Real-Time Systems manuscript No.
(will be inserted by the editor)

Design Optimization of TTEthernet-based Distributed Real-Time Systems

Domiţian Tămaş–Selicean · **Paul Pop** ·
Wilfried Steiner

Received: date / Accepted: date

Abstract Many safety-critical real-time applications are implemented using distributed architectures, composed of heterogeneous processing elements (PEs) interconnected in a network. Our focus in this paper is on the TTEthernet protocol, a deterministic, synchronized and congestion-free network protocol based on the Ethernet standard and compliant with the ARINC 664 Specification Part 7. TTEthernet is highly suitable for safety-critical real-time applications since it offers separation for messages using the concept of virtual links and supports three time-criticality classes: Time-Triggered (TT), Rate-Constrained (RC) and Best-Effort (BE). In this paper we are interested in the design optimization of TTEthernet networks used to transmit real-time application messages. Given the set of TT and RC messages, and the topology of the network, our approach optimizes the packing of messages in frames, the assignment of frames to virtual links, the routing of virtual links and the TT static schedules, such that all frames are schedulable and the worst-case end-to-end delay of the RC messages is minimized. We propose a Tabu Search-based metaheuristic for this optimization problem. The proposed algorithm has been evaluated using several benchmarks.

Keywords TTEthernet · real-time · network protocols · scheduling · frame packing · routing

1 Introduction

Many safety-critical real-time applications, following physical, modularity or safety constraints, are implemented using distributed architectures, composed of heteroge-

D. Tămaş–Selicean · P. Pop
DTU Compute, Technical University of Denmark, Kongens Lyngby, Denmark
E-mail: dota@dtu.dk, paupo@dtu.dk

W. Steiner
TTTech Computertechnik AG, Vienna, Austria
E-mail: wilfried.steiner@tttech.com

neous 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 has high speeds (100 Mbps, 1 Gbps, and 10 Gbps), is known to be unsuitable for real-time and safety-critical applications (Decotignie, 2005; Lee et al, 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, such as FTT-Ethernet (Pedreiras et al, 2005), ARINC 664 Specification Part 7 (ARINC 664p7, for short) (ARINC, 2009), TTEthernet (SAE, 2011), EtherCAT (ETG, 2013) and IEEE Audio Video Bridging[1] (AVB). 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. The ARINC 664p7 specification (ARINC, 2009) 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 3). ARINC 664p7 provides predictable event-triggered communication suitable for hard real-time applications, and separation of safety-critical messages through the concept of virtual links. In addition to the functionality offered by Ethernet and ARINC 664p7, TTEthernet supports time-triggered communication based on static communication schedules which rely on a synchronized time base. Such time-triggered static scheduling approach is especially suitable for applications with highest criticality requirements in both temporal and safety domains.

TTEthernet supports applications with mixed-criticality requirements in the temporal domain, as it provides 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, and BE traffic has the lowest priority. TTEthernet is suitable for automotive (Steinbach et al, 2012), avionics (Suen et al, 2013) and space (Fletcher, 2009) applications.

In this paper we are interested in safety-critical real-time applications implemented using heterogeneous processing elements interconnected using TTEthernet. Furthermore, we assume the designer has partitioned the messages into TT, RC or BE traffic classes, depending on the particularities of the application. We are interested in optimizing the TTEthernet network implementation, such that the TT and RC messages are schedulable, and the end-to-end delay of RC messages is minimized.

[1] Audio Video Bridging is a collection of technical specifications IEEE (2011a, 2010, 2009, 2011b) that target synchronized communication with low jitter and low latency on Ethernet networks.

In (Tămaş-Selicean et al, 2012) we have proposed an approach which focuses only on deriving the static schedules for the TT messages. In this paper, synthesizing a network implementation means deciding on: (i) the routing of virtual links (VLs) on top of the physical network, (ii) the packing and fragmenting of messages into frames, (iii) the assignment of frames to VLs, (iv) the bandwidth of the RC VLs and (v) the static schedules of the TT messages. We have proposed a Tabu Search-based metaheuristic to solve this optimization problem.

The paper is organized as follows: Section 2 presents the related work. Section 3 and 4 introduce the architecture and application models. Section 5 describes the TTEthernet protocol in detail. Section 6 presents the problem formulation, while Section 7 consists of a few motivational examples. Section 8 and Section 9 describe and evaluate the proposed optimization.

2 Related work

There are two basic approaches for handling real-time applications (Kopetz, 2011). In the *Event-Triggered* (ET) approach, activities are initiated whenever a particular event is noted. In the *Time-Triggered* (TT) approach, activities are initiated at predetermined points in time. This duality is also reflected at the level of the communication infrastructure, where communication activities can be triggered either dynamically, in response to an event, as with the Controller Area Network (CAN) bus (ISO, 2003), or statically, at predetermined moments in time, as in the case of Time-Division Multiple Access (TDMA) protocols such as the Time-Triggered Protocol (TTP) (Kopetz, 2011). The trend is towards bus protocols that support both static and dynamic communication FlexRay (ISO, 2010), TTEthernet (SAE, 2011) and FTT-CAN (Almeida et al, 2002a).

There is a lot of work on bus scheduling and schedulability analysis (Dobrin and Fohler, 2001; Davis et al, 2007; Davis and Burns, 2009; Almeida et al, 2002b; Marau et al, 2010). The focus of this paper is on the optimization of the TTEthernet protocol. Researchers have proposed analysis and optimization approaches addressing, for example, TDMA bus such as the TTP (Pop et al, 1999) and a mixed TT/ET bus such as FlexRay (Pop et al, 2008), where the focus has been on optimizing the TDMA bus schedules, to decrease the end-to-end delays. For the CAN protocol, Burns and Davis (2013) propose two extensions that provide separation for mixed-criticality messages; they also propose a response-time analysis for these extensions. PROFINET Isochronous Real-Time (IRT) is another Ethernet-based communication protocol, which similarly to other protocols, splits the cycle time into a synchronous and an asynchronous part. Hanzalek et al (2010) propose an algorithm formulated as a Resource Constrained Project Scheduling with Temporal Constraints problem to obtain the schedule tables for the PROFINET IRT protocol. The proposed solution uses Integer Linear Programming (ILP) for smaller cases, and an iterative modulo scheduling-inspired heuristic for bigger topologies, and in both cases, they obtain similar results to the commercial tool for PROFINET IRT. For multi-mode distributed application connected by TDMA-based protocols, Azim et al (2014) propose a strategy to generate state-based schedules and show that such an approach reduces the mode-change delays compared to schedules generated using EDF.

For ARINC 664p7 systems, Nam et al (2013) have proposed a new real-time switching algorithm that guarantees an upper bound on the switching period. Having such an upper bound simplifies the worst-case delay analysis. For TTEthernet, Steiner (2010) proposes an approach for the synthesis of static TT schedules, where he ignored the RC traffic and used a Satisfiability Modulo Theory (SMT)-solver to find a solution which satisfies an imposed set of constraints. Steiner (2011) has proposed an SMT-solver approach to introduce periodic evenly-spaced slots into the static schedules to help reduce RC delays. Suethanuwong (2012) proposes a scheduling approach of the TT traffic, ignoring RC traffic, that introduces equally distributed available time slots for BE traffic. We have proposed a Tabu Search-based metaheuristic (Tămaş-Selicean et al, 2012) for the optimization of TT schedules that does not restrict the space inserted into the TT schedules to evenly-spaced periodic slots and is able to take into account the RC end-to-end delays during the design space exploration, and not only as a post-synthesis check.

Researchers have also addressed the issue of frame packing (Tanasa et al, 2011; Pop et al, 2005; Saket and Navet, 2006; Sandstrom et al, 2000). Schumacher et al (2008) and Wisniewski et al (2012) propose the use of a dynamic packing mechanism for PROFINET IRT systems to decrease the frame transfer time (by minimizing the frame overhead associated with each message), but also to reduce the number of conflicting situations where several frames compete for transmission (at the end station or network switch), by packing the competing messages into the same frame. Recent work has also addressed the frame packing for the ARINC 664p7 protocol (which TTEthernet extends with TT traffic). Ayed et al (2012) propose a packing strategy for multi-cluster networks, where the critical avionics subsystems are based on CAN buses, and are interconnected via ARINC 664p7. This strategy, meant to minimize the CAN bandwidth through the ARINC 664p7 network, performs packing at the CAN-ARINC 664p7 gateway based on a timer. Messages are not packed based on destinations, but on availability. As a consequence, the virtual link delivers the messages to all the possible destinations. Also for ARINC 664p7, Al Sheikh et al (2013) propose a packing strategy for messages with the same source and destinations, with the goal of minimizing the reserved bandwidth.

Routing in networks is a very well researched topic (Grammatikakis et al, 1998; Wang and Hou, 2000; Baransel et al, 1995). Researchers have also addressed routing in safety-critical systems (Herpel et al, 2009; Pedreiras and Almeida, 2004). For ARINC 664p7, Al Sheikh et al (2013) propose a load-balancing routing strategy. None of the existing work addresses packing/fragmenting and routing for TTEthernet.

3 Architecture model

A TTEthernet network is composed of a set of clusters. Each cluster consists of End Systems (ESes) interconnected by links and Network Switches (NSes). The links are full duplex, allowing thus communication in both directions, and the networks can be multi-hop. An example cluster is presented in Fig. 1, where we have 4 ESes, ES_1 to

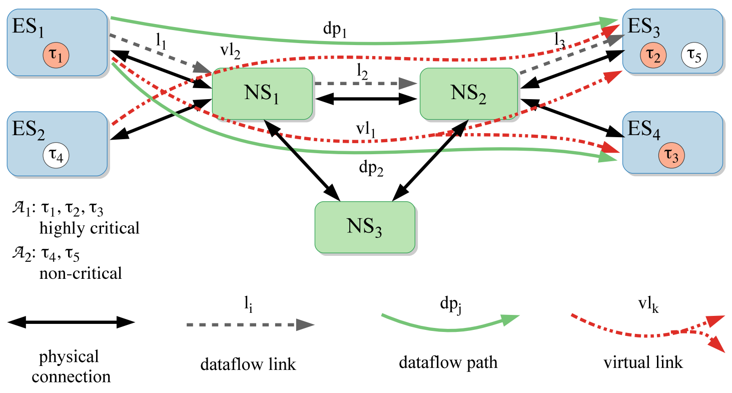


Fig. 1: TTEthernet cluster example

ES_4, and 3 NSes, NS_1 to NS_3. The optimization problem addressed in this paper is performed at the cluster-level. Each ES consists of a processing element containing a CPU, RAM and non-volatile memory, and a network interface card (NIC).

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.

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. A *dataflow path* $dp_i \in \mathcal{DP}$ is an ordered sequence of dataflow links connecting one sender to one receiver. 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). Moreover, dp_1 in Fig. 1 can be denoted as $[[ES_1, NS_1], [NS_1, NS_2], [NS_2, ES_3]]$.

The space partitioning between messages of different criticality transmitted over physical links and network switches is achieved through the concept of *virtual link*. Virtual links are defined by ARINC 664p7 (ARINC, 2009), which is implemented by the TTEthernet protocol, as a “logical unidirectional connection from one source end system to one or more destination end systems”.

Let us assume that in Fig. 1 we have two applications, $\mathcal{A}_1$ and $\mathcal{A}_2$. $\mathcal{A}_1$ is a high criticality application consisting of tasks τ_1 to τ_3 mapped on ES_1, ES_3 and ES_4, respectively. $\mathcal{A}_2$ is a non-critical application, with tasks τ_4 and τ_5 mapped on ES_2 and ES_3, respectively. τ_1 sends message m_1 to τ_2 and τ_3. Task τ_4 sends message m_2 to τ_5. With TTEthernet, a message has a single sender and may have multiple receivers. The flow of these messages will intersect in the physical links and switches. Virtual links are used to separate the highly critical message m_1 from the non-critical message m_2. Thus, m_1 is transmitted over virtual link vl_1, which is isolated from virtual link vl_2, on which m_2 is sent, through protocol-level temporal and spatial mechanisms (which are briefly presented in Section 5).

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. For example, 64 – 1518 B Ethernet frame length

7 B	1 B	14 B	28 B	17 - 1471 B	1 B	4 B	12 B
Preamble	Start Frame Delimiter	Ethernet Frame Header	ARINC 664p7 Frame Header	ARINC 664p7 Payload	SN	Frame Check Sqn	Inter Frame Gap

Fig. 2: Simplified frame format

vl_1, depicted in Fig. 1 using dot-dash red arrows, 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, $\mathcal{R}_{VL}(vl_1) = \{dp_1, dp_2\}$.

For a given message, with one sender and multiple receivers, there are several virtual links which can be used for transmission. For example, in Fig. 1, message m_2 from ES_2 to ES_3 can be transmitted via vl_2, containing dataflow path $dp_3 = [[ES_2, NS_1], [NS_1, NS_2], [NS_2, ES_3]]$, or via $vl_3 = \{dp_4\}$, with $dp_4 = [[ES_2, NS_1], [NS_1, NS_3], [NS_3, NS_2], [NS_2, ES_3]]$. Deciding each virtual link vl_i for a message m_i is a routing problem: we need to decide which route to take from a set of possible routes. This routing is determined by our optimization approach and, for real life systems, which can contain tens to hundreds of connected ESes and NSes, this is not a trivial problem.

4 Application model

In (Tămaş-Selicean and Pop, 2011) we have proposed an application model for mixed-criticality applications composed of interacting tasks, which communicate using messages. The proposed model can capture aspects specific to mixed-criticality applications, e.g., higher criticality tasks cannot receive messages from lower criticality tasks (because lower criticality tasks could send corrupted data) and additional task-level separation requirements, preventing certain tasks to share the same partition on an ES. In this paper we focus only on messages.

TTEthernet transmits data using *frames*. The TTEthernet frame format fully complies with the ARINC 664p7 specification (ARINC, 2009), and is presented in Fig. 2. A complete description of the ARINC 664p7 frame fields can be found in (ARINC, 2009). Messages are transmitted in the payload of frames. A bit pattern specified in the frame header identifies the traffic class of each frame (TT, RC or BE). The total frame header (Ethernet header and ARINC 664p7 header) is of 42 B, while the payload of each frame varies between a minimum of 17 B and a maximum of 1471 B. In case a frame carries data smaller than 17 B, the frame payload will be padded with zeroes to reach the minimum payload of 17 B. Thus, as shown in Fig. 2, the total protocol overhead for a frame (including the frame header, preamble, start frame delimiter and interframe gap) varies from 67 B, for data bigger than 17 B, to 83 B for data of 1 B.

The size $m_i.size$ for each message $m_i \in \mathcal{M}$ is given, where $\mathcal{M}$ is the set of all messages. As mentioned, TTEthernet supports three traffic classes: TT, RC and BE.

We assume that the designer has decided the traffic classes for each message. We define the sets $\mathcal{M}^{TT}$, $\mathcal{M}^{RC}$ and $\mathcal{M}^{BE}$, respectively, with $\mathcal{M} = \mathcal{M}^{TT} \cup \mathcal{M}^{RC} \cup \mathcal{M}^{BE}$. In addition, for the TT and RC messages we know their periods / rate and deadlines, $m_i.period$ or $m_i.rate$, and $m_i.deadline$, respectively. Furthermore, we also know the safety-criticality level (also called Safety Integrity Level, or SIL) for each message. RC messages are not necessarily periodic, but have a minimum inter-arrival time. We define the rate of an RC message m_i as $m_i.rate = 1/m_i.period$.

So far, researchers have assumed that each message $m_i \in \mathcal{M}$ is transmitted in the payload of a dedicated frame f_i. This was also the assumption of our earlier work, which focused only on the scheduling of TT frames (Tămaş-Selicean et al, 2012). However, the payload of a frame can in practice carry several messages. Moreover, messages can also be fragmented into several pieces, each carried by a different frame. In this paper, our optimization also determines the *fragmenting* and the *packing* of messages into frames at the source and destination nodes. Unlike in other protocols, e.g., PROFINET IRT, see (Schumacher et al, 2008), the NSes do not change the contents of the frames. Currently the only fragmentation/defragmentation supported by TTEthernet is a limited form of standard IP fragmentation/defragmentation. Hence, we assume that the fragmenting and packing of messages into frames is performed at the application level.

The *fragmenting* of messages into message fragments is denoted with $\Phi_m(m_i) = \{\forall m_j \in \mathcal{M}^+ | m_j \in m_i\}$, where $\mathcal{M}^+$ is the set that contains all the message fragments resulted from fragmenting, and the messages which were not fragmented. The message fragments $m_j \in \Phi_m(m_i)$ inherit the temporal constraints of the original message, and have equal sizes. For example, let us consider $\mathcal{M} = \{m_1, m_2\}$. In this case, $\mathcal{M}^+ = \mathcal{M}$. Message m_1 has a period and deadline of 30 ms, and a size of 300 B. Message m_2 has a deadline and period of 24 ms, and $m_2.size = 1200\ B$. We fragment m_2 into 3 same-sized message fragments, such that $\Phi_m(m_2) = \{m_3, m_4, m_5\}$ and m_3, m_4 and m_5 have the same period and deadline as m_2, but their size is 400 B. After fragmenting m_2, $\mathcal{M}^+ = \{m_1, m_3, m_4, m_5\}$.

The *packing* of messages and message fragments into frames is denoted with $\mathcal{P} : \mathcal{M}^+ \to \mathcal{F}$, $\mathcal{P}(m_j) = f_i$, where $\mathcal{F}$ is the set of all the frames in the cluster. In this paper, the frame is *the transmission unit in the cluster*. A periodic frame f_i has several frame instances $f_{i,j}$, where a frame instance $f_{i,j}$ is one instantiation of frame f_i, like a job is the instantiation of a task. Each frame is assigned to a virtual link, which specifies among others, the routing for the frame. In TTEthernet, each virtual link has assigned only one frame. The function $\mathcal{M}_F : \mathcal{F} \to \mathcal{VL}$, $\mathcal{M}_F(f_i) = vl_i$ captures this assignment of frames to virtual links. Let us consider the example given in Fig. 1, with message m_1 sent from ES_1 to ES_3 and ES_4. We assume that m_1 is packed by frame f_1, $\mathcal{P}(m_1) = f_1$. In this case, f_1 is assigned to vl_1, $\mathcal{M}_F(f_1) = vl_1$. Fig. 1 shows vl_1 routed along the shortest route.

The properties of the frames are derived based on what messages or message fragments are packed, such that the timing constraints are satisfied. Let us consider the prior example. The packing of message m_1 and message fragment m_3, where $m_1, m_3 \in \mathcal{M}^+$, in frame $f_1 \in \mathcal{F}$, is denoted with $\mathcal{P}(m_1) = f_1$ and $\mathcal{P}(m_3) = f_1$, respectively. In this case, the data packed by f_1 has a size of 700 B. Note that, unlike in the case of the EtherCAT (ETG, 2013) protocol, not all fragmenting and packing combinations are valid (e.g., messages packed into a frame must have the same source and destination ESes and must be of the same safety-criticality level). Also, the timing properties of the new frame depends on the timing constraints of messages. Our optimization takes care of these aspects, see Section 8 for details. Knowing the size of a frame f_j and the given speed of a dataflow link $[\nu_m, \nu_n]$, we can determine the transmission duration $C_j^{[\nu_m,\nu_n]}$ of f_j on $[\nu_m, \nu_n]$.

5 TTEthernet protocol

Next we will shortly describe the TTEthernet protocol. A more detailed presentation of the protocol can be found in (Tămaş-Selicean et al, 2012).

5.1 Time-Triggered Transmission

TT communication is done according to static communication schedules determined offline and stored into the ESes and NSes. The complete set of local schedules in a cluster are denoted with $\mathcal{S}$. The schedules $\mathcal{S}$ are derived by our optimization approach. There are several approaches to the synchronization of tasks (which could be TT or ET) and TT messages (Obermaisser, 2011). Often, TT tasks are used in conjunction with TT messages, and the task and message schedules are synchronized such that the task is scheduled to finish before the message is scheduled for transmission.

In addition, TTEthernet provides fault-tolerance services, such as fault-containment, to the application level. For example, if a task becomes faulty and sends more messages than scheduled (called a "babbling idiot" failure), the sending ES will protect the network as it will only transmit messages as specified in the schedule table S_S.

Besides the sending schedule tables, each NS also contains a receiving schedule table S_R. Thus, an NS will rely on the receive schedule S_R stored in the switch to check if a TT frame has arrived within a specified receiving window. This window is determined based on the sending times in the send schedules, the precision of the clock synchronization mechanism and the "integration policy" used for integrating the TT traffic with the RC and BE traffic (see next subsection for details). TT message frames arriving outside of this receiving window are considered faulty. In order to provide virtual link isolation and fault-containment, a TT receiver task TT_R will drop such faulty frames.

The schedules $\mathcal{S}$ contain the sending times and receiving windows for all the frames transmitted during an application cycle, T_{cycle}. A periodic frame f_i may occur in several instances (a *frame instance* is the equivalent of the periodic *job* of a task) within T_{cycle}. We denote the *x-th* instance of frame f_i with $f_{i,x}$. The sending time of a frame f_i relative to the start time of its period is called the *offset*, denoted with f_i*.offset*. In (Tămaş-Selicean et al, 2012) we have assumed a TTEthernet implementation where within an application cycle, the offset of a frame may vary from period to period. In this paper we consider a realistic implementation, where the sending time offset of a frame is identical in all periods, with the advantage of reducing the size needed to store the schedules.

5.2 Rate Constrained Transmission

RC traffic consists of event-triggered messages. The separation of RC traffic is enforced through "bandwidth allocation". Thus, for each virtual link vl_i carrying an RC frame f_i the designer decides the Bandwidth Allocation Gap (BAG). A BAG is the minimum time interval between two consecutive instances of an RC frame f_i and has a value of 2^i ms, i=0..7. The BAG is set in such a way to guarantee that there is enough bandwidth allocated for the transmission of a frame on a virtual link, with $BAG_i \leq 1/f_i.rate$. If the minimum inter-arrival time is greater than 128 ms, the BAG is set to 128 ms. The BAG is enforced by the sending ES. Thus, an ES will ensure that each BAG_i interval will contain at most one instance of f_i. Therefore, even if a frame is sent in bursts by a task, it will leave the ES within a specified BAG. Thus, the maximum bandwidth used by a virtual link vl_i transmiting an RC frame f_i is $BW(vl_i) = f_i.size/BAG(vl_i)$. The BAG for each RC frame is computed offline, based on the requirements of the messages it packs.

Fault-containment at the level of RC virtual links is provided by the NSes by implementing an algorithm known as *leaky bucket* (ARINC, 2009; SAE, 2011), which checks the time interval between two consecutive instances on the same virtual link. If this interval is shorter than the specified BAG time corrected by the maximum allowed transmission jitter, the frame instance is dropped. Thus, the NS prevents a faulty ES to send faulty RC frames (more often than allowed) and thus to disturb the network.

RC traffic also has to be integrated with TT traffic, which has higher priority. Thus, RC frames are transmitted only when there is no TT traffic on the dataflow link. With integration, contention situations can occur when a TT frame is scheduled for transmission, but an RC frame is already transmitting. There are three approaches in to handle such situations (SAE, 2011; Steiner et al, 2009): (i) shuffling, (ii) pre-emption and (iii) timely block. (i) With *shuffling*, the higher priority TT frame is delayed until the RC frame finishes the transmission. Thus, in the worst-case scenario, the TT frame will have to wait for the time needed to transmit the largest Ethernet frame, which is 1542 Bytes. In the case (ii) of *pre-emption*, the current transmission of the RC frame is aborted, and is restarted after the TT frame finished transmitting. In the case (iii) of *timely block*, the RC frame is blocked (postponed) from transmission on a dataflow link if a TT frame is scheduled to be sent before the RC frame would complete its transmission. Note that, as discussed in the previous subsection, the integration approaches have an impact on the receiving window of a TT frame, which has to account for the delays due to shuffling, for example.

6 Problem Formulation

The problem we are addressing in this paper can be formulated as follows: given (1) the topology $\mathcal{G}$ of the TTEthernet cluster, (2) the set of TT and RC messages $\mathcal{M}^{TT} \cup \mathcal{M}^{RC}$ and (3) for each message m_i the size, deadline, period / rate, SIL and the source and destination ESes, we are interested to determine an optimized implementation such that the TT and RC frames are schedulable. Determining an implementation means deciding on the (i) fragmenting Φ_m of messages and packing $\mathcal{P}$ of messages and messages fragments into frames, (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 $\mathcal{S}$.

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 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 (WCD), from the moment it is sent to the moment it is received. We denote this worst-case delay with R_{f_j}. We have presented in (Tămaş-Selicean et al, 2012) a schedulability analysis technique to determine the WCD of an RC frame. By comparing R_{f_j} with the deadline $f_j.deadline$, we can determine if an RC frame f_j is schedulable.

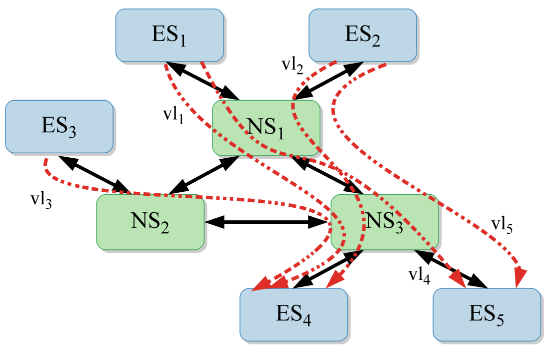
Once both TT and RC frames are schedulable several optimization objectives can be tackled. In this paper we are interested to optimize the design of the network such that the end-to-end delay of RC frames is minimized. Section 8.1 presents the cost function used for the optimization. In this paper we ignore the BE traffic, but we have shown in (Tămaş-Selicean, 2014) how BE traffic can be taken into account, by adding a quality-of-service measure for the BE traffic to the objective function. In this paper we are not concerned with scheduling redundant message delivery for fault-tolerance, since TTEthernet networks can be physically replicated. The schedules we derive for TT messages are used for all the replicated channels. The problem is illustrated in the next subsections using several motivational examples.

7 Straightforward Solution and Motivational Examples

7.1 Straightforward Solution

Let us illustrate the design optimization problem using the setup from Fig. 3, where we have a cluster composed of five end systems, ES_1 to ES_5 and three network switches NS_1 to NS_3 (see Fig. 3a) and an application with five TT messages, m_1 to m_5, and two RC messages, m_6 and m_7, see the table in Fig. 3b. The periods $m_i.period$ and deadlines $m_i.deadline$ of each message m_i are given in the table. For simplicity, in this example we assume that all messages have the same SIL. Although the standard TTEthernet speed is 100 Mbps or higher, for the sake of this example we consider a link speed of only 2 Mbps, and that all the dataflow links have the same speed. In Fig. 3b we also specify the source and destination for each message. For simplicity, we considered one destination for each message. The table also contains the transmission times C_i for each message m_i in our setup, considering for the moment that each message is packed into its own frame. We take into account the total overhead of the protocol for one frame (67 B for each frame). For this example we consider that the RC and TT traffic are integrated using a “timely block” policy (see Section 5), i.e., an RC frame will be delayed if it could block a scheduled TT frame.

A Straightforward Solution (SS) to our optimization problem is to (i) pack each message into its own frame and (ii) assign this frame to a virtual link, (iii) route each virtual link on the shortest paths from the frame source to its destinations, (iv) set



(a) Example architecture model

	period (ms)	deadline (ms)	size (B)	C_i (ms)	Source	Dest
$m_1 \in \mathcal{M}^{TT}$	40	40	233	1.2	ES_1	ES_4
$m_2 \in \mathcal{M}^{TT}$	40	40	683	3	ES_2	ES_4
$m_3 \in \mathcal{M}^{TT}$	10	10	433	2	ES_3	ES_4
$m_4 \in \mathcal{M}^{TT}$	40	40	1183	5	ES_1	ES_4
$m_5 \in \mathcal{M}^{TT}$	10	10	183	1	ES_2	ES_4
$m_6 \in \mathcal{M}^{RC}$	40	32	233	1.2	ES_1	ES_5
$m_7 \in \mathcal{M}^{RC}$	20	16	483	2.2	ES_2	ES_5

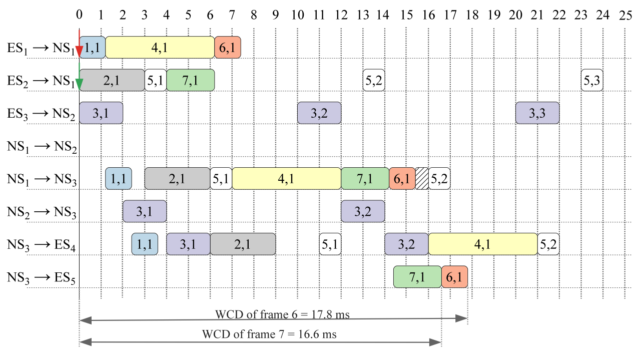
(b) Example application model

Fig. 3: Example system model

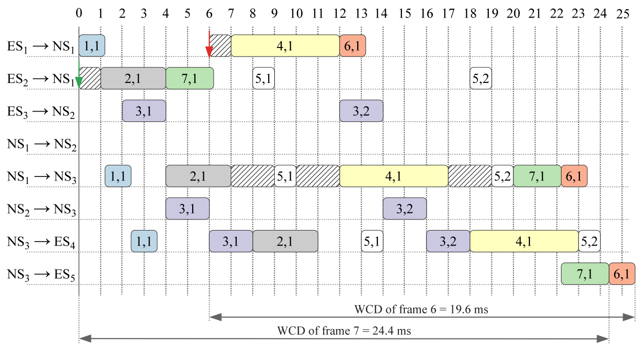
the bandwidth for each RC virtual link to the minimum required for the respective RC frame rate, and (v) schedule the TT frames using As-Soon-As-Possible (ASAP) scheduling. Such a straightforward solution would be chosen by a good designer without the help of our optimization tool. For the example in Fig. 3, this solution is depicted in Fig. 4a. Let us discuss it in more detail.

(i) *Fragmenting*, *packing*: SS does not fragment messages, and packs each message m_i into a frame f_i, with f_i inheriting the size, period and deadline of m_i. (ii) Frame *assignment* and (iii) VL *routing*. We assign each frame f_i to a virtual link vl_j and route the VL along the shortest path in the physical topology. The resulting VLs are depicted with dot dash red arrows in Fig. 3a. (iv) Each RC VL carrying an RC frame has an associated *bandwidth* parameter called BAG. BAG(vl_j) is the minimum time interval between two consecutive instances of an RC frame f_i on VL vl_j. SS will set the BAG in such a way to guarantee the rate of the frame f_i, while respecting the protocol constraints on BAG sizes (see Section 5.2).

(v) *Scheduling* of TT frames. As mentioned, SS uses ASAP scheduling to derive the TT frame schedules. Fig. 4a presents these schedules for our example. Instead of presenting the actual schedule tables, we show a Gantt chart, which shows on a timeline from 0 to 25 ms what happens on the eight dataflow links of interest, $[ES_1, NS_1]$, $[ES_2, NS_1]$, $[ES_3, NS_2]$, $[NS_1, NS_2]$, $[NS_1, NS_3]$, $[NS_2, NS_3]$, $[NS_3, ES_4]$ and $[NS_3, ES_5]$. For the TT frames f_1 through f_5, the Gantt chart captures their sending times (the left edge of the rectangle) and transmission duration (the length of the rectangle). In the Gantt chart, for readability, the rectangles associated to each frame



(a) Straighforward Solution, each message assigned to one frame, routed along the shortest path, and scheduled ASAP, results in f_7 missing its deadline in the worst-case scenario



(b) Alternative baseline solution, using SS approach for message packing and frame routing, but a different TT schedule. In this case f_7 misses its deadline in the worst-case scenario

Fig. 4: Baseline solutions

instance $f_{i,j}$ are labelled only with i,j. We can see in Fig. 4a that all the TT frames are schedulable (they are received before their deadlines).

Since the transmission of RC frames is not synchronized with the TT frames, there are many scenarios that can be depicted for the RC frames f_6 and f_7, depending on when the frames are sent in relation to the schedule tables. Because we are interested in the schedulability of the RC frames f_6 and f_7, we show in the Gantt charts their worst-case scenario, i.e., the situation which has generated the largest (worst-case) end-to-end delay for these frames. Thus, in Fig. 4a, the worst-case end-to-end delay (WCD) of the RC frame f_6, namely $f_{6,1}$, is 17.8 ms, smaller than its deadline of 32 ms, and hence, it is schedulable. For f_7 though, the WCD is 16.6 ms, larger than its deadline of 16 ms, thus frame f_7 is not schedulable. This worst-case for f_7 happens for the frame instance $f_{7,1}$, see Fig. 4a, when $f_{7,1}$ is ready for transmission by ES_2 at 0 ms, depicted with a downward pointing green arrow. The worst-case arrival time for f_6, which leads to the largest WCD R_{f_6}, is depicted with a downward pointing red arrow. In this case, as the network implements the *timely block* integration algorithm, the frame f_7 cannot be sent if its transmission interferes with the TT schedule. Thus,

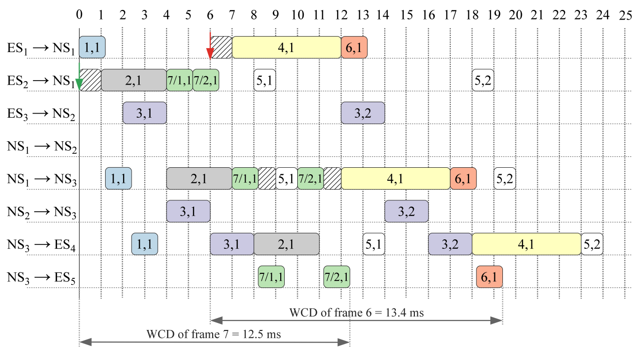


Fig. 5: Fragmenting RC message m_7 into two frames, reduces the WCD to 12.5 ms, below its deadline.

$f_{7,1}$ cannot be sent by ES_2 until the TT frames $f_{2,1}$ and $f_{5,1}$ finish transmitting and it cannot be forwarded by NS_1 to NS_3 until $f_{4,1}$ is completely relayed by NS_1.

Let us illustrate the optimizations that can be performed to reduce the WCD of RC frames, and thus make frame f_7 schedulable. In order to show all the optimizations that can be performed, we propose to use Fig. 4b as the alternative initial solution. The solution presented in Fig. 4b is built using the SS approach of packing messages into frames and routing the frames, but has an alternative schedule table. In this case, the TT frames are schedulable, and the WCD for the RC frames are 19.6 ms for f_6, and 24.4 ms for f_7. Thus f_7 misses its deadline, leading to an unschedulable solution. As the network implements the *timely block* integration algorithm, the frame $f_{7,1}$ cannot be sent 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 on dataflow link $[NS_1, NS_3]$ starts only at time 20 ms, right after $f_{5,2}$ is received by NS_3, which is too late to meet f_7's deadline.

7.2 Message Fragmenting and Packing

Next we will discuss about the benefits of fragmenting and packing messages to frames, and we will show how they can improve the schedulability of messages.

Let us perform the following modification to the solution from Fig. 4b, shown in Fig. 5. We *fragment* the RC message m_7 into two frames $f_{7/1}$ and $f_{7/2}$. The sum of the frames packing the message fragments is larger than the frame packing the message due to frame overheads. Thus, we can use the existing empty time slots between the TT frames on dataflow link $[NS_1, NS_3]$. The new RC frames, with a $C_{7/1,1} = C_{7/2,1} = 1.25$ ms and a BAG of 16 ms, can be transmitted in the available time between $f_{2,1}$–$f_{5,1}$, and $f_{5,1}$–$f_{4,1}$, respectively. This solution reduces the WCD for message m_6 to 13.4 ms and for m_7 to 12.5 ms, thus making all the RC messages schedulable. Fragmenting RC messages allows the RC frames to better use the existing available time slots between the TT frames. On the other hand, fragmenting TT messages can increase the porosity of the schedule. The porosity of a schedule

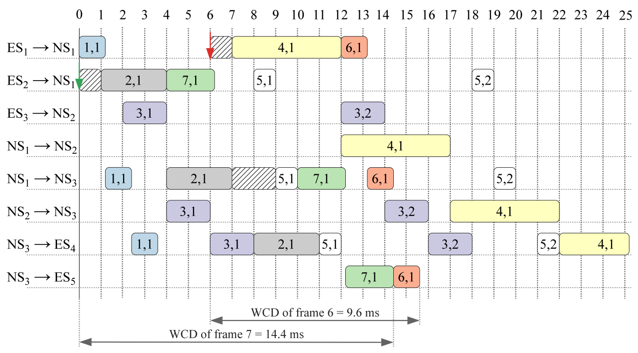


Fig. 6: Rerouting TT frame f_4 via NS_2 frees up traffic on dataflow link $[NS_1, NS_3]$, reducing the WCD of the RC messages, compared to Fig. 4b

is a measure of the size and distribution of idle gaps the schedule of time-triggered frames provides for non time-triggered frames.

Packing is especially advantageous to small messages, as it reduces the ratio of protocol overhead to frame payload. Consider 3 RC messages m_{RC1}, m_{RC2} and m_{RC3}, with the same SIL, transmitted from the same source to the same destinations. Messages have a size of 18 B, 10 B and 21 B, respectively, and a deadline of 20 ms, 19 ms and 50 ms. If each message is packed in its own frame, the corresponding frames would have a size of 85 B, 81 B and 88 B, respectively, with a BAG of 16, 16 and 32 ms, respectively. If we pack the three messages into one frame, the new frame would have a size of 116 B with a BAG of 16 ms. Thus, with an increase in the frame size of less than 50% compared to the smallest frame, we can deliver all three messages at once, reducing also the delivery time of the frames. Moreover, the benefits of packing several TT messages into one frame has the advantage of consolidating the available time intervals for RC transmission, between scheduled TT frames, into bigger chunks.

7.3 Virtual Link Routing

Fig. 3a shows the routing of VLs as performed by the Straightforward Solution, which selects the shortest route. Let us assume, however, that we route the TT frame f_4 (from ES_1 to ES_4) via the longer route through NS_2 ($[[ES_1, NS_1], [NS_1, NS_2], [NS_2, NS_3], [NS_3, ES_4]]$), instead of the shortest route ($[[ES_1, NS_1], [NS_1, NS_3], [NS_3, ES_4]]$). Thus, in Fig. 6 we can see we have a WCD of 9.6 ms and 14.4 ms for RC frames f_6 and f_7, respectively, which are schedulable.

This example shows that by selecting, counterintuitively, a longer route for a message, we can improve the schedulability.

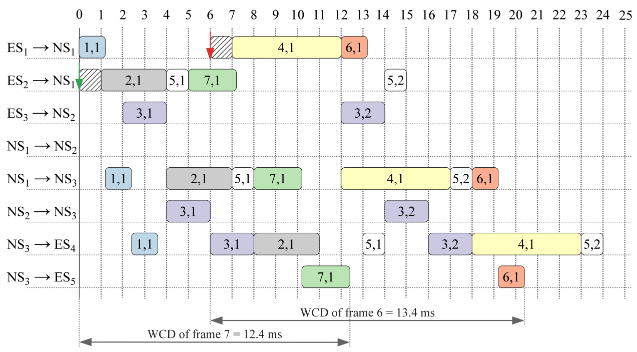


Fig. 7: Rescheduling frame f_5 to an earlier instant on [ES_2, NS_1] groups the TT frames and eliminates the timely block intervals, resulting in the WCD of the RC messages

7.4 Scheduling of TT messages

In (Tămaş-Selicean et al, 2012) we have shown how carefully deciding the schedules for the TT messages can improve schedulability. Compared to (Tămaş-Selicean et al, 2012), which has focused only on scheduling, in this paper we also address fragmenting, packing and routing. In addition, we also consider realistic scheduling constraints imposed by the current TTEthernet implementations. In (Tămaş-Selicean et al, 2012) we have assumed that the offset of a TT frame instance on a dataflow link can vary across periods. Thus, frame instances of the same TT frame may have different offsets. However, this is not supported by the current TTEthernet implementations, and hence in this paper we impose the scheduling constraint that all the frame instances of a TT frame on a dataflow link should have the same offset in all periods.

Fig. 7 presents the impact of rescheduling a TT frame, in the context of the example in Fig. 4b. We reschedule the TT frame f_5 for an earlier transmission on [ES_2, NS_1]. Although this move increases the worst-case delay for f_7 on that dataflow link, the move groups the TT frames together on the dataflow link [NS_1, NS_3]. Consequently, this move eliminates the timely blocked intervals that block the transmission of RC frames, thus reducing the overall WCD for both RC frames.

8 Design optimization

The scheduling problem presented in Section 6 is similar to the flow-shop scheduling problem and is shown to be NP-complete (Garey and Johnson, 1979), with the packing and fragmenting of frames adding to the complexity of the problem. In order to solve this problem, we propose the "Design Optimization of TTEthernet-based Systems" (DOTTS) strategy from Fig. 8, which is based on a Tabu Search metaheuristic.

Our strategy has 2 steps, see the two boxes in Fig. 8: (1) In the first step we determine an initial solution using the straightforward approach introduced in Section 7.1. The initial set $\mathcal{B}^\circ$ of BAGs for each RC VL is set as explained in Section 7.1.

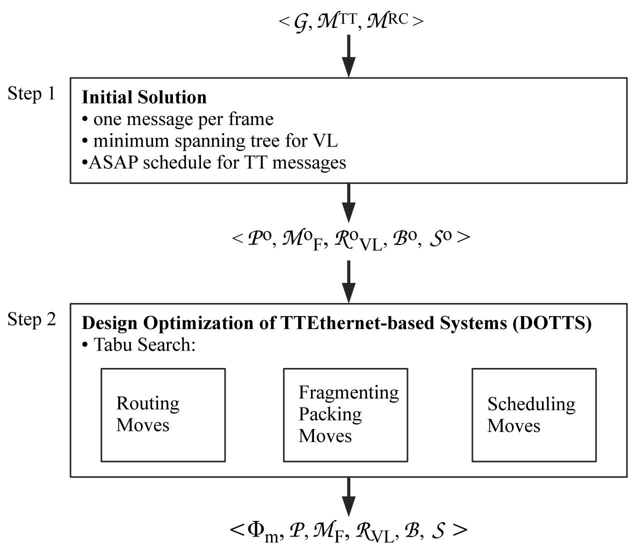

$< \mathcal{G}, \mathcal{M}^{TT}, \mathcal{M}^{RC} >$
Step 1
Initial Solution
• one message per frame
• minimum spanning tree for VL
•ASAP schedule for TT messages
$< \mathcal{P}^o, \mathcal{M}^o_F, \mathcal{R}^o_{VL}, \mathcal{B}^o, \mathcal{S}^o >$
Step 2
Design Optimization of TTEthernet-based Systems (DOTTS)
• Tabu Search:
Routing Moves
Fragmenting Packing Moves
Scheduling Moves
$<\Phi_m, \mathcal{P}, \mathcal{M}_F, \mathcal{R}_{VL}, \mathcal{B}, \mathcal{S} >$

Fig. 8: Design Optimization of TTEthernet-based Systems

The initial routing of virtual links $\mathcal{R}^\circ_{VL}$ is done to minimize the paths. We use Prim's algorithm (Cormen et al, 2009) for minimum spanning tree to determine the initial vl°_i for each frame f_i. We call Prim's algorithm for each frame f_i. Let ES^{src}_i be the source of frame f_i and $\mathcal{ES}^{dest}_i$ be the set of destinations of frame f_i. The input to Prim's algorithm is the topology graph $\mathcal{G}$ from which we have removed all the ESes, except $ES^{src}_i \cup \mathcal{ES}^{dest}_i$. That is, we are interested in the minimum spanning tree in the graph that connects the ESes involved in a particular frame's transmission. Then we remove from the resulting spanning tree all NSes that are not on the path between the ESes, obtaining thus the routing that minimizes the paths. For frame f_1 packing m_1 in Fig. 1, the graph is composed of vertices $\{ES_1, NS_1, NS_2, ES_3, ES_4\}$ and the interconnecting edges. The virtual link routing for frame f_1 is the minimum spanning tree in this graph, depicted with a red dash-dotted arrow, see Fig. 1.

The initial schedules $\mathcal{S}^\circ$ for the TT messages are built using the ASAP scheduling, where the ESes, NSes and dataflow links are considered the resources onto which the frame instances have to execute. The initial routing for the example in Fig. 3 is presented in Fig. 3a with red dash-dotted arrow, and the initial schedule results from the one-to-one packing and ASAP scheduling depicted in Fig. 4a.

(2) In the second step, we use a Tabu Search meta-heuristic (see Section 8.1) to determine the fragmenting Φ_m and packing $\mathcal{P}$ of messages in frames, the final set of virtual links $\mathcal{VL}$, the assignment of frames to virtual links $\mathcal{M}_F$, the routing of virtual links $\mathcal{R}_{VL}$, the BAGs for the RC VLs and the TT schedules $\mathcal{S}$, such that the TT and RC frames are schedulable, and the end-to-end delay of RC frames is minimized.

8.1 Tabu Search

Tabu Search (TS) (Glover and Laguna, 1997) is a meta-heuristic optimization, which searches for that solution that minimizes the *cost function*. Tabu Search explores the design space by using design transformations (or "moves") applied to the current solution in order to generate neighboring solutions. In order to increase the efficiency of the Tabu Search, and to drive it intelligently towards the solution, these "moves" are not performed randomly, but chosen to improve the search. If the currently explored solution is better than the best known solution, it is saved as the "best-so-far" *Best* solution. To escape local minima, TS incorporates an adaptive memory (called "tabu list"), to prevent the search from revisiting previous solutions. Thus, moves that improve the search are saved as "Tabu". In case there is no improvement in finding a better solution for a number of iterations, TS uses *diversification*, i.e., TS visits previously unexplored regions of the search space. The TS algorithm runs while the termination condition is not reached. This termination condition can be, for example, a time limit, a certain number of iterations or a number of iterations without improvement, considering the cost function (Gendreau, 2002).

Next we describe our implementation. Fig. 9 presents our Tabu Search algorithm, which takes as input the topology of the network $\mathcal{G}$, the set of TT and RC messages $\mathcal{M}^{TT} \cup \mathcal{M}^{RC}$ (including the size, period/rate and deadline), and returns at the output the best configuration of (i) message fragmenting Φ_m and packing $\mathcal{P}$, (ii) the assignment of frames to virtual links $\mathcal{M}_F$, (iii) the routing of virtual links $\mathcal{R}_{VL}$, the (iv) the bandwidth for each RC virtual link and (v) the TT schedules $\mathcal{S}$ found during the design space exploration, in terms of the cost function. We define the cost function of an implementation as:

$$Cost = w_{TT} \cdot \delta_{TT} + w_{RC} \cdot \delta_{RC} \tag{1}$$

where δ_{TT} is the "degree of schedulability" for the TT frames and δ_{RC} is the "degree of schedulability" for the RC frames. These are summed together into a single value using the weights w_{TT} and w_{RC}, given by the designer. These weights change their values depending on the schedulability of the frames: in case a frame is not schedulable, its corresponding weight is a very big number, i.e., a "penalty" value. Such a penalty value has to be several orders of magnitudes larger than the weights used, see Section 9 for details. This allows us to explore unfeasible solutions (which correspond to unschedulable frames) in the hope of driving the search towards a feasible region. Once the TT frames are schedulable we set the weight w_{TT} to zero, to drive the search towards solutions that minimize the end-to-end delays for the RC frames. Setting the w_{TT} weight to zero does not "freeze" the TT schedules (TT related moves will still be performed), but allows the cost function to evaluate a solution by considering the RC traffic.

The degree of schedulability is calculated as:

$$\delta_{TT/RC} = \begin{cases} c_1 = \sum_i \max(0, R_{f_i} - f_i.deadline) & \text{if } \exists \text{ j } R_{f_j} \geq f_j.deadline \\ c_2 = \sum_i (R_{f_i} - f_i.deadline) & \text{if } c_1 = 0 \end{cases} \tag{2}$$

If at least one frame is not schedulable, there exists one R_{f_i}, i.e., the worst-case end-to-end delay (WCD) of f_i, greater than the deadline $f_i.deadline$, and therefore the term

```
**TabuSearch**($\mathcal{G}$, $\mathcal{M}^{TT} \cup \mathcal{M}^{RC}$, $\mathcal{P}^\circ$, $\mathcal{M}_F^\circ$, $\mathcal{R}_{VL}^\circ$, $\mathcal{B}^\circ$, $\mathcal{S}^\circ$)
Best ← Current ← < $\mathcal{P}^\circ$, $\mathcal{M}_F^\circ$, $\mathcal{R}_{VL}^\circ$, $\mathcal{B}^\circ$, $\mathcal{S}^\circ$ >
L ← {}
while termination condition not reached do
    remove tabu with the oldest tenure from L if Size(L) = maxLSize
    // generate a subset of neighbors of the current solution
    $\mathcal{C}$ ← CLG(Current, $\mathcal{G}$, $\mathcal{M}^{TT} \cup \mathcal{M}^{RC}$)
    Next ← solution from $\mathcal{C}$ that minimizes the cost function
    if Cost(Next) < Cost(Best) then
        // accept Next as Current solution if better than the best-so-far Best
        Best ← Current ← Next
        add tabu(Next) to L
    else if Cost(Next) < Cost(Current) and tabu(Next) ∉ L then
        // also accept Next as Current solution if better than Current and not tabu
        Current ← Next
        add tabu(Next) to L
    end if
    if diversification needed then
        Current ← Diversify(Current)
        empty L
    end if
end while
return < $\Phi_m$, $\mathcal{P}$, $\mathcal{M}_F$, $\mathcal{R}_{VL}$, $\mathcal{B}$, $\mathcal{S}$ >
```

Fig. 9: The Tabu Search algorithm

c_1 will be positive. We have discussed in (Tămaş-Selicean et al, 2012) how the WCD of a frame is calculated. The assumption in that analysis is that all the RC frames have the same priority. In our future work we intend to extend the analysis to allow different priority levels for RC frames. However if all the frames are schedulable, this means that each R_{f_i} is smaller than $f_i.deadline$, and the term $c_1 = 0$. In this case, we use c_2 as the degree of schedulability, since it can distinguish between two schedulable solutions.

Line 1 initializes the *Current* and *Best* solutions to the initial solution formed by the tuple $< \mathcal{P}^\circ, \mathcal{M}_F^\circ, \mathcal{R}_{VL}^\circ, \mathcal{B}^\circ, \mathcal{S}^\circ >$. Line 2 initializes the tabu list L to an empty list. The size of the tabu list (maxLSize), i.e., its *tenure*, is set by the user. The algorithm runs while the termination condition is not reached (see line 3), which in our implementation stops the search after a predetermined amount of time set by the user. In case the tabu list L is filled, we remove the oldest tabu from this list (see line 4).

Evaluating all the neighboring solutions is infeasible, therefore we generate a subset of neighbors of the *Current* solution (line 6), called *Candidate List*, by running the *Candidate List Generation* (CLG) algorithm (see Section 8.3), and the algorithm chooses from this *Candidate List*, as the *Next* solution, the one that minimizes the cost function (line 7).

In case this *Next* solution is better than the best-so-far *Best* solution (lines 8–11), TS sets the *Best* and *Current* solutions as the *Next* solution. TS accepts a solution (i.e., sets it as *Current*) generated by a tabu move only if it is better than the best known solution *Best*. Accepting a solution generated by a tabu move is referred to as "aspiration criteria". Then, the TS algorithm adds the move that generated this solution to the tabu list, to prevent cycling. If the move is already a tabu, it will be added to the head of the list, thus setting its tenure to the size of the list. If *Next* improves the cost function compared to the *Current* solution, but not to the *Best*, and furthermore, the move that generated *Next* is not a tabu, TS sets the *Current* solution as *Next*, and adds the move to the tabu list.

In case the TS algorithm does not manage to improve the current solution after a number of iterations (lines 17–20), TS proceeds to a diversification stage (line 18). During this stage, TS attempts to drive the search towards an unexplored region of the design space. To achieve this, the algorithm generates a new solution by applying all the moves that generated the neighboring solutions in the current candidate list. In case the candidate list contains solutions obtained by applying contradictory moves to the same message, only the first move will be executed. For example, let us consider that the candidate list contains the following two solutions: (i) one obtained by *packing* m_i, m_j and m_k and (ii) another solution obtained by *fragmenting* m_i. In this case, during the diversification stage only the *packing* move will be taken into account, being the first move applied to m_i. After such a diversification stage, the tabu list L is emptied.

8.2 Design Transformations

We use three classes of moves in our Tabu Search: (1) *routing* moves applied to virtual links, (2) *fragmenting/packing* moves applied to messages and (3) *scheduling* moves applied to the TT frames.

(1) 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 the *reroute* move can also have a parameter specifying a dataflow link l_i to avoid in the new tree, because, for example, we have determined that l_i is too congested.

The reroute move selects from the complete set of trees that can be used to route a virtual link vl_i. This set is determined only once, before TS is run, for every message m_i. We use breadth-first search to find every path between the source of m_i and it's destinations, and we combine these paths to obtain a complete set of unique trees.

(2) The *fragmenting / packing* moves change the structure of the extended messages set $\mathcal{M}^+$ and the assignment of messages to frames $\mathcal{P}$. There are two types of *fragmenting moves*: *fragment message* and *un-fragment message*, and two types of *packing moves*: *pack messages* and *unpack frames*. The *fragment message* move splits a message m_i into several same-sized message fragments $m_j \in \mathcal{M}^+, m_j \in \Phi_m(m_i)$. Each message fragment inherits the period and deadline of the message m_i. In case of the RC messages, each vl_j carrying the RC frame f_j that is packing one message fragment m_j, will inherit the BAG of vl_i carrying m_i. The *un-fragment message* undoes the *fragment message* move, and regroups all the fragments $m_j \in \Phi_m(m_i)$ back into the original message m_i.

The *pack messages* move packs into the same frame several messages and/or message fragments that (i) have the same source and destinations, (ii) belong to the same traffic class, (iii) have the same SIL and (iv) that the sum of their size does not

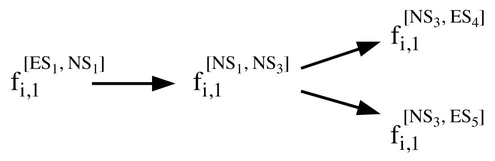


Fig. 10: Representation of a frame as a tree

exceed the maximum allowed payload size of 1471 B. In case we pack messages with different periods and deadlines, the new frame f_i will inherit the tightest deadline and the smallest period of the composing messages and fragments. For RC messages, the new frame f_i will inherit the smallest BAG of the composing messages.

Packing of message fragments from different frames can further reduce the WCD of the messages involved, similarly to the example of packing RC frames given in Section 7.2. Although packing message fragments of different messages is possible, we do not consider this to be realistic, hence we do not employ this in our optimization. Also note that the ARINC 664p7 protocol has a restriction of 4096 VLs per cluster. The *pack messages* move can be used to circumvent this restriction, in case there are more than 4096 messages to be sent.

The *unpack* move applied to frame f_i assigns each $m_j \in \mathcal{M}^+, \mathcal{P}(m_j) = f_i$, to a new frame f_j, on a one-to-one basis.

For example, let us consider $\mathcal{M} = \{m_1, m_2, m_3\}$. By fragmenting m_1 into 3 fragments, we obtain $\Phi_m(m_1) = \{m_{1/1}, m_{1/2}, m_{1/3}\}$, with the periods and deadlines equal to m_1, and their size equal to $\lceil m_1.size/3 \rceil$. Similarly, fragmenting m_3 into 2 fragments, we get $\Phi_m(m_3) = \{m_{3/1}, m_{3/2}\}$. Thus, $\mathcal{M}^+ = \{m_{1/1}, m_{1/2}, m_{1/3}, m_2, m_{3/1}, m_{3/2}\}$. Performing the *un-fragment* move on m_3 will result in $\mathcal{M}^+ = \{m_{1/1}, m_{1/2}, m_{1/3}, m_2, m_3\}$. If we pack $m_{1/1}$ and m_2 into frame f_x, such that $\mathcal{P}(m_{1/1}) = f_x$ and $\mathcal{P}(m_2) = f_x$, $f_x.deadline$ is determined as $min(m_{1/1}.deadline, m_2.deadline)$ and for TT messages, $f_x.period{=}min(m_{1/1}.period,\ m_2.period)$, while for RC messages, $f_x.rate$ is equal to $min(m_{1/1}.rate,\ m_2.rate)$.

(3) Let us now discuss the *scheduling* moves. A periodic frame f_i has several frame instances. For the *scheduling* moves we introduce the following notation: we denote with $f_{i,x}^{[\nu_j,\nu_k]}$ the instance of frame $f_{i,x}$ sent on the dataflow link $[\nu_j, \nu_k]$. All the frame instances $f_{i,x}^{[\nu_j,\nu_k]}$ of frame f_i have the same offset across all periods. Let us consider the topology presented in Fig. 3a, and frame f_i transmitted from ES_1 to ES_4 and ES_5 along the shortest route, that is [[ES_1, NS_1], [NS_1, NS_3], [NS_3, ES_4], [NS_3, ES_5]]. The tree model that represents the frame f_i is shown in Fig. 10. Each frame f_i is assigned a virtual link vl_i. A virtual link is a tree structure, where the sender is the root and the receivers are the leafs. In the case of a virtual link, the ESes and NSes are the nodes, and the dataflow links are the edges of the tree. However, in our tree model of a frame, the dataflow links are the nodes and the edges are the precedence constraints. Naturally, frame instance $f_{i,1}$ on dataflow link $[NS_3, ES_5]$ cannot be sent before it is transmitted on $[NS_1, NS_3]$ and received in NS_3. Such a precedence constraint is captured in the model using an edge, e.g., $f_{i,1}^{[NS_1,NS_3]} \rightarrow f_{i,1}^{[NS_3,ES_5]}$. We denote with

$pred(f_{i,x}^{[\nu_j,\nu_k]})$ the set of predecessor frame instances of the frame instance $f_{i,x}$ on dataflow link $[\nu_j,\nu_k]$.

We propose 4 *scheduling* moves: *advance*, *advance predecessors*, *postpone* and *postpone predecessors*. The *advance* move will advance the scheduled send time offset of a TT frame f_i from node ν_j on a dataflow link $[\nu_j,\nu_k]$ to an earlier moment in time. The *advance predecessors* applied to a frame f_i will advance the scheduled send time offset for all its predecessors. Similarly, the *postpone* move will postpone the schedule send time offset of a TT frame from a node, while *postpone predecessors* will postpone the send time offset for one random predecessor of that frame.

The schedule offset of a frame is advanced or postponed by a random value that is upper bounded. This maximum amount of time a frame instance is advanced or postponed at a node $\nu_j \in \mathcal{V}$ is computed such that the frame instance will not be sent before it is scheduled to be received, or sent too late to meet its deadline. For each node ν_j, we compute the latest absolute send time for frame f_i so that it may still meet its deadline, ignoring other traffic. Also, after each move we may need to adjust the schedules (move other frame offsets later or earlier) to keep the solution valid, i.e., the schedules respect the precedence and resource constraints.

Tabu Search relies on a memory structure called "Tabu List" to prevent the search from cycling through previously visited solutions. Our algorithm relies on a *tabu list* with tabu-active attributes, that is, it does not remember whole solutions, but rather attributes of the moves that generated the tabu solutions. For each tabu, we record the move that generated it, and the affected frames or messages.

8.3 Candidate List

As previously mentioned, Tabu Search drives the search towards schedulable solutions by applying "moves" to the current solution in order to generate neighboring solutions. The number of neighbors for each solution is very large, therefore evaluating all the neighboring solutions is infeasible. Instead, our algorithm evaluates only a subset of neighbors of the *Current* solution, called *Candidate List*. One option is to randomly select the neighbors placed on the candidate list. However, our algorithm uses a heuristic approach that selects those neighbors which have a higher chance to quickly lead to a good result. The Candidate List Generation (CLG) algorithm is described in the following. Each candidate solution is obtained by performing moves on the *Current* solution.

We consider the following classes of candidates: (1) candidates for TT frames, (2) candidates for RC frames and (3) randomly generated candidates.

8.3.1 Candidates for TT frames

CLG generates a set of candidates for the unschedulable TT frames, and another set for schedulable TT frames. First we describe the candidates for unschedulable frames. For each unschedulable TT frame f_{TT}, CLG identifies the first dataflow link $l_x \in \mathcal{R}_{VL}(\mathcal{M}_F(f_{TT}))$ where f_{TT} is unschedulable, i.e., where $f_{TT}^{l_x}$ is sent too late for f_{TT} to reach its deadline. CLG creates candidate solutions by performing separately on the *Current* solution: *reschedule* moves to $f_{TT}^{l_x}$, *reroute* to f_{TT}, and *packing* and *fragmenting* moves to the message m_{TT} packed by frame $f_{TT}^{l_x}$. In case f_{TT} packs several messages, CLG performs an *unpack* move instead. Similarly, if f_{TT} pack a message fragment, CLG performs an *unfragment* move. Next, CLG targets TT frames that might delay f_{TT} excessively. The high rate frames and the very "large" frames on l_x are such frames. CLG *reroutes* the TT frame with the highest rate on l_x to another link, thus decongesting l_x and increasing f_{TT}'s chances to be schedulable. Similarly, CLG *reroutes* the largest TT frame to another randomly selected route. Furthermore, CLG *reroutes* a random frame on l_x to another randomly selected route.

Our optimization is driven by the cost function specified in Eq. 1 (see Section 8.1). Thus, TS searches for a solution that makes TT and RC frames schedulable, and minimizes the end-to-end delay of the RC frames. Therefore, once the TT frames are schedulable, TS does not look for solutions that reduce the end-to-end delay of the TT frames. Instead, it applies moves to the schedulable TT frames to minimize the end-to-end delay of the RC frames. Thus, the next moves focus on schedulable TT frames.

In this context, first, the CLG algorithm selects the TT frames with the highest degree of schedulablity and generates a candidate solution by *rerouting* each such frame to another route. Although this move may reduce the degree of schedulability of the rerouted frames, as a side effect, it may decongest some dataflow links. Furthermore, CLG also generates other candidates by *rescheduling* these frames.

Second, CLG selects schedulable TT frames with lowest degree of schedulability, and *reroutes* each such frame on a randomly chosen alternative route. Third, CLG generates candidates by *packing* the smallest schedulable TT messages, to consolidate the schedule. Fourth, similarly with the previous candidates, CLG *fragments* in equally sized frame fragments the largest TT messages. For the *pack* and *fragment* moves, CLG randomly choses the number of the frames and the number of the fragments, respectively, so the size of the resulting frames respect the size constraints (see Section 4).

8.3.2 Candidates for RC frames

Similarly with the candidates for TT frames (previously described), CLG generates two sets of candidates: one set for the unschedulable, and another set for schedulable RC frames. For each f_{RC} unschedulable RC frame, the CLG algorithm identifies the first dataflow link l_x where f_{RC} is unschedulable. Then, CLG creates candidate solutions by applying the following moves separately on the current solution: (i) CLG *reroutes* f_{RC} to another, randomly selected route, (ii) *fragments* and (iii) *packs* the message in f_{RC}'s payload. In case f_{RC} packs a message fragment, CLG *unfragments* the message instead. Similarly, if the frame packs several messages, CLG *unpacks* it.

There are cases where a high rate TT frame might greatly delay RC frames. Let $hr_{TT}^{l_x}$ be the TT frame on l_x with the highest rate. *Rerouting* $hr_{TT}^{l_x}$ to another, randomly selected, route decongests l_x, possibly reducing the delay for f_{RC} on this dataflow link. *Rescheduling* $hr_{TT}^{l_x}$ might create sufficient time to reduce f_{RC}'s delay. CLG also creates candidates by *packing* and *fragmenting* the message transported by $hr_{TT}^{l_x}$. Similarly to the high rate TT frame $hr^{l_x}_{TT}$ on l_x, there are cases where large TT frame will delay RC frames. Let $lg^{l_x}_{TT}$ be the largest TT frame on l_x. CLG applies moves that *reroute* $lg^{l_x}_{TT}$, *pack* and *fragment* the message carried by $lg^{l_x}_{TT}$ and moves that *advance* and *postpone* $lg^{l_x}_{TT}$ on l_x, just like in the case of $hr^{l_x}_{TT}$.

Next, CLG focuses on schedulable RC frames to improve their schedulability. For these candidates, first, CLG targets the f_{RC} RC frames with highest degree of schedulability, *rerouting* each such frame to another route. Although this move may reduce the degree of schedulability of f_{RC}, as a side effect, it may decongest some dataflow links, reducing the worst-case end-to-end delay (WCD) of other RC frames. Second, CLG focuses on schedulable RC frames with the lowest degree of schedulability, *rerouting* them in order to increase their schedulability. Third, CLG focuses on the smallest and largest RC frames. Thus, CLG creates candidates by *packing* the smallest RC messages, and by *fragmenting* the largest RC messages, respectively. The packing and fragmenting moves are done such that they respect the constraints presented in Section 8.2.

8.3.3 Randomly generated candidates

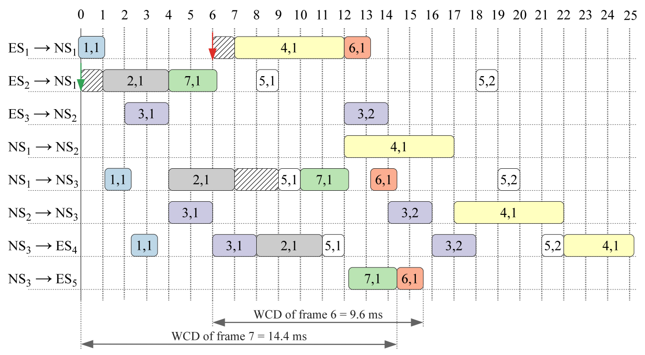
As the previous moves are targeting specific frames, in order to increase the degree of schedulability, CLG introduces a third set of candidates. On a randomly selected set of frames, CLG randomly applies packing, fragmenting or routing moves.

8.4 Tabu Search Example

We illustrate in this section how Tabu Search works. We consider the applications from Fig. 3. The current solution, which is also the best-so-far solution, is presented in Fig. 11a. This solution is also presented in Fig. 6, and is obtained from Fig. 4b by *rerouting* the TT frame f_4 via NS_2. The following 3 solutions, Fig. 11b to Fig. 12a show possible candidate solutions obtained from Fig. 11a. Next to each solution, we present the associated tabu list. We consider a tabu tenure of 5. The current state of the tabu list is shown next to Fig. 11a.

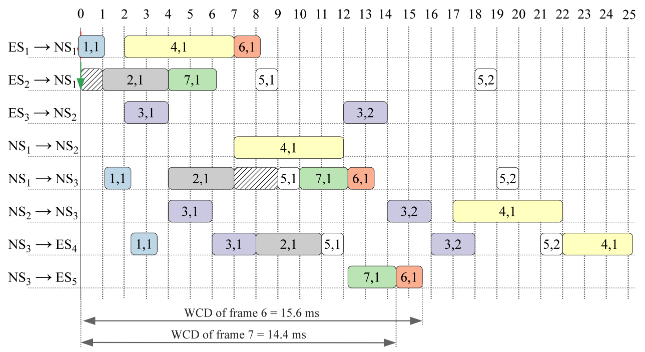
To reduce the delays, the CLG algorithm proposes candidates by rescheduling the largest TT frame on the dataflow link where it delays RC frames. Fig. 11b presents such a candidate solution, advancing f_4 in the schedule of [ES_1, NS_1]. This solution is tabu (tenure 4), and because this candidate is not better than the *Best* solution, it is ignored.

Another set of candidate solutions is obtained by fragmenting the largest RC frames in the system. Fig. 11c presents such a solution. Message m_7 is fragmented into $f_{7/1}$ and $f_{7/2}$. The newly created frames have the same BAG as f_7, of 16 ms, and a transmission duration of $C_{7/1} = C_{7/2} = 1.25ms$. Thus, $f_{7/1,1}$ can be transmitted on [NS_1, NS_3], in the interval between $f_{2,1}$ and $f_{5,1}$, which previously was timely blocked for f_7. This move reduces the WCD of m_7 to 12.5 ms, thus improving the solution. Overall, this solution is better than the *Current* solution. If this candidate solution is chosen as the next solution, the tabu for the fragmenting m_7 is added to the head of the tabu list, with a tenure of 5. The tenures of the other tabus in the list



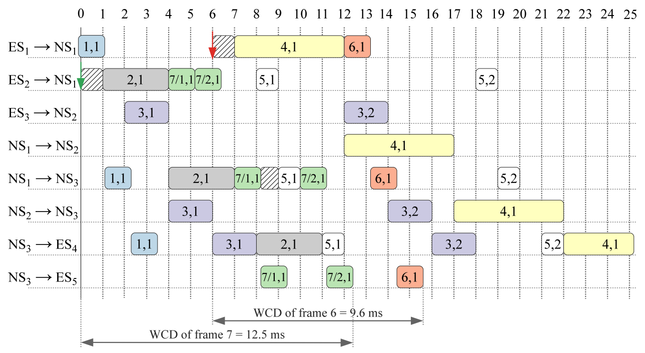
Tenure	Tabu
5	Reschedule f_4
4	Reschedule f_5
3	Reroute f_4
2	Reschedule f_2
1	Reroute f_2

(a) Current solution



Tenure	Tabu
5	Reschedule f_4
4	Reschedule f_5
3	Reroute f_4
2	Reschedule f_2
1	Reroute f_2

(b) Reschedule f_4 does not improve the best-so-far solution and is tabu, thus ignored



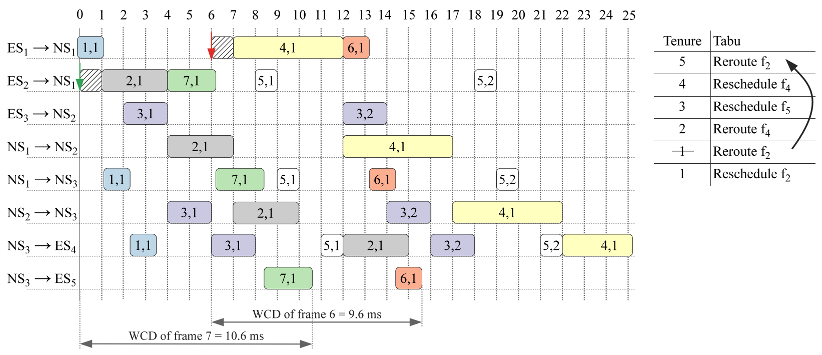
Tenure	Tabu
5	Fragment m_7
4	Reschedule f_4
3	Reschedule f_5
2	Reroute f_4
1	Reschedule f_2
removed	Reroute f_2

(c) Fragment message m_7. Better than the current solution

Fig. 11: Candidate solutions and their tabu list

are decremented, and the "Reroute f_2" tabu, previously with a tenure of 1, is removed from the list. The update tabu list is in Fig. 11c.

Rerouting f_2 via NS_2, see Fig. 12a, reduces the WCD for f_7 from 14.4 to 10.6 ms. Although the move is tabu (tenure 1), the solution is better than the *Current* and *Best* solutions, and thus, its tabu status is *aspired* and the solution is accepted as the



(a) Reroute f_2 via NS_2. Although tabu, the move results in a solution better than the current solution, and thus accepted as the best so far and set as current

Fig. 12: Candidate solutions and their tabu list (continued)

Current and best-so-far *Best* solution. The next TS iteration will continue with this solution as *Current*. The updated tabu list is also presented in Fig. 12a.

9 Experimental evaluation

For the evaluation of our proposed optimization approach, “Design Optimization of TTEthernet-based Systems” (DOTTS), we used 30 synthetic test cases and two real-life case studies. 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.

The details of the test cases are presented in Table 1. For the synthetic test cases, we have used 10 network topologies, and we have randomly generated the parameters for the frames, taking into account the details of the TTEthernet protocol. All the dataflow links have a transmission speed of 100 Mbps. In columns 3–10, we have the details of each test case: the number of ESes, NSes, the load of the system, the number of messages, the minimum and maximum message size in the test case, and the minimum and the maximum message period. The load within an application cycle T_{cycle} is calculated as the ratio of the sum of the required bandwidth by each message divided by the network speed. The required bandwidth for a message is computed as the size of the message divided by the period. The number of frame instances in the network, considering a one-to-one mapping of messages to frames, can be found in column 11. This number is much larger than the number of messages since there is a frame instance for each period of a message on a virtual link.

We used a time limit of 45 minutes for all algorithms and all test cases. We were first interested to determine how close are the results obtained by DOTTS to the optimal result. Thus, we have run an exhaustive search for a small test case (called test case 10 in Table 1) and obtained thus the optimal solution. Running DOTTS for 45 minutes on this test case we have been able to obtain a result which is less than 3.5% from the optimal value of the cost function. In addition, we have selected test case 18, presented in Table 1. The design space for this test case is prohibitively large to perform an exhaustive search. Thus, we ran DOTTS for 12 hours, for 8 times. Out of the 8 long runs, 6 runs returned solutions with the same value of the cost function -618.702, which was the best cost function value among the 8 runs. We considered this to be a "near-optimal" solution. Running DOTTS for 45 minutes returned a solution with a cost function of -612.742, thus less than 1% away from this "near-optimal" solution.

We used 7 sets of test cases. In the first set of test cases, "Set 1", we use 10 test cases where we gradually increase the size of the system, both in the number of networks and the number of messages. In the next five sets, labeled "Set 2" to "Set 6", we consider in each set a fixed architecture, and we increase the load of the system on that architecture. "Set 2" uses the topology of test case 12, "Set 3" uses the topology of test case 14, and "Set 6" uses the topology of test case 17.

We have used the following weights w_{TT} and w_{RC}. For the case when all the TT and RC frames are schedulable we use $w_{TT} = 0$ (once the TT frames are schedulable, we ignore δ_{TT} and focus on finding solutions that minimize the WCD of the RC frames), and $w_{RC} = 1$. If there are TT frames which are not schedulable, we use a penalty value, $w_{TT} = 2,000$. The same penalty value is used if there are unschedulable RC frames, $w_{RC} = 2,000$.

We were interested to evaluate the performance of DOTTS. The results obtained by DOTTS were compared with four other optimization approaches. The first approach is the Straightforward Solution (SS) presented in Section 7.1 and implemented by the box "Initial Solution" in Fig. 8. This is what a good designer would do without the help of our optimization tool. The other three approaches are based on the same Tabu Search optimization as DOTTS, but they restrict the type of optimization performed. Thus, Routing Optimization (RO) optimizes only routing, using SS for packing and scheduling. Packing and Fragmenting Optimization (PFO) optimizes only fragmenting and packing, and not routing and scheduling. Scheduling Optimization (SO) optimizes the schedules but keeps the packing and routing from SS. These TS implementations correspond to the boxes RO, PFO and SO in Fig. 8, where only the respective type of moves are performed in the TS.

The results are presented in Table 1. We are interested in finding schedulable implementations. Thus, for each optimization algorithm, we report the percentage of schedulable messages in the system, after applying the respective optimization. As we can see from the results, DOTTS is able to find schedulable implementations (all the TT and RC messages are schedulable) for all the test cases in Table 1. On the other hand, SS performs poorly, with only 42% schedulable messages for example for test case 10 (column 12). This shows that performing the design optimization of TTEthernet-based systems is very important.

Next, we compared DOTTS with RO, PFO and SO. The question is, where is the improvement of DOTTS coming from, compared to SS, from which kind of optimization: routing, packing/fragmenting or scheduling? SO, which performs schedule optimization, obtains the best result among RO, PFO and SO, but very rarely obtains schedulable solutions. Furthermore, PFO and RO are not consistently better one than the other. The conclusion is that the optimizations should all be used together, as we do in DOTTS.

Table 1: Experimental results

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)	(16)	(17)
Set	Test case	ES	NS	Load	Msgs	Min size	Max size	T_{min}	T_{max}	Frame Instances	SS Sched.%	RO Sched.%	PFO Sched.%	SO Sched.%	DOTTS Sched.%	Δ_{WCD} [%]
1	10	5	3	30	61	29	1451	2	25	697	42.62	50.81	54.38	83.60	100.00	–
	11	6	3	40	100	70	1429	2	60	1470	51.00	56.00	63.33	99.00	100.00	–
	12	8	5	50	99	6	1458	2	37.5	1300	51.51	58.58	60.20	88.88	100.00	–
	13	10	5	50	109	10	1470	2	37.5	1557	46.78	51.37	57.54	88.07	100.00	–
	14	12	4	40	101	1	1347	2	37.5	24146	50.49	60.39	61.00	99.01	100.00	–
	15	20	5	60	84	2	1461	1	150	2388	52.38	67.85	68.86	97.19	100.00	–
	16	20	5	90	145	4	1469	2	126	5509	58.62	62.96	63.88	81.37	100.00	–
	17	35	8	40	132	98	1471	4	250	4064	63.63	66.66	68.18	98.48	100.00	–
	18	37	8	40	43	1	1470	1	150	2305	53.48	86.04	81.39	95.34	100.00	–
	19	37	8	40	77	2	1467	2	126	2441	48.05	70.12	64.93	100.00	100.00	36.35
2	20	8	5	30	46	29	1463	2	25	640	56.52	69.56	73.91	95.66	100.00	–
	21			40	63	72	1461	2	37.5	929	41.26	50.79	52.45	95.23	100.00	–
	22			50	99	6	1458	2	37.5	1300	51.51	58.58	60.20	88.88	100.00	–
	23			60	106	1	1468	2	37.5	1579	48.11	54.71	56.19	94.33	100.00	–
3	30	12	4	30	46	20	1443	2	25	536	56.52	73.91	76.08	97.82	100.00	–
	31			40	50	28	1470	2	25	660	52.00	72.00	74.00	98.00	100.00	–
	32			50	99	19	1464	2	37.5	1542	51.51	60.60	61.61	96.96	100.00	–
	33			60	153	22	1468	2	37.5	1899	49.67	54.24	57.33	94.77	100.00	–
4	40	21	6	40	91	13	1442	2	37.5	1156	45.05	56.04	72.54	100.00	100.00	12.76
	41			50	86	31	1467	2	37.5	1303	47.67	60.46	62.29	90.98	100.00	–
	42			60	165	20	1469	2	37.5	2178	49.09	57.57	59.84	90.97	100.00	–
	43			70	131	8	1466	2	37.5	1785	46.56	56.48	55.49	86.12	100.00	–
5	50	30	8	30	67	26	1445	2	37.5	897	46.26	65.67	65.67	100.00	100.00	42.99
	51			40	180	18	1459	4	375	5809	48.88	53.33	55.00	100.00	100.00	0
	52			50	220	20	1469	4	375	6871	50.00	53.18	54.09	78.18	100.00	–
	53			60	220	17	1471	4	375	7811	50.00	51.81	54.54	74.09	100.00	–
6	60	35	8	40	51	11	1466	2	37.5	785	41.17	62.74	64.04	100.00	100.00	2.21
	61			50	122	5	1466	2	37.5	1596	50.00	58.19	65.11	97.67	100.00	–
	62			60	133	24	1460	2	37.5	1838	45.86	53.38	61.72	92.72	100.00	–
	63			70	173	3	1464	2	37.5	2481	46.82	52.02	60.46	96.94	100.00	–
7	auto	15	7	50	79	28	1461	4	100	5180	53.16	58.22	72.34	89.87	100.00	–
	orion	31	14	40	187	20	1460	4	375	6130	46.52	58.82	57.75	100.00	100.00	43.81

Moreover, even in the rare cases where SO finds schedulable solutions, DOTTS is able to improve on that solution by reducing the WCD of the RC frames. The last column in Table 1 presents this improvement as the average percentage improvement in the WCD of RC frames, between the solution found by SO and the solution found by DOTTS. The percentage improvement $\Delta_{WCD}(f_i)$ for a RC frame f_i is computed according to Eq. 3, where $R_{f_i}^{SO}$ is the WCD for f_i obtained by SO, and $R_{f_i}^{DOTTS}$ is the WCD for f_i obtained by DOTTS. The average percentage improvement Δ_{WCD} presented in Table 1 is obtained by averaging the percentage improvement $\Delta_{WCD}(f_i)$ for all the f_i RC frames in each test case,

$$\Delta_{WCD}(f_i) = \frac{R_{f_i}^{SO} - R_{f_i}^{DOTTS}}{R_{f_i}^{SO}} 100. \tag{3}$$

As we can see, for example for the test case 50, DOTTS is able to significantly reduce the WCD of the RC frames in this case with 42.99%.

From the experiments in Table 1 we can see that DOTTS is able to produce schedulable results as the size of the system increases. By using the sets "Set 2" to "Set 6" we were interested to determined how DOTTS handles increased loads (while the architecture in a set does not change). For example, in "Set 4" we have used an architecture of 21 ESes and 6 NSes and we have increased the number of messages leading to loads of 40 to 70%. As we can see, although there are situations where SO is able to find schedulable solutions when the load is reduced (benchmarks 40, 50, 60), only DOTTS is able to find schedulable implementation as the load of the system increased.

In the last set of experiments, labeled with "Set 7", we used two real-life benchmarks. The first test case is derived from (Mohammad and Al-holou, 2010), based on the SAE automotive communication benchmark (SAE, 1993). In this benchmark we have 22 network nodes (ESes and NSes), and 79 messages (with the parameters generated based on the messages presented in (Mohammad and Al-holou, 2010)). The results for this test case are shown in Table 1, in the row labelled "auto". The other test case is derived from (Paulitsch et al, 2011), based on the Orion Crew Exploration Vehicle (CEV), 606E baseline (Paulitsch et al, 2011) and labeled in Table 1 with "orion". In this test case we have 45 network nodes (ESes and NSes) and 187 messages (with the parameters generated based on the messages presented in (Paulitsch et al, 2011)). The topology for this test case is shown in Fig. 13. The results obtained for the real-life test cases confirm the results of the synthetic test cases.

Finally, we have validated the output produced by DOTTS. We have taken the solution produced for the "orion" test case and we have used the TTEthernet simulator from (Zafirov, 2013) to simulate it, validating thus the results.

10 Conclusions

In this paper we have addressed the optimization of TTEthernet-based real-time systems. TTEthernet is very suitable for mixed-criticality systems, both in the temporal and safety domain. In the temporal domain, TTEthernet offers three types of traffic classes, Time-Triggered, Rate Constrained and Best Effort. In the safety domain,

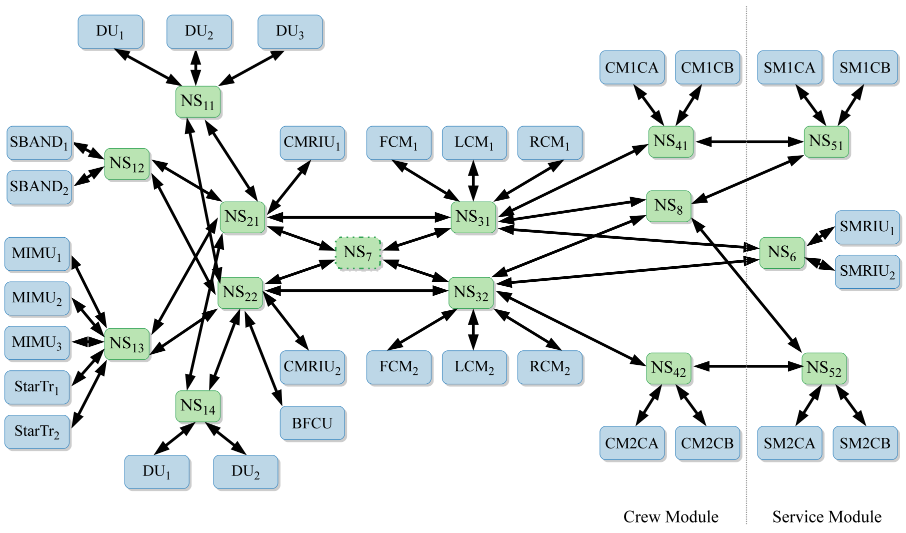


Fig. 13: Network topology of the Orion CEV, derived from (Paulitsch et al, 2011)

the protocol offers separation between mixed-criticality frames using the concept of virtual links, and protocol-level specialized dependability services.

We have considered mixed-criticality hard real-time applications implemented on distributed heterogenous architectures. Given the sets of TT and RC messages and the topology of the network, we have proposed a Tabu Search optimization strategy for the packing of messages into frames, assignment of frames to virtual links, routing of virtual links and synthesis of the TT schedules. The optimization is performed such that the frames are schedulable, and the degree of schedulability is improved. The results on several synthetic benchmarks and two real-life case studies show that through the careful optimization of communication design, significant improvements can be obtained.

Acknowledgements

This work has been funded by the Advanced Research & Technology for Embedded Intelligence and Systems (ARTEMIS) within the project ‘RECOMP’, support code 01IS10001A, agreement no. 100202.

References

Al Sheikh A, Brun O, Chéramy M, Hladik PE (2013) Optimal design of virtual links in AFDX networks. Real-Time Systems Journal 49(3):308–336

Almeida L, Pedreiras P, Fonseca J (2002a) The FTT-CAN protocol: why and how. IEEE Transactions on Industrial Electronics 49(6):1189–1201

Almeida L, Tovar E, Fonseca JAG, Vasques F (2002b) Schedulability analysis of real-time traffic in WorldFIP networks: an integrated approach. IEEE Transactions on Industrial Electronics 49(5):1165–1174

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

Ayed H, Mifdaoui A, Fraboul C (2012) Frame packing strategy within gateways for multi-cluster avionics embedded networks. In: Proceedings of Emerging Technologies Factory Automation, pp 1–8

Azim A, Carvajal G, Pellizzoni R, Fischmeister S (2014) Generation of communication schedules for multi-mode distributed real-time applications. In: Design, Automation and Test in Europe Conference and Exhibition, pp 1–6

Baransel C, Dobosiewicz W, Gburzynski P (1995) Routing in multihop packet switching networks: Gb/s challenge. IEEE Network 9(3):38–61

Burns A, Davis R (2013) Mixed criticality on Controller Area Network. In: Proceedings of the Euromicro Conference on Real-Time Systems, pp 125–134

Cormen TH, Leiserson CE, Rivest RL, Stein C (2009) Introduction to Algorithms, Third Edition, 3rd edn. The MIT Press

Cummings R, Richter K, Ernst R, Diemer J, Ghosal A (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

Davis R, Burns A (2009) Robust priority assignment for messages on ControllerArea Network (CAN). Real-Time Systems 41(2):152–180

Davis R, Burns A, Bril R, Lukkien J (2007) Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised. Real-Time Systems 35(3):239–272

Decotignie JD (2005) Ethernet-based real-time and industrial communications. Proceedings of the IEEE 93(6):1102–1117

Dobrin R, Fohler G (2001) Implementing off-line message scheduling on Controller Area Network (CAN). In: Proceedings of the International Conference on Emerging Technologies and Factory Automation, pp 241–245 vol.1

ETG (2013) ETG.1000.1 EtherCAT Specification. EtherCAT Technology Group

Fletcher M (2009) Progression of an open architecture: from Orion to Altair and LSS. White paper S65-5000-20-0, Honeywell, International

Garey MR, Johnson DS (1979) Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA

Gendreau M (2002) An Introduction to Tabu Search. Centre for Research on Transportation (C.R.T.)

Glover F, Laguna M (1997) Tabu Search. Kluwer Academic Publishers, Norwell, MA, USA

Grammatikakis MD, Hsu D, Kraetzl M, Sibeyn JF (1998) Packet routing in fixed-connection networks: A survey. Journal of Parallel and Distributed Computing 54(2):77 – 132

Hanzalek Z, Burget P, Sucha P (2010) Profinet io irt message scheduling with temporal constraints. IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS 6(3):369–380, DOI 10.1109/TII.2010.2052819

Herpel T, Kloiber B, German R, Fey S (2009) Routing of Safety-Relevant Messages in Automotive ECU Networks. In: Proceedings of the Vehicular Technology Conference, pp 1–5

IEEE (2009) IEEE 802.1Qav - IEEE Standard for Local and Metropolitan Area Networks - Virtual Bridged Local Area Networks Amendment 12: Forwarding and Queueing Enhancements for Time-Sensitive Streams. The Institute of Electrical and Electronics Engineers, Inc.

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.

IEEE (2011a) IEEE 802.1AS - IEEE Standard for Local and Metropolitan Area Networks - Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks. The Institute of Electrical and Electronics Engineers, Inc.

IEEE (2011b) IEEE 802.1BA - IEEE Standard for Local and Metropolitan Area Networks - Audio Video Bridging (AVB) Systems. The Institute of Electrical and Electronics Engineers, Inc.

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

ISO (2003) ISO 11898: Road Vehicles – Controller Area Network (CAN). International Organization for Standardization (ISO), Geneva, Switzerland

ISO (2010) ISO 10681: Road vehicles – Communication on FlexRay. International Organization for Standardization (ISO), Geneva, Switzerland

Kopetz H (2011) Real-Time Systems: Design Principles for Distributed Embedded Applications. Springer

Lee YH, Rachlin E, Scandura PA (2005) Safety and Certification Approaches for Ethernet-Based Aviation Databuses. Tech. Rep. DOT/FAA/AR-05/52, Federal Aviation Administration

Marau R, Almeida L, Pedreiras P, Lakshmanan K, Rajkumar R (2010) Utilization-based schedulability analysis for switched Ethernet aiming dynamic QoS management. In: Proceedings of the Conference on Emerging Technologies and Factory Automation, pp 1–10

Mohammad U, Al-holou N (2010) Development of an automotive communication benchmark. Canadian Journal on Electrical and Electronics Engineering 1(5):99–115

Nam MY, Lee J, Park KJ, Sha L, Kang K (2013) Guaranteeing the end-to-end latency of an IMA system with an increasing workload. IEEE Transactions on Computers 99(PrePrints):1

Obermaisser R (2011) Time-Triggered Communication. CRC Press, Inc.

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

Pedreiras P, Almeida L (2004) Message routing in multi-segment FTT networks: the isochronous approach. In: Proceedings of the Parallel and Distributed Processing Symposium, pp 122–129

Pedreiras P, Gai P, Almeida L, Buttazzo G (2005) FTT-Ethernet: a flexible real-time communication protocol that supports dynamic QoS management on Ethernet-based systems. IEEE Transactions on Industrial Informatics 1(3):162–172

Pop P, Eles P, Peng Z (1999) Scheduling with optimized communication for time-triggered embedded systems. In: Proceedings of the International Workshop on Hardware/Software Codesign, pp 178–182

Pop P, Eles P, Peng Z (2005) Schedulability-driven frame packing for multicluster distributed embedded systems. ACM Transasctions on Embedded Computing Systems 4(1):112–140

Pop T, Pop P, Eles P, Peng Z, Andrei A (2008) Timing analysis of the FlexRay communication protocol. Real-Time Systems 39(1-3):205–235

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

SAE (1993) SAE J2056/1 Class C Application Requirement Considerations. Standard, SAE International

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

Saket R, Navet N (2006) Frame packing algorithms for automotive applications. Journal of Embedded Computing 2(1):93–102

Sandstrom K, Norstom C, Ahlmark M (2000) Frame packing in real-time communication. In: Proceedings of the Conference on Real-Time Computing Systems and Applications, pp 399–403

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

Schumacher M, Jasperneite J, Weber K (2008) A new approach for increasing the performance of the industrial ethernet system profinet. In: International Workshop on Factory Communication Systems, pp 159–167, DOI 10.1109/WFCS.2008.4638725

Steinbach T, Lim HT, Korf F, Schmidt TC, Herrscher D, Wolisz A (2012) Tomorrow's In-Car Interconnect? A Competitive Evaluation of IEEE 802.1 AVB and Time-Triggered Ethernet (AS6802). In: IEEE Vehicular Technology Conference, IEEE Press, pp 1–5

Steiner W (2010) An Evaluation of SMT-based Schedule Synthesis For Time-Triggered Multi-Hop Networks. In: Proceedings of the Real-Time Systems Symposium, pp 375–384

Steiner W (2011) Synthesis of Static Communication Schedules for Mixed-Criticality Systems. In: Proceedings of the International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops, pp 11–18

Steiner W, Bauer G, Hall B, Paulitsch M, Varadarajan S (2009) TTEthernet Dataflow Concept. In: Proceedings of the International Symposium on Network Computing and Applications, pp 319–322

Suen J, Kegley R, Preston J (2013) Affordable avionic networks with Gigabit Ethernet assessing the suitability of commercial components for airborne use. In: Proceedings of SoutheastCon, pp 1–6

Suethanuwong E (2012) Scheduling time-triggered traffic in TTEthernet systems. In: Proceedings of the Conference on Emerging Technologies and Factory Automation, pp 1–4

Tanasa B, Dutta Bordoloi U, Eles P, Peng Z (2011) Reliability-aware Frame Packing for the Static Segment of FlexRay. In: Proceedings of the International Conference on Embedded Software, ACM, New York, NY, USA, EMSOFT '11, pp 175–184

Tămaş-Selicean D (2014) Design of Mixed-Criticality Applications on Distributed Real-Time Systems. PhD thesis, Technical University of Denmark

Tămaş-Selicean D, Pop P (2011) Design Optimization of Mixed-Criticality Real-Time Applications on Cost-Constrained Partitioned Architectures. In: Proceedings of the Real-Time Systems Symposium, pp 24–33

Tămaş-Selicean D, Pop P, Steiner W (2012) Synthesis of Communication Schedules for TTEthernet-based Mixed-Criticality Systems. In: Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis, pp 473–482

Wang B, Hou J (2000) Multicast routing and its QoS extension: problems, algorithms, and protocols. IEEE Network 14(1):22–36

Wisniewski L, Schumacher M, Jasperneite J, Schriegel S (2012) Fast and simple scheduling algorithm for profinet irt networks. In: International Workshop on Factory Communication Systems, pp 141–144, DOI 10.1109/WFCS.2012.6242556

Zafirov AT (2013) Modeling and simulation of the TTEthernet communication protocol

Appendix: Notations and abbreviations

Abbreviation	Meaning
ASAP	As-Soon-As-Possible (ASAP)
AVB	Audio Video Bridging
BAG	Bandwidth Allocation Gap
BE	Best-Effort
CAN	Controller Area Network
DOTTS	Design Optimization of TTEthernet-based Systems
ES	End System
ET	Event-Triggered
FIFO	First In First Out
FU	Filtering Unit
NS	Network Switch
PE	Processing Element
PFO	Packing and Fragmenting Optimization
RC	Rate-Constrained
RO	Routing Optimization
SIL	Safety-Integrity Level
SMT	Satisfiability Modulo Theory
SO	Scheduling Optimization
SS	Straightforward Solution
TDMA	Time-Division Multiple Access
TP	Traffic Policing task
TR	Traffic Regulator task
TS	Tabu Search
TT	Time-Triggered
TTP	Time-Triggered Protocol
VL	Virtual Link
WCD	Worst-Case end-to-end Delay

Symbol	Meaning
$\mathcal{G}(\mathcal{V},\mathcal{E})$	TTEthernet cluster
$\mathcal{V} = \mathcal{ES} \cup \mathcal{NS}$	Set of all the end systems and network switches in the cluster
$\mathcal{ES}$	Set of all the end systems
ES_i^{src}	The source end system for frame f_i
$\mathcal{ES}_i^{dest}$	The set of destination end systems for frame f_i
$\mathcal{NS}$	Set of all the network switches
ES_i	An end system
NS_i	A network switch
$\mathcal{E}$	Set of physical links
$\mathcal{L}$	Set of dataflow links in the cluster
l_i	A dataflow link
$\mathcal{DP}$	Set of dataflow paths in the cluster
dp_i	A dataflow path
$\mathcal{VL}$	Set of virtual links in the cluster
vl_i	A virtual link
$BAG(vl_i)$	The BAG of vl_i
$\mathcal{B}$	The set of BAG for all VLs
$\mathcal{B}^\circ$	The initial set of BAG for all VLs
$\mathcal{R}_{VL}(vl_i)$	The routing of virtual link vl_i
$\mathcal{M} = \mathcal{M}^{TT} \cup \mathcal{M}^{RC} \cup \mathcal{M}^{BE}$	Set of all the messages in the cluster
$\mathcal{M}^{TT}$	Set of the TT messages
$\mathcal{M}^{RC}$	Set of the RC messages
$\mathcal{M}^{BE}$	Set of the BE messages
m_i	A message
$m_i.rate$	The rate of message m_i
$m_i.period$	The period of message m_i
$m_i.deadline$	The deadline of message m_i
$m_i.size$	The size of message m_i

Symbol	Meaning
$\Phi_m(m_i)$	Fragmenting of message m_i into message fragments
Φ_m°	The initial fragmenting of messages into message fragments
$\mathcal{M}^+$	The set of message fragments, and messages that were not fragmented
$\mathcal{P}$	The packing of messages
$\mathcal{P}^\circ$	The initial packing of messages
$\mathcal{A}_i$	An application
τ_j	A task
$\mathcal{F}$	Set of all the frames in the cluster
f_i	A frame
$f_i.deadline$	Deadline for f_i
$f_i.offset$	Offset, i.e., send time relative to the start of the period of frame f_i
$f_{i,x}$	x^{th} instance of frame f_i
$f_{i,x}.jitter$	Jitter for $f_{i,x}$
$f_{i,x}^{l_j}$	The x^{th} instance of f_i on dataflow link l_j
$pred(f_{i,x}^{l_j})$	The set of predecessor frame instances of $f_{i,x}$ on l_j
$succ(f_{i,x}^{l_j})$	The set of successor frame instances of $f_{i,x}$ on l_j
R_{f_i}	Worst-case delay of f_i
$\mathcal{M}_F(f_i)$	The assignment of frame to VLs
$C_j^{[\nu_m,\nu_n]}$	Transmission duration of f_j on dataflow link $[\nu_m,\nu_n]$
$B_{1,Tx}$	A transmission buffer
$Q_{1,Tx}$	A transmission queue in an ES
Q_{Tx}	An outgoing queue in a NS
TT_R	Receiver task
$\mathcal{S}$	Complete set of local schedules
$\mathcal{S}^\circ$	Initial set of local schedules
$\mathcal{S}_R$	Receive schedule
$\mathcal{S}_S$	Send schedule
T_{cycle}	System cycle
$BW(vl_i)$	The maximum bandwidth required by vl_i
δ_{TT}	Degree of schedulability for the TT frames
δ_{RC}	Degree of schedulability for the RC frames
w_{TT}	Weight for the TT frames, used to compute the cost function
w_{RC}	Weight for the RC frames, used to compute the cost function
Best	The best-so-far solution in the Tabu Search algorithm
Current	The current solution in the Tabu Search algorithm
Next	The solution selected as the next solution in the Tabu Search algorithm
$\mathcal{C}$	Candidate list
L	Tabu list
maxLSize	Tabu tenure
tabu(*Next*)	The tabu of the move that generated the *Next* solution
$hr_{TT}^{l_x}$	The TT frame with the highest rate on dataflow link l_x
$lg_{TT}^{l_x}$	The largest TT frame on dataflow link l_x
$BW_{Avail}(l_j)$	The available bandwidth on dataflow link l_j
$BW_{Req}(l_j)$	The required bandwidth on dataflow link l_j
$BW_{\%}^{BW}$	The percentage of BE messages that have their bandwidth requirements met