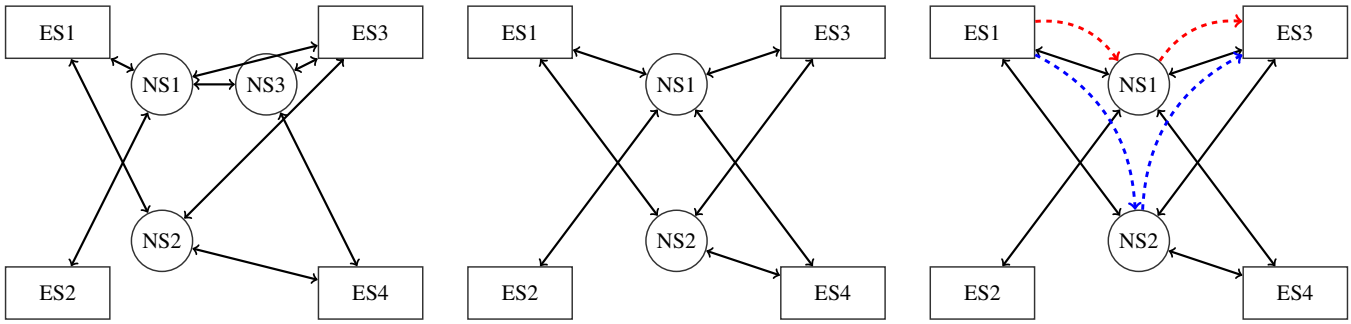
The design transformations are applied by RAS using the *Tweak* function presented in Algorithm 1. We have designed the moves such that the resulted neighbor solutions are valid. A valid network topology is one where there exists a path between any two ESes involved in a communication and the number of ports of NSes and ESes are not exceeded, and a valid routing is considered one in which there exists a number of disjoint channels that connect the source and destinations of each message such that its RL is satisfied.

Algorithm 1 Tweak(Υ)

```
1: if a random probability < P_RR then
2:     return Reroute(Υ)
3: else if a random probability < P_NS then
4:     return InDel_NS(Υ)
5: else
6:     return InDel_L(Υ)
7: end if
```

To present our moves, let us use the setup presented in Section 3.1. The initial solution from which SA starts is presented in Fig. 2(a). Frames f_1 and f_2 are redundant frames for m_1, and f_3 and f_4 are used for m_2 and m_3, respectively. The four frames are assigned to their own VLs; f_i to vl_i, with $i = 1..4$. Thus, the initial set of VLs is the following: vl_1=[[ES_1, NS_1], [NS_1, NS_3], [NS_3, ES_4]], vl_2=[[ES_1, NS_2], [NS_2, NS_4], [NS_4, ES_4]], vl_3=[[ES_1, NS_1], [NS_1, NS_3], [NS_3, ES_3]] and vl_4=[[ES_2, NS_1], [NS_1, NS_3], [NS_3, NS_4]]. Let us assume that the current solution is the one presented in Fig. 5(a) where f_2 is routed on vl_2=[[ES_1, NS_2], [NS_2, ES_4]], the other VLs routings remain unchanged.

The first group of moves is related to NSes, i.e., insertion and deletion of an NS ($InDel_{NS}$ in Algorithm 1). The insertion move implies that a new NS is created and added to the current solution. With a certain probability all old components are considered to be connected with the new inserted NS until at most half of its ports are occupied. After that, the frames routed over the components connected with the new NS are rerouted following the general rerouting strategy, the new candidate being considered for acceptance. For the deletion move, an NS is randomly picked and it and its direct connections are removed. Furthermore, if the new solution is invalid regarding the redundancy criteria, it will be "repaired". After the elements are deleted all VLs which involved these components are considered. For each



(a) Current solution (b) Solution after the NS_3 deletion (c) Solution after the NS_3 deletion and rerouting

Figure 5: Design transformation examples

interrupted path, an incidence-constrained backtracking is applied in order to find an alternative continuous path which is also disjoint regarding the other redundant paths. Starting with the source, and following the old path as furthest as possible, a partial path is created. Then, from the its end point, exploring its neighbors, another path to the destination is searched. If no alternative paths are found, a stepping back move is applied and the last physical link is removed from the partial path. The alternative paths are explored until one which satisfies the conditions of continuity and disjointedness is found or until the partial path become empty. If there are no such alternative paths from the set of NSes of the initial partial path, one with available ports is picked. A new connection between this NS and the destination is created such that the path becomes continuous. If the solution generated cannot be thus "repaired", it is rejected.

For example, if from the architecture presented in Fig 5(a) the NS_3 is deleted, after its deletion a physical link should be added to assure the RL of m_1, e.g., [NS_1, ES_4]. After the architecture is "repaired" the frames passing through deleted components are rerouted. Thus, applying the mentioned deletion move to the current solution, we get the topology in Fig. 5(b) and the routings: f_1 on vl_1=[[ES_1, NS_1], [NS_1, ES_4]], f_3 on vl_3=[[ES_1, NS_1], [NS_1, ES_3]] and f_4 on vl_4=[[ES_2, NS_1], [NS_1, ES_4]], the second frame being further assigned to vl_2. If a successive NS deletion is applied there will be no alternative paths for f_1 or f_2 and the new created solution is rejected.

Let us now present the *Reroute* move, which reroutes VLs. The *Reroute* move is applied to a virtual link vl_i carrying a frame f_i. This move returns a new tree for the virtual link vl_i, which has the same source and destinations, but goes through different dataflow links and network switches. The new tree is randomly selected, but we encourage the selection of routes which are less congested. Applying this move to the solution in Fig 5(b), we obtain the solution from Fig 5(c). Let us assume that the move randomly picks first to reroute the VL of f_1; it will find that it passes through the congested NS_1 (vl_1, vl_3 and vl_4 passing through). However, because another uncongested path which does not intersect the route of f_2 cannot be found, the first frame is not rerouted. If the moves picks f_2, which does not follow a congested path, it will be ignored. If f_3 is randomly considered by the *Reroute* move, an alternative uncongested path is found, namely [[ES_1, NS_2], [NS_2, NS_3]], thus the vl_3 will be rerouted over this path. In Fig.5(c), the routings of f_3 are depicted with dot-dash arrows; the old one with red and the newest one with blue.

The last set of design transformations used in RAS is the one related to physical links $InDel_L$. Similarly to the previous moves a physical link can be either inserted or deleted from the current solution. A new link is inserted such that it will join two randomly chosen components of which at most one can be an ES. After that, a rerouting strategy is applied. The deletion move will choose a connection such that the resulted solution is not invalid. For example, if the insertion move is firstly applied to the solution in Fig 5(b), a possible connection is the one between ES_2 and NS_2. Furthermore, considering that the solution is accepted and that the deletion is applied, a possible removal is of the link [ES_2, NS_1]. In this sequence of moves, the deletion is not rejected because the ES_2 is not disconnected. After such moves, f_4 is rerouted via vl_4=[[ES_2, NS_2], [NS_2, ES_4]], resulting the solution in Fig. 2(c).

5 EXPERIMENTAL EVALUATION

For the evaluation of our proposed optimization approach, "Redundant Architecture Selection" (RAS), we used a synthetic test case and a real-life case study, based on the Orion Crew Exploration Vehicle (CEV), 606E baseline (Paulitsch et al. 2011). The details of the two test cases are presented in Table 1, where we have listed the number of ESes and the number of messages in columns 2 and 3, respectively. We have considered a switch capacity of 12 ports. The algorithm was implemented in Java (JDK 1.6), running on Sun-Fire v440 computers with UltraSPARC IIIi CPUs at 1.062 GHz and 8 GB of RAM.

In the table we compare two algorithms, our optimization approach RAS and a "Straightforward Solution" (SS), which derives a fault-tolerant solution by adding the required redundancy, as follows. The con-

Name	ESes	RC msgs.	No. NSes		No. links		Running Time	$Cost(\Upsilon)$		Schedulable	
			SS	RAS	SS	RAS		SS	RAS	SS	RAS
Synthetic test case	8	20	6	5	58	49	8 min 30 s	700	590	no	yes
Orion CEV	30	30	24	19	232	86	9 h 25 min 80 s	2,800	1,240	no	yes

Table 1: Experimental results: comparison of SS and RAS

struction of SS starts with the set of messages $\mathcal{M}$ and the graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ having as vertices $\mathcal{V}$ the $\mathcal{ES}$ and an empty set $\mathcal{E}$ of edges. It assumes that, for all messages, the source ES, destination ES and the RL are known. The method creates an inverted index of redundancies and then fetches the messages mapped on each RL. For each message m_i, a set of disjoint paths, which is initially empty, is stored. Regarding the already constructed $\mathcal{G}$, the source and the destination of m_i and the stored set of paths, a new disjoint path is searched. If such a path is found, it is added to the set of disjoint paths and a map of the frame and of a VL consisting of this path is added to the system. Otherwise, we add to the network $\mathcal{G}$ a new NS and connect it in order to help the creation of the path regarding the previously mentioned conditions. The search for disjoint paths is repeated until the size of the set of disjoint paths is equal to RL_i. We consider that SS is a solution which can be obtained by a good engineer without the help of our optimization tool. In the RAS implementation we enforce the number of links going our from an ES to be the maximum over the *Redundancy Level* of messages sent from the respective ES. In addition, we do not consider the lack of disjoint redundant paths in a solution visited during search to be an invalid solution.

As we can see from the table, by carefully optimizing the introduction of redundancy using RAS, we are able to significantly reduce the cost compared to SS. For example, for the Orion test case, we have reduced the cost from 2,800 units to 1,240, using only 19 NSes and 86 physical links, compared to SS, which used 24 NSes and 232 physical links. In addition, because the optimization considers the schedulability of the frames, the solutions obtained by RAS are schedulable, compared to the SS solutions which are not schedulable (see the last two columns of the table). Although SS introduces a lot of redundant links and NSes, because it does not take into account the schedulability of messages, there are situations in which certain messages are delayed such that they miss their deadlines. The runtime of RAS is presented in column 8; the runtime of SS is under one second.

6 CONCLUSIONS

In this paper we have considered safety-critical real-time applications implemented using distributed architectures using the TTEthernet protocol. Our focus was on the synthesis of the network topology such that the real-time and safety properties of the applications are satisfied, and the cost of the architecture is minimized. We have proposed a Simulated Annealing-based optimization approach. The experimental results show that by using our optimization approach we are able to significantly reduce the cost of an architecture, obtaining architectures which are at the same time fault-tolerant and meet the deadlines of the messages.

REFERENCES

Annighoefer, B., C. Reif, & F. Thieleck (2014). Network topology optimization for distributed integrated modular avionics. In *Digital Avionics Systems Conference (DASC), 2014 IEEE/AIAA 33rd*, pp. 4A1–1. IEEE.

ARINC (2009). *ARINC 664P7: Aircraft Data Network, Part 7, Avionics Full-Duplex Switched Ethernet Network*. ARINC (Aeronautical Radio, Inc).

Cummings, R., K. Richter, R. Ernst, J. Diemer, & A. 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), 72–88.

Decotignie, J. D. (2005). Ethernet-based real-time and industrial communications. *Proceedings of the IEEE 93*(6), 1102–1117.

IEEE (2012). *IEEE 802.3 - IEEE Standard for Ethernet*. The Institute of Electrical and Electronics Engineers, Inc.

Kim, J.-H. & B.-J. Yum (1993). A heuristic method for solving redundancy optimization problems in complex systems. *Reliability, IEEE Transactions on 42*(4), 572–578.

Konak, A. & A. E. Smith (2006). Network reliability optimization. In *Handbook of Optimization in Telecommunications*, pp. 735–760. Springer.

Kuo, W. & V. R. Prasad (2000). An annotated overview of system-reliability optimization. *Reliability, IEEE Transactions on 49*(2), 176–187.

Paulitsch, M., E. Schmidt, B. Gstöttenbauer, C. Scherrer, & H. Kantz (2011). Time-triggered communication (industrial applications). In *Time-Triggered Communication*, pp. 121–152. CRC Press.

Reeves, C. R. (1993). *Modern heuristic techniques for combinatorial problems*. John Wiley & Sons, Inc.

Rushby, J. (2001). A comparison of bus architectures for safety-critical embedded systems. Technical report, Computer Science Laboratory, SRI International.

SAE (2011). *AS6802: Time-Triggered Ethernet*. SAE International.

Schneele, S. & F. Geyer (2012). Comparison of IEEE AVB and AFDX. In *Proceedings of the Digital Avionics Systems Conference*, pp. 7A1–1–7A1–9.

Tamas-Selicean, D. & P. Pop (2014). Optimization of ttethernet networks to support best-effort traffic. In *Proceedings of Emerging Technologies and Factory Automation*.

Tămaş-Selicean, D., P. Pop, & W. Steiner (2014). Design optimization of ttethernet-based distributed real-time systems. *Real-Time Systems 51*(1), 1–35.

Tamas-Selicean, D., P. Pop, & W. Steiner (2015). Timing analysis of rate constrained traffic for the ttethernet communication protocol. In *International Symposium On Real-time Computng (ISORC)*.