Presented at the 10th International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS 2012, part of the Eighth Embedded Systems Week, ACM, 2012

Synthesis of Communication Schedules for TTEthernet-Based Mixed-Criticality Systems

Domiţian Tămaş–Selicean
Technical University of Denmark
Kongens Lyngby, Denmark
dota@imm.dtu.dk

Paul Pop
Technical University of Denmark
Kongens Lyngby, Denmark
paul.pop@imm.dtu.dk

Wilfried Steiner
TTTech Computertechnik AG
Vienna, Austria
wilfried.steiner@tttech.com

ABSTRACT

In this paper we are interested in safety-critical distributed systems, composed of heterogeneous processing elements interconnected using the TTEthernet protocol. We address hard real-time mixed-criticality applications, which may have different criticality levels, and we focus on the optimization of the communication configuration. TTEthernet integrates three types of traffic: Time-Triggered (TT) messages, Event-Triggered (ET) messages with bounded end-to-end delay, also called Rate Constrained (RC) messages, and Best-Effort (BE) messages, 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. TT and RC traffic can carry safety-critical messages, while BE messages are non-critical. Mixed-criticality tasks and messages can be integrated onto the same architecture only if there is enough spatial and temporal separation among them. TTEthernet offers spatial separation for mixed-criticality messages through the concept of virtual links, and temporal separation, enforced through schedule tables for TT messages and bandwidth allocation for RC messages. Given the set of mixed-criticality messages in the system and the topology of the virtual links on which the messages are transmitted, we are interested to synthesize offline the static schedules for the TT messages, such that the deadlines for the TT and RC messages are satisfied, and the end-to-end delay of the RC traffic is minimized. We have proposed a Tabu Search-based approach to solve this optimization problem. The proposed algorithm has been evaluated using several benchmarks.

Categories and Subject Descriptors

B.4.4 [**Input/Output and Data Communications**]: Performance Analysis and Design Aids; C.2.2 [**Computer-Communication Networks**]: Network Protocols

Keywords

mixed-criticality; real-time systems; optimization of communication configuration; network protocol; partitioned architectures; TTEthernet

1. INTRODUCTION

Depending on the particular application, an embedded system has certain requirements on performance, cost, dependability, size etc. In a *hard real-time* application the correctness depends not only on the logical results of the computations, but also on the physical instant at which these results are produced [21]. *Safety* is a property of a system that will not endanger human life or the environment. *Safety-Integrity Levels* (SILs) capture the required protection against failure when building a safety-critical embedded system, and will dictate the development processes and certification procedures that have to be followed.

There are two basic approaches for handling real-time applications [21]. 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. There has been a long debate in the real-time and embedded systems communities concerning the advantages of each approach [7, 21, 39]. Several aspects have been considered in favor of one or the other approach, such as flexibility, predictability, jitter control, processor utilization, testability etc. However, the consensus is that the right approach depends on the particularities of the application, as it has been shown in an automotive context [23]. This means not only that there is no single "best" approach to be used, but also that within a system the two approaches can be used together, some tasks and messages being TT and others ET.

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 [2], 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) [21]. The trend is towards bus protocols that support both static and dynamic communication [4, 5].

Many safety-critical real-time applications, following physical, modularity or safety constraints, are implemented using distributed architectures, composed of heterogeneous processing elements (PEs), interconnected in a network. Initially, each function was implemented in a separate PE, which has led to a large increase in the number of PEs. The current trends are towards *integrated architectures* where several functions are integrated onto the same PE. Mixed-criticality applications, i.e., applications with different SIL levels, as well as non-critical applications, can be integrated onto the same architecture only if there is enough spatial and temporal separation among them.

In the avionics area, the PE-level separation mechanisms are provided by implementations of the ARINC 653 standard, also called *Integrated Modular Avionics* (IMA) [31]. ARINC 653 consists of hardware-mediated operating system-level spatial and temporal partitioning [31] mechanisms. Similar PE-level separation mechanisms are available in other industries [15, 22]. At the communication level, there are also several partitioning solutions, such as SAFEBus [19], ARINC 664 Specification Part 7 (ARINC 664p7, for short) [3] and TTP [21]. In this paper we are interested in the TTEthernet [5] protocol, which provides both spatial and temporal partitioning, and can handle both TT and ET communication.

There is a large amount of research on hard real-time systems [21]. At the PE-level, researchers have addressed systems with mixed time-criticality requirements, showing how TT/ET tasks [29] or hard/soft real-time tasks [20, 33] can be integrated onto the same

platform. Researchers have also started to address the integration of mixed safety-criticality tasks onto the same platform [8, 9, 10, 37]. In [37], researchers have proposed an optimization approach to determine the mapping of tasks to PEs, the assignment of tasks to partitions, the sequence and size of the time slots on each PE and the schedule tables, such that all the applications are schedulable and the development costs are minimized. In that work, communications were ignored, and a simple un-partitioned statically scheduled shared bus was used.

The problem of the optimization of time-partitions has been addressed at the bus level. Researchers have shown how a TDMA bus such as the TTP [26] and a mixed TT/ET bus such as FlexRay [30] can be optimized to decrease the end-to-end delays. FlexRay allows the sharing of the bus among ET and TT messages, thus offering the advantages of both worlds. In [30], an optimization approach for determining a FlexRay bus configuration is proposed, which is adapted to the particular features of an application and guarantees that all time constraints are satisfied.

Our focus in this paper is on the optimization of the TTEthernet protocol [5], which is a deterministic, synchronized and congestion-free network based on the IEEE 802.3 Ethernet standard and compliant with the ARINC 664p7The widespread Ethernet protocol is known to be unsuitable for real-time or safety-critical applications [3]. For example, in half-duplex implementations, frame collision is unavoidable, leading to unbounded transmission times. The ARINC 664p7 specification [3] is a full-duplex Ethernet network, which emulates point-to-point connectivity over the network by defining *virtual links*, tree-like structures with one sender and one or several receivers (see Section 2). 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 synchronization 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: TT traffic and ET 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 highly suitable for applications of different safety criticality levels, since it offers spatial separation for mixed-criticality messages through the concept of virtual links. For more details on the traffic classes, protocol services and separation mechanisms, see Section 5.

In this paper we are interested in mixed-criticality applications (both in the safety and time domains) implemented using heterogeneous processing elements interconnected using TTEthernet. We consider that the architecture, the topology of the virtual links and the assignment of messages to virtual links is given. Furthermore, we assume that the designer has decided the partitioning of messages into TT, RC or BE, depending on the particularities of the application. We are interested in synthesizing the static communication schedules for the TT messages, such that the TT and RC messages are schedulable, and the end-to-end delay of RC messages is minimized. We have proposed a Tabu Search-based metaheuristic to solve this optimization problem.

There is very limited work in this area. Steiner proposes in [34] 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. The same author has proposed an SMT-solver approach to introduce periodic evenly-spaced slots into the static schedules to help reduce RC delays in [35]. Our Tabu Search-based metaheurstic 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.

The paper is organized as follows: after we introduce the architecture model in Section 2 and the application model in Section 3, we present the problem formulation in Section 4. Section 5 describes how the TTEthernet protocol works. Section 6 presents a motivation example to better understand the problem, and the proposed solution is described in Section 7. The proposed Tabu Search-based approach is evaluated in Section 8.

2. ARCHITECTURE MODEL

A TTEthernet network is composed of a set of clusters. Each cluster consists of a set 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 ES_4, and 2 NSes, NS_1 and NS_2.

Each cluster has its own separate clock synchronization domain, hence the TT schedules are derived per cluster. The problem of optimizing the TT schedules 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), see Fig. 1.

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, $\mathcal{V} = \mathcal{ES} \cup \mathcal{NS} = \{ES_1, ES_2, ES_3, ES_4\} \cup \{NS_1, NS_2\}$, and the physical links $\mathcal{E}$ are depicted with thick, black, double arrows.

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 [3], which is implemented by the TTEthernet protocol, as a "logical unidirectional connection from one source end system to one or more destination end systems". Virtual links connect one sender to multiple receivers. Each virtual link carries a single message.

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. 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, 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, a *dataflow path* dp_i is an ordered sequence of dataflow links connecting one sender to one receiver. For example, in Fig. 1, $vl_1 = dp_1 \cup dp_2$, and dp_1 connects ES_1 to ES_3, while dp_2 connects ES_1 to ES_4 (the dataflow paths are depicted with green arrows). A *dataflow link* $l_i = [\nu_j, \nu_k] \in \mathcal{L}$, where $\mathcal{L}$ is the set of dataflow links

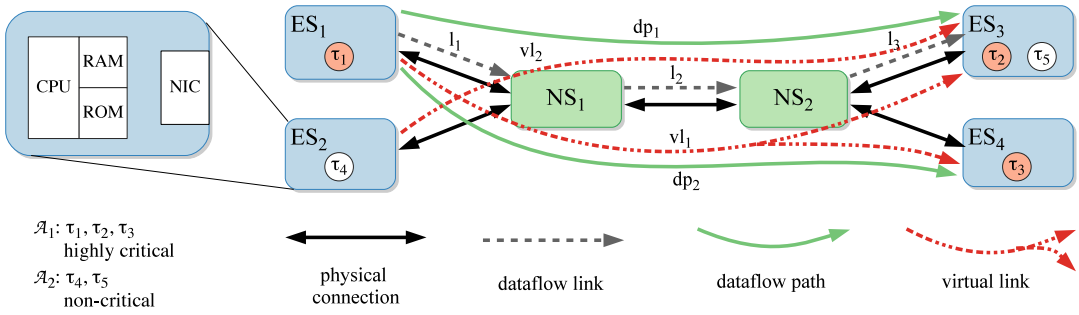


Figure 1: TTEthernet cluster example

in a cluster, is a directed communication connection from v_j to v_k, where v_j and $v_k \in \mathcal{V}$ can be ESes or NSes. Using this notation, a dataflow path such as dp_1 in Fig. 1 can be denoted as $[[ES_1, NS_1], [NS_1, NS_2], [NS_2, ES_3]]$.

For a given application, with tasks mapped to ESes and exchanging messages over the network, there are several possible virtual link configurations, depending on how senders and receivers are grouped. In this paper, we assume that the virtual links are given. Determining the set of virtual links for a given application has been studied in the context of Steiner trees [16, 17].

3. APPLICATION MODEL

In [37] researchers 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 these messages could be corrupted) 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. There are several approaches to integrate task and message scheduling, see [25] for a survey. Each message m_i transmitted using TTEthernet is packed into a frame f_i. In this paper we assume that a frame carries only one message, and that messages are not split into packets. The issue of frame packing [28] is orthogonal to our problem.

We assume that the topology of virtual links and the assignment of frames to virtual links are given. This assignment is captured by the function $\mathcal{M}(f_i) = vl_i, \mathcal{M} : \mathcal{F} \rightarrow \mathcal{VL}$, where $\mathcal{F}$ is the set of all frames in the cluster.

As mentioned, TTEthernet supports three traffic classes: time-trig- gered (TT), rate constrained (RC) and best effort (BE). A bit pattern specified in the frame header identifies the traffic class. We assume that the designer has decided the traffic classes for each frame, and we define the sets $\mathcal{F}^{TT}$, $\mathcal{F}^{RC}$ and $\mathcal{F}^{BE}$, respectively, with $\mathcal{F} = \mathcal{F}^{TT} \cup \mathcal{F}^{RC} \cup \mathcal{F}^{BE}$.

The size $f_i.size$ for each frame $f_i \in \mathcal{F}$ is given. In addition, for the TT and RC frames we know their periods and deadlines, $f_i.period$ and $f_i.deadline$, respectively. RC frames are not necessarily periodic, but have a minimum inter-arrival time f_i. We define the rate of an RC frame f_i as $f_i.rate = 1/f_i.period$. Knowing the size of a frame f_i and the given speed of a dataflow link $[v_j, v_k]$, we can determine the transmission duration $C_i^{[v_j,v_k]}$ of f_i on $[v_j, v_k]$.

4. PROBLEM FORMULATION

The problem we are addressing in this paper can be formulated as follows: given (1) the topology of the network $\mathcal{G}$, (2) the set of TT and RC frames $\mathcal{F}^{TT} \cup \mathcal{F}^{RC}$, (3) the set of virtual links $\mathcal{VL}$, (4) the assignment of frames to virtual links $\mathcal{M}$ and (5) for each frame f_i the size, the deadline and the period, we are interested to find the set of TT schedules $\mathcal{S}$ such that the deadlines for the TT and RC frames are satisfied. Once both TT and RC frames are schedulable several optimization objectives can be tackled. In this paper we are interested to synthesize the schedules $\mathcal{S}$ such that the end-to-end delay of RC frames is minimized. Section 7.1 presents the cost function used for the optimization. In this paper we ignore the BE traffic, but a quality-of-service measure for the BE traffic could easily be added to the objective function, if desired. The problem is illustrated in Section 6 using a motivational example.

5. TTETHERNET PROTOCOL

Let us illustrate how the TTEthernet protocol works using the example in Fig. 2, where we have two end systems, ES_1 and ES_2, and three network switches, NS_1 to NS_3. Task τ_2 on ES_1 sends the TT message m_2 to task τ_4 mapped on ES_2, while task τ_1 on ES_1 sends the RC message m_1 to task τ_3 on ES_2. Let us assume that tasks τ_1 and τ_3 are part of application $\mathcal{A}_1$ and tasks τ_2 and τ_4 belong to application $\mathcal{A}_2$. Furthermore, $\mathcal{A}_1$ and $\mathcal{A}_2$ are of different safety criticality. The separation of the applications is achieved at the CPU-level through partitioning. Thus, tasks τ_1 and τ_3 are placed in partitions $\mathcal{P}_{1,1}$ and $\mathcal{P}_{2,2}$, respectively, while tasks τ_2 and τ_4 are assigned to partitions $\mathcal{P}_{1,2}$ and $\mathcal{P}_{2,1}$, see Fig. 2.

Message m_1 is sent within application $\mathcal{A}_1$ and packed in frame f_1. m_2 is sent within $\mathcal{A}_2$ and packed into the frame f_2. The different criticality frames are separated by assigning them to two different virtual links, vl_1 and vl_2 (not depicted in the figure). Frames f_1 and f_2 have to transit the switch NS_1, which also forwards frames f_3 and f_4, from NS_2 and NS_3, respectively, see Fig. 2.

5.1 Time-Triggered Transmission

In this section we present how TT frames are transmited by TTEthernet, using the example of the TT message m_2 sent from task τ_2 on ES_1, to task τ_4 on ES_2. We depict each step of the TT transmission on Fig. 2 and mark it with a letter from (a) to (m) on a blue background.

Thus, in the first step denoted with (a), task τ_2 packs m_2 into frame f_2 and in the second step (b), f_2 is placed into buffer $B_{1,Tx}$ for transmission. Conceptually, there is one such buffer for every TT message sent from ES_1. 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. Thus, in step (d), the scheduler task TT_S will send f_2 to NS_1 at the time specified in the send schedule S_S stored in ES_1 (c). There are several approaches to the synchronization of tasks (which could be TT or ET) and TT messages [25]. Often, TT tasks are used in conjunction with TT messages, and the task and

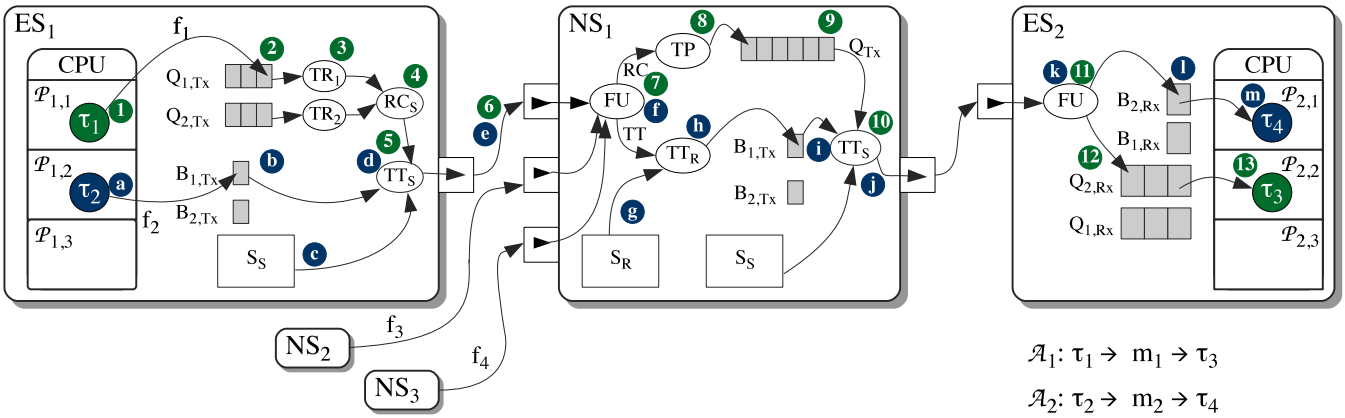


Figure 2: TT and RC message transmission example

message schedules are synchronized such that the task is scheduled to finish before the message is scheduled for transmission.

Next, f_2 is sent on a dataflow link to NS_1 (e). The computational logic of the TTEthernet protocol is implemented in hardware and, conceptually, consists of several hardware tasks working in parallel to implement the protocol services. Such is the case of the Filtering Unit (FU) task, which is invoked every time a frame is received by an NS. The FU checks the integrity and validity of frame f_2 (see step (f)) and forwards it to the TT receiver task TT_R (h), which copies it into the sending buffer $B_{1,Tx}$ for later transmission.

The separation mechanisms implemented by TTEthernet to isolate mixed-criticality frames, such as f_1 and f_2 in our example, are spread across several hardware tasks. In addition, TTEthernet provides fault-tolerance services, such as fault-containment, to the application level. For example, if a task such as τ_2 becomes faulty and sends more messages than scheduled (called a "babbling idiot" failure), the TT sender task TT_S on ES_1 will protect the network as it will only transmit messages as specified in the schedule table S_S.

Also, a TT receiver task TT_R in an NS will rely on a receive schedule S_R (g) 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 (schedule S_S on ES_1 for the case of frame f_2), 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 contain 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 $f_i.offset$. Within an application cycle, the offset of a frame may vary from period to period. However, a particular TTEthernet implementation may restrict the sending times such that the offset of a frame is identical in all the periods (which has the advantage of reducing the size needed to store the schedules). Our implementation considers the general case, and such constraints can be easily added to our optimization approach. Also, a restricted TTEthernet implementation can "emulate" the general case by assigning a message to multiple virtual links.

Let us continue to follow the transmission of f_2 in Fig. 2. The frame has arrived in NS_1 and has been placed in $B_{1,Tx}$ (h). Next, f_2 is sent by the TT sender task TT_S in NS_1 to ES_2 at the time specified in the TT send schedule S_S in NS_1. When f_2 arrives at ES_2 (k), the FU task will store the frame into a dedicated receive buffer $B_{2,Rx}$ (l). Finally, when task τ_4 is activated, it will read f_2 from the buffer (m).

5.2 Rate Constrained Transmission

This section presents how RC traffic is transmitted using the example of frame f_1 sent from τ_1 on ES_1 to τ_3 on ES_2. Similarly to the discussion of TT traffic, we mark each step in Fig. 2 using numbers from (1) to (13), on a green background.

Thus, τ_1 packs message m_1 into frame f_1 (1) and inserts it into a queue $Q_{1,Tx}$ (2). Conceptually, there is one such queue for each RC virtual link. 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. 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 $BAG_i \leq 1/f_i.rate$.

The BAG is enforced by the Traffic Regulator (TR) task. Thus, TR_1 in ES_1 in Fig. 2 will ensure that each BAG_1 interval will contain at most one instance of f_1 (3). Therefore, even if a frame is sent in bursts by a task, it will leave the TR task within a specified BAG. Thus, the maximum bandwidth used by a virtual link vl_i which transmits an RC frame f_i is $BW(vl_i) = f_i.size/BAG_i$. In this paper we assume that the BAG for each RC frame is given by the designer.

Several messages will be sent from an ES. Let us first discuss how RC messages are *multiplexed*, and then we will discuss the *integration* with the TT traffic.

In an ES, the RC scheduler task RC_S (such as the one in ES_1) will multiplex several RC messages (4) coming from the traffic regulator tasks, TR_i, such as TR_1 and TR_2 in ES_1. Fig. 3 depicts how this multiplexing is performed for the frames f_x and f_y with the sizes and BAGs as specified in Fig. 3(a) and (b), respectively. Fig. 3(c) shows how the two frames will be sent on the outgoing dataflow link $[ES_1, NS_1]$ by the RC_S task. In the case several TRs attempt to transmit messages at the same time, due to the multiplexer, the frames waiting to be transmitted will be affected by jitter. This is the case of f_y in Fig. 3(c), which is delayed to allow for the transmission of f_x. Thus, the $f_{y,1}.jitter$ jitter for $f_{y,1}$ equals to the transmission duration of f_x.

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. Hence, for our example on ES_1, the TT_S task on ES_1 will transmit frame f_1 (5) to NS_1 on the dataflow link $[ES_1, NS_1]$ only when there is no TT traffic (6). With integration, contention situations can occur when a TT frame

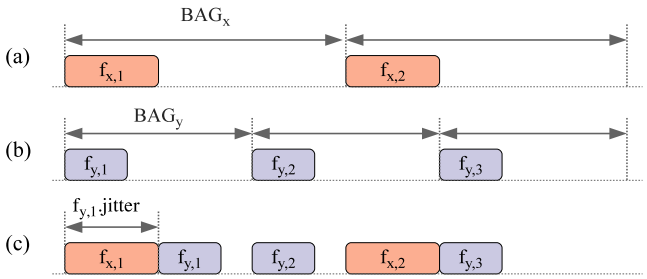


Figure 3: Multiplexing two RC frames

is scheduled for transmission, but an RC frame is already transmitting.

There are three approaches in TTEthernet to handle such situations [5, 36]: (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 *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. In the case (iii) of *pre-emption*, the RC frame is pre-empted, and its transmission is restarted after the TT frame finished transmitting. 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.

When the RC frame f_1 arrives at NS_1, the Filtering Unit task (7) will check its validity and integrity. As mentioned, TTEthernet provides services to separate the mixed-criticality frames, such that a faulty transmission of a lower-criticality frame will not impact negatively a higher-criticality frame. Fault-containment at the level of RC virtual links is provided by the Traffic Policing (TP) task, see NS_1 in Fig. 2. TP implements an algorithm known as *leaky bucket* [3, 5], which checks the time interval between two consecutive instances on the same virtual link. If this interval is shorter than the specified BAG, the frame instance is dropped. Thus, the TP function prevents a faulty ES to send faulty RC frames (more often than allowed) and thus to disturb the network.

After passing the checks of the TP task (8), f_1 is copied to the outgoing queue Q_{Tx} (9). In this paper we assume that all the RC frames have the same priority, thus the TT_S (10) will send the RC frames in Q_{Tx} in a FIFO order, but only when there is no scheduled TT traffic. At the receiving ES, after passing the FU (11) checks, f_1 is copied in the receiving $Q_{2,Rx}$ queue (12). Finally, when τ_3 is activated, it will take f_1 (13) from this queue.

6. MOTIVATIONAL EXAMPLE

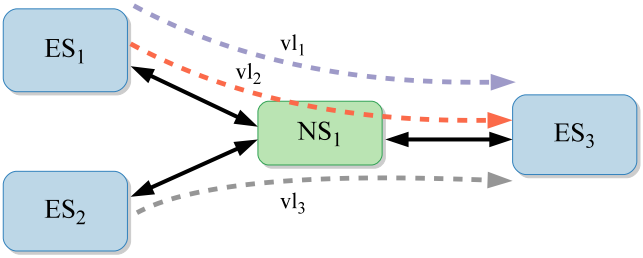
Let us illustrate the schedule synthesis problem presented in Section 4 using the setup from Fig. 4, where we have an architecture model for a cluster composed of three ESes, ES_1 to ES_3 and a network switch NS_1 (see Fig. 4a) and an application model with three frames, see the table in Fig. 4b. We have three virtual links, vl_1, vl_2 and vl_3 one for each frame, f_1, f_2 and f_3, respectively, as captured by the function $\mathcal{M}$ in the table. The periods $f_i.period$, deadlines $f_i.deadline$ and transmission times C_i on a dataflow link are given in the table for each frame. The dataflow links have the same speed, hence the C_i of a frame f_i is the same for each link. For this example we consider that the RC and TT traffic are integrated using a timely block policy, i.e., an RC frame will be delayed if it could block a scheduled TT frame.

Our problem is to determine the TT schedules $\mathcal{S}$ such that all the TT and RC frames are schedulable. The schedulability of a TT frame f_i is easy to determine: we just have to check the schedules $\mathcal{S}$ to see if the times are synthesized such that the TT frame f_i is received before its deadline $f_i.deadline$. To determine the schedulability of an RC frame f_j we have to compute its worst-case end-to-end delay, from the moment it is sent to the moment it is received. We denote this worst-case delay with R_{f_j}. Section 7.2 will present a schedulability analysis technique for determining the worst-case end-to-end delay of an RC frame, which can then be compared to the deadline $f_j.deadline$ to determine if the RC frame f_j is schedulable.

Fig. 5 presents two possible solutions for synthesizing the TT schedules $\mathcal{S}$. In both cases, Fig. 5a and 5b, instead of presenting the actual schedule tables, we show a Gantt chart, which shows on a timeline from 0 to 600 μs what happens on the three dataflow links, $[ES_1, NS_1]$, $[ES_2, NS_1]$ and $[NS_1, ES_3]$. For the TT frames f_2 and f_3 the Gantt chart captures their sending times (the left edge of the rectangle) and transmission duration (the length of the rectangle).

Since the transmission of RC frames is not synchronized with the TT frames, there are many scenarios that can be depicted for f_1, depending on when f_1 is sent in relation to the schedule tables. Because we are interested in the schedulability of RC frames, for the RC frame f_1 we show in both cases (a) and (b) in Fig. 5 the worst-case scenario, i.e., the situation which has generated the largest (worst-case) end-to-end delay. The two TT frames are schedulable in both cases. In Fig. 5a the TT schedules are constructed such that the end-to-end delay of TT frames is minimized, i.e., the TT frames arrive at their destination as soon as possible. In this case, the worst-case end-to-end delay of the RC frame f_1, namely R_{f_1}, is 470 μs, which is greater than its deadline of 300 μs, hence f_1 is not schedulable. This worst-case for f_1 happens for the first frame instance $f_{1,1}$, see Fig. 5a, when $f_{1,1}$ happens to be sent by ES_1 at 105 μs. In this case, as the network implements the *timely block* integration algorithm, the frame cannot be forwarded by NS_1 to ES_3 until there is a big enough time interval to transmit the frame without disturbing the scheduled TT frames. We denote these "blocked" time intervals with hatched boxes. The first big enough interval starts only at time 500, right after $f_{2,3}$ is received by ES_3, which is too late.

However, if we instead schedule the TT frame f_3 such that its second instance $f_{3,2}$ will be sent by ES_2 to NS_1 at 350 μs, the worst case end-to-end delay for f_1 is reduced to 275, hence f_1 is schedulable. Such a solution is depicted in Fig. 5b, where we also depict the worst-case scenario for f_1.

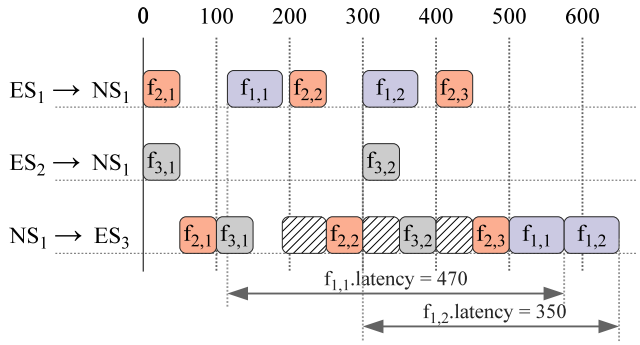


(a) Example architecture model

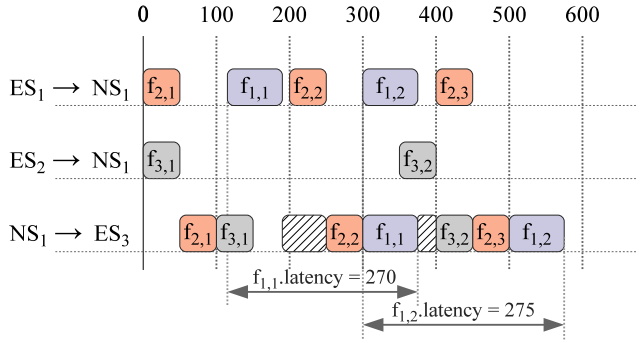
	period (μs)	deadline (μs)	C_i (μs)	$\mathcal{M}$
$f_1 \in \mathcal{F}^{RC}$	300	300	75	vl_1
$f_2 \in \mathcal{F}^{TT}$	200	200	50	vl_2
$f_3 \in \mathcal{F}^{TT}$	300	300	50	vl_3

(b) Example application model

Figure 4: Example system model



(a) Initial TT schedule



(b) Optimized TT schedule

Figure 5: Worst-case scenario for RC frame f_1

This example shows that by considering the RC traffic when scheduling the TT frames, the impact of the TT schedule on the latency of the RC frames can be greatly reduced.

7. SCHEDULE OPTIMIZATION

The problem presented in Section 4 is NP-complete [38]. In order to solve this problem, we will use the "TTEthernet Schedule Optimization" (TTESO) strategy from Fig. 7. TTESO takes as input the topology of the network $\mathcal{G}$, the set of TT and RC frames $\mathcal{F}^{TT} \cup \mathcal{F}^{RC}$ (including the size, period/rate and deadline), the set of virtual links $\mathcal{VL}$ and the mapping of frames to virtual links $\mathcal{M}$, and returns the schedules $\mathcal{S}$ for the TT frames.

Our synthesis strategy uses a tree model to represent each frame f_i. Each frame f_i is assigned to 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. A periodic frame f_i has several frame instances. We denote with $f_{i,x}$ the x^{th} instance of frame f_i, and with $f_{i,x}^{[\nu_j,\nu_k]}$ the instance sent on the dataflow link $[\nu_j,\nu_k]$. Fig. 6 presents the tree model of a frame instance $f_{1,1}$ transmitted on virtual link vl_1, from ES_1 to ES_3 and ES_4 considering the topology from Fig. 1. Naturally, frame instance $f_{1,1}$ on dataflow link $[NS_2,ES_3]$ cannot be sent before it is transmitted on $[NS_1,NS_2]$ and received in NS_2. Such a precedence constraint is captured in the model using an edge, e.g., $f_{1,1}^{[NS_1,NS_2]} \rightarrow f_{1,1}^{[NS_2,ES_3]}$.

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 $[\nu_j,\nu_k]$.
In Fig. 6, $pred(f_{1,1}^{[NS_2,ES_3]}) = \{f_{1,1}^{[ES_1,NS_1]}, f_{1,1}^{[NS_1,NS_2]}\}$. We denote

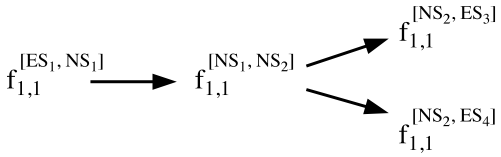


Figure 6: Representation of a frame as a tree

with $succ(f_{i,x}^{[\nu_j,\nu_k]})$ the set of successor frame instances of the frame instance $f_{i,x}^{[\nu_j,\nu_k]}$. In Fig. 6, $succ(f_{1,1}^{[NS_1,NS_2]}) = \{f_{1,1}^{[NS_2,ES_3]}, f_{1,1}^{[NS_2,ES_4]}\}$.

Our strategy has 2 steps:
(1) In the first step, we determine an initial set of TT schedules $\mathcal{S}^\circ$, line 1 in Fig. 7. The initial schedules $\mathcal{S}^\circ$ are built without using the analysis of RC traffic, and with the goal of minimizing the end-to-end response time of the TT frames. In this step we use a List Scheduling (LS) based heuristic to construct the static schedules $\mathcal{S}^\circ$. Before LS is called, we merge [27] all the trees representing the frames (which can have different periods) into a single graph covering the least common multiple of all the periods. The graph has a dummy source node to which all root nodes are connected, and a dummy sink node to which all leafs are predecessors. LS schedules this graph onto the given architecture considering the given virtual link topology. The ESes, NSes and dataflow links are considered the resources onto which the frame instances have to "execute".

(2) In the second step, we use a Tabu Search meta-heuristic (see Section 7.1) to determine 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.

7.1 Tabu Search

Tabu Search (TS) [18] is a meta-heuristic optimization, which sear- ches for that solution which minimizes the *cost function*. Tabu Search takes as input the topology of the network $\mathcal{G}$, the set of TT and RC frames $\mathcal{F}^{TT} \cup \mathcal{F}^{RC}$ (including the size, period/rate and deadline), the set of virtual links $\mathcal{VL}$ and the mapping of frames to virtual links $\mathcal{M}$, and returns at the output the best 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:

$$Cost = w_{TT} \times \delta_{TT} + w_{RC} \times \delta_{RC} \quad (1)$$

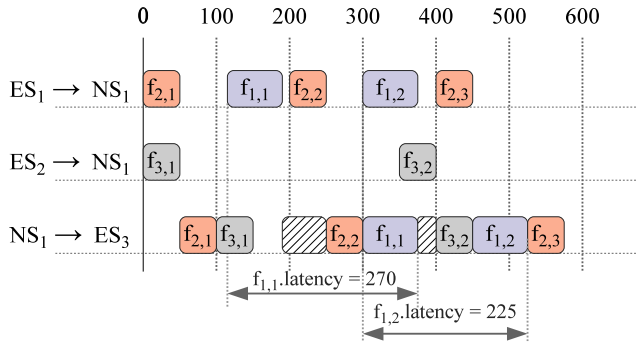
where δ_{TT} is the "degree of schedulability" for the TT frames and δ_{RC} is the degree of schedulability for the RC frames. These are summed together into a single value using the weights w_{TT} and w_{RC}, given by the designer. In case a frame is not schedulable, its corresponding weight is a very big number, i.e., a "penalty" value. 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, since we are interested to minimize the end-to-end delays for the RC frames. The degree of schedulability is calculated as:

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

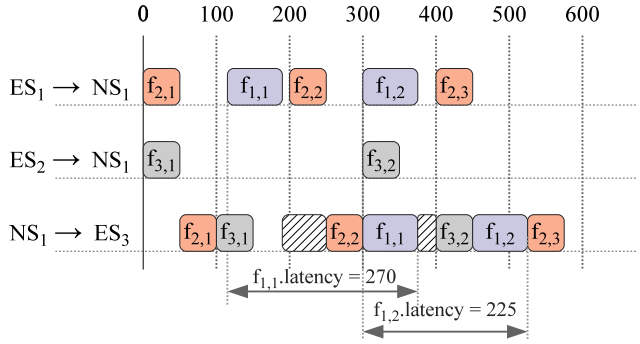
If at least one frame is not schedulable, there exists one R_{f_i} greater

TTESO($\mathcal{G}$, $\mathcal{F}^{TT} \cup \mathcal{F}^{RC}$, $\mathcal{VL}$, $\mathcal{M}$)
1 $\mathcal{S}^\circ$ = InitialSolution($\mathcal{G}$, $\mathcal{F}^{TT} \cup \mathcal{F}^{RC}$, $\mathcal{VL}$, $\mathcal{M}$)
2 $\mathcal{S}$ = TabuSearch($\mathcal{G}$, $\mathcal{F}^{TT} \cup \mathcal{F}^{RC}$, $\mathcal{VL}$, $\mathcal{M}$, $\mathcal{S}^\circ$)
3 **return** $\mathcal{S}$

Figure 7: TTEthernet Schedule Optimization strategy



(a) *Postpone* move on $f_{2,3}^{[NS_1,ES_2]}$ from Fig. 5b



(b) *Advance* move on $f_{3,2}^{[ES_2,NS_1]}$ from Fig. 8a

Figure 8: Moves for TT traffic

than the deadline $f_i.deadline$, and therefore the term c_1 will be positive. 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.

Tabu Search explores the design space by using design transformations (or "moves") applied to the current solution in order to generate neighboring solutions. As it is practically impossible to exhaustively evaluate the whole design space, in order to increase the efficiency of the Tabu Search, and to drive it intelligently towards the solution, these "moves" are not performed random, but chosen based on a *candidate list* of moves that may improve the search. Each candidate is evaluated. If the currently explored solution is better than the best known solution, it is saved as the "best-so-far" 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, we use *diversification*, i.e., we visit previously unexplored regions of the search space. In case the search diversification is unsuccessful, we *restart* the search from the best known solution.

We use four types of moves applied to TT frame instances: *advance*, *advance predecessors*, *postpone* and *postpone successors*. The *advance* move will advance the scheduled send time of a TT frame instance $f_{i,x}$ from a node ν_j on a dataflow link $[\nu_j, \nu_k]$ to an earlier moment in time. The *advance predecessors* applied to a frame instance $f_{i,x}^{[\nu_j,\nu_k]}$, will advance the scheduled send time for all its predecessors, $pred(f_{i,x}^{[\nu_j,\nu_k]})$. Similarly the *postpone* move will postpone the schedule send time of a TT frame instance from a node, while *postpone successors* will postpone the send time for all the successors of that frame instance.

The 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 received, or sent too late to meet its deadline. Also, after each move we may need to adjust the schedules (move other frame instances later or earlier) to keep the solution valid, i.e., the schedules respect the precedence and resource constraints.

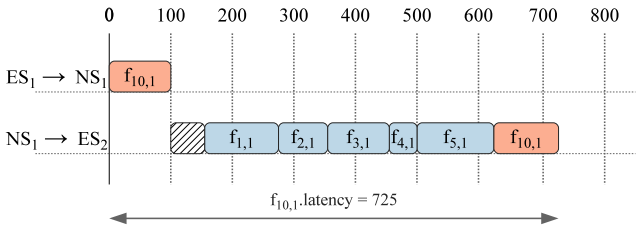
Let us illustrate these moves using the example presented in Section 6. The setup from Fig. 4 shows the architecture model in Fig. 4a and the application model in Fig. 4b. Fig. 5a presents a possible solution for synthesizing the TT schedule. In this case, the worst-case end-to-end delay R_{f_1} for the RC frame f_1 is 470 μs. Fig. 5b shows the result of a *postpone successors* move applied to the frame instance $f_{3,2}$ from Fig. 5a on dataflow link $[ES_2, NS_1]$. Consequently, frame instance $f_{3,2}^{[NS_1,ES_3]}$ is also postponed, thus creating sufficient space for $f_{1,1}$ to execute. The latency of frame instance $f_{1,2}$ can be further reduced by applying a *postpone* move to $f_{2,3}^{[NS_1,ES_3]}$ from Fig. 5b, as shown in Fig. 8a. Fig. 8b presents the result of an *advance* move applied to $f_{3,2}$ from Fig. 8a on dataflow link $[NS_1, ES_3]$, with no effect on the latencies of any of the frames involved.

For situations when there are several TT frames scheduled for transmission back-to-back on a dataflow link $[\nu_j, \nu_k]$ which may lead to large delays for RC frames, our optimization applies an *add blank* move, which adds a blank interval $bi_i^{[\nu_j,\nu_k]}$ on dataflow link $[\nu_j, \nu_k]$, which is reserved for RC traffic. Blank spaces will also be introduced by advance/postpone moves. The difference between an *add blank* and these moves is that the blank interval introduced through advance/postpone may be used by other TT frames, while the space introduced by an *add blank* move is reserved for RC frames only. In case a TT frame instance misses its deadline due to a certain blank interval $bi_i^{[\nu_j,\nu_k]}$ on dataflow link $[\nu_j, \nu_k]$, the optimization will either remove or resize the blank interval, by performing a *remove blank* move or a *resize blank* move, respectively.

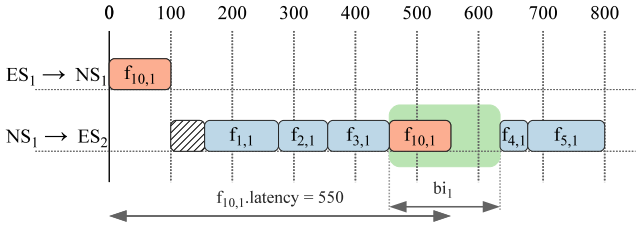
Let us consider the situation presented in Fig. 9. We assume the topology presented in Fig. 4a, and we consider dataflow links $[ES_1, NS_1]$ and $[NS_1, ES_2]$. We assume that the dataflow links have equal transmission speeds. For this example, we have one RC frame, f_{10} sent by ES_1 to ES_2, with the transmission duration $C_{f_{10}} = 100$ μs, and 5 TT frames f_1 to f_5 to be forwarded by NS_1 to ES_2. The transmission durations for the TT frames f_1 to f_5 are 100 μs, 75 μs, 100 μs, 50 μs and 125 μs, respectively. Let us consider the TT schedule presented in Fig. 9a, where the TT frames are scheduled back-to-back on dataflow link $[NS_1, ES_2]$, starting at time 150 μs. In this case, the worst-case delay for the RC frame f_{10} is 725 μs. The network implements the *timely block* approach (see Section 5.2), and we represent with a hatched box the time interval RC frame is blocked to transmit so it does not disturb the TT frames.

For situations such as the one presented in Fig. 9a, where an RC frame is blocked from transmission due to TT frames scheduled back-to-back, the *candidate list* will contain an *add blank* move to reduce the delay of the RC frame. By applying an *add blank* move to dataflow link $[NS_1, ES_2]$, as shown in Fig. 9b, the algorithm reserves a time interval of 175 μs for RC traffic, marked by a green box on the schedule. In this case, the worst-case delay for f_{10} is of only 550 μs.

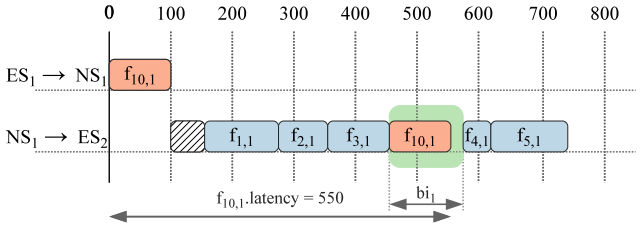
Let us assume frame f_5 has a deadline of 775 μs. In case a frame instance misses its deadline due to a blank interval $bi_i^{[\nu_j,\nu_k]}$ on dataflow link $[\nu_j, \nu_k]$, the algorithm can apply either a *remove blank* or a *resize blank* move to the blank interval. In Fig. 9c we apply a *resize blank* move on the blank interval bi_1 on dataflow link $[NS_1, ES_2]$. Thus, we resize bi_1 from 175 μs to 100 μs, which allows us to advance the scheduled send time for frames f_4 and f_5,



(a) Initial TT schedules



(b) The *add blank* move reduces the latency for $f_1 0$



(c) The *resize blank* move reduces the latency for f_4 and f_5

Figure 9: Moves for RC traffic

and consequently, allows frame f_5 to be delivered before its deadline.

7.2 RC frame end-to-end delay analysis

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

$$R_{f_i} = \sum_{\substack{\nu_j,\nu_k \in \mathcal{V} \\ [\nu_j,\nu_k] \in vl_i}} (Q_{f_i}^{[\nu_j,\nu_k]} + C_{f_1}^{[\nu_j,\nu_k]}) \quad (3)$$

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

$$Q_{f_i}^{[\nu_j,\nu_k]} = Q_{f_i,[\nu_j,\nu_k]}^{TT} + Q_{f_i,[\nu_j,\nu_k]}^{RC} + Q_{\nu_j}^{TL} \quad (4)$$

where $Q_{f_i,[\nu_j,\nu_k]}^{TT}$ is the queueing delay due to the transmission of TT frames scheduled to be sent between the moment f_i arrives at the network node ν_j and the moment the frame instance is sent, $Q_{f_i,[\nu_j,\nu_k]}^{RC}$ is the delay caused by the RC frames that can arrive, in the worst-case, before f_i at the node and thus are placed before f_i into the outgoing queue. $Q_{\nu_j}^{TL}$ is the technical latency introduced by the network node for frame f_i, due to the hardware tasks implementing the TTEthernet protocol functionality, other than the latency resulting from queueing effects.

Let us illustrate in Fig. 10 these sources of delay for an RC frame at a network node considering the topology example presented in

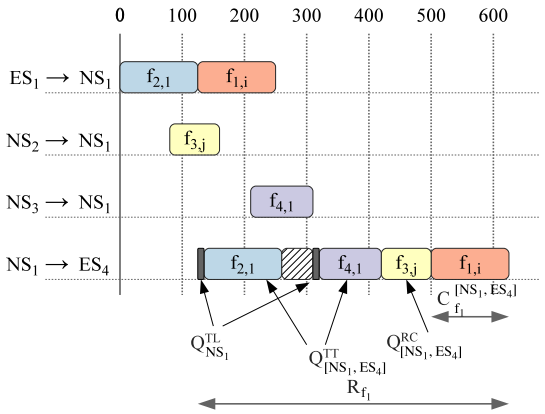


Figure 10: Worst-case end-to-end analysis for frame f_1

Fig. 2. There are 4 frames, with frames f_1 and f_2 sent from ES_1 to ES_2 and frames f_3 and f_4 forwarded by NS_2 and NS_3, respectively, to ES_2, with $f_1, f_3 \in \mathcal{F}^{RC}$ and $f_2, f_4 \in \mathcal{F}^{TT}$. All the dataflow links have the same speed, hence the transmission duration for the frames are $C_1 = 120$ μs for f_1, $C_2 = 125$ μs for f_2, $C_3 = 80$ μs for f_3 and $C_4 = 100$ μs for f_4. The RC frame under analysis is $f_{1,1}$. We consider the technical latency introduced by NS_1 to be $Q_{NS_1}^{TL} = 5$ μs. The network implements the timely block approach.

Fig. 10 presents the worst-case scenario for frame f_1, i.e., the case in which the end-to-end response time R_{f_1} is the largest. This happens for the frame instance $f_{1,i}$, which is delayed by frame instance $f_{2,1}$, $f_{4,1}$ and $f_{3,j}$. Thus, the TT frames $f_{2,1}$ and $f_{4,1}$ are scheduled for transmission on dataflow link $[NS_1, ES_4]$ at 130 μs and 310 μs, respectively, according to the TT schedules determined at design time. In the worst-case scenario, frame $f_{1,i}$ arrives at NS_1 at time moment 250 μs. Note that the RC frames are not synchronized with the TT schedules, so they can arrive at any time. The network implements the timely block algorithm, hence frame $f_{1,i}$ cannot be dispatched as soon as it arrives at NS_1, as it would interfere with the transmission of the scheduled TT frame $f_{4,1}$. We marked this blocking time of 60 μs in Fig. 10 with an hatched box. In this case, $Q_{f_1,[NS_1,ES_4]}^{TT} = 265$ μs, and it includes the blocking time.

In the worst-case scenario for $f_{1,i}$, the RC frame instance $f_{3,j}$ arrives at NS_1 before $f_{1,i}$, hence, $f_{3,j}$ will be sent to ES_4 before $f_{1,i}$. Consequently, $Q_{f_1,[NS_1,ES_4]}^{RC} = 80$ μs. The worst-case queueing delay for $f_{1,i}$ in NS_1, using Eq. 4, is $Q_{f_1}^{[NS_1,ES_4]} = Q_{f_1,[NS_1,ES_4]}^{TT} + Q_{f_1,[NS_1,ES_4]}^{RC} = 265 + 80 = 345$ (μs). Thus, we can compute the worst-case end-to-end delay for $f_{1,i}$ using Eq. 3 as $R_{f_1} = C_{f_1}^{[ES_1,NS_1]} + Q_{f_1}^{[NS_1,ES_4]} + C_{f_1}^{[ES_1,NS_1]} = 120 + 345 + 120 = 485$ (μs).

Researchers have proposed several worst case end-to-end delay analyses for the traffic in an ARINC 664p7 network, including analyses based on Network Calculus [14, 13], Finite State Machine [32], Timed Automata [6] or Trajectory Approach [11, 12]. However, none of these analysis methods are applicable to TTEthernet, since they do not consider the impact of TT messages on the schedulability of RC messages. In this paper we use the TTEthernet analysis from [35], which shows how to consider TT messages. In our future work, we plan to extend the Trajectory Approach [12] to consider TT messages, with the aim to reduce the pessimism of the analysis from [35]. However, note that the analysis used for RC frames is orthogonal to our optimization problem.

8. EXPERIMENTAL EVALUATION

Set	Test case	ES	NS	Messages	Frame instances	Load [%]	Δ_{cost} [%]
1	11	13	4	80	12593	50	2.58
	12	25	6	88	1787		24.44
	13	35	8	103	2285		20.06
	14	45	10	165	3299		11.90
2	21	11	4	115	16904	70	9.17
	22	25	6	179	2523		20.61
	23	35	8	154	3698		39.34
3	31	25	6	76	1387	40	37.97
	32			88	1787	50	24.44
	33			115	2503	60	40.47
	34			179	2523	70	20.61
	35			155	2960	80	32.10
4	41	35	8	65	1976	40	38.75
	42			103	2285	50	20.06
	43			89	2801	60	12.73
	44			176	3856	70	12.75
	45			135	3490	80	20.23
5	automotive	15	3	170	38305	80	50.88

Table 1: Experimental results

For the evaluation of our proposed optimization approach, "TTEthernet Schedules Optimization" (TTESO), we used 17 synthetic benchmarks and one real-life case study. The TTESO algorithm was implemented in Java (JDK 1.6), running on SunFire v440 computers with UltraSPARC IIIi CPUs at 1.062 GHz and 8 GB of RAM.

The results are presented in Table 1. For the synthetic benchmarks, we have used 6 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 3–7, we have the details of each benchmark, the number of ESes, NSes, number of messages, the number of frame instances and the load on the network, respectively. The load within an application cycle T_{cycle} is calculated as the ratio of the sum of the sizes of all frame instances divided by the network speed (in our case 100 Mbps).

For all experiments, we have compared TTESO with a baseline solution, namely the Straightforward Solution (SS), which builds the TT schedules with the goal of minimizing the end-to-end response time of the TT frames without using the analysis of RC traffic. The comparison between SS and TTESO, Δ_{cost}, is shown in the last column in the table as a percentage improvement of TTESO over SS, in terms of the cost function (Eq. 1).

In the first two sets of experiments, labeled "Set 1" and "Set 2" in Table 1, we were interested to evaluate the quality of the results obtained with TTESO as the size of the system increases. Thus, we have used 7 synthetic benchmarks, with the number of network nodes ranging between 16 and 55 nodes. The first set of 4 benchmarks have a load of 50%, and the second set of benchmarks have a load of 70%. As we can see, TTESO is able to significantly improve the cost function over SS, even as the size of the system increases. We used a time limit of 45 minutes for the first set and 90 minutes for the second set.

In the third sets of experiments, labeled "Set 3" were interested on how TTESO performs as the load of the network increases from 40% to 80%. As we can see, TTESO is able to significantly improve on the the solution provided by SS. These results were obtained using a time limit of 30, 45, 70, 90 and 120 minutes for the test cases with a load of 40%, 50%, 60%, 70% and 80%, respectively. A similar evaluation was performed in the case of experimental "Set 4", with the difference that we considered a larger architecture.

Finally, we used one real-life benchmark derived from [24], based on the SAE automotive communication benchmark [1]. In this benchmark we have 18 network nodes (ESes and NSes), and 83 frames (with the parameters generated based on the messages presented in [24]). Table 1 contains the results for this benchmark—the last line labeled with "Set 5". The results obtained for the real-life benchmark confirms the results of the synthetic benchmarks.

9. CONCLUSIONS

In this paper we have addressed the optimization of the TTEthernet protocol. 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, 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 frames and the topology of the virtual links to which they are assigned, we have proposed a Tabu Search optimization strategy for the synthesis of the TT schedules. The synthesis is performed such that the frames are schedulable, and the degree of schedulability is improved. The results on several synthetic benchmarks and a real-life case study show that through the careful optimization of TT static schedules, significant improvements can be obtained.

10. 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.

11. REFERENCES

[1] SAE Technical Report J2056/1. Technical report, SAE International.

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

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

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

[5] *AS6802: Time-Triggered Ethernet*. SAE International, 2011.

[6] M. Adnan, J.-L. Scharbarg, J. Ermont, and C. Fraboul. Model for worst case delay analysis of an AFDX network using timed automata. In *Proceedings of the Conference on Emerging Technologies and Factory Automation*, pages 1 –4, 2010.

[7] N. Audsley, K. Tindell, and A. Burns. The end of the line for static cyclic scheduling. In *Proceedings of Euromicro Workshop on Real-Time Systems*, pages 36–41, 1993.

[8] S. Baruah and G. Fohler. Certification-Cognizant Time-Triggered Scheduling of Mixed-Criticality Systems. *Proceedings of the Real-Time Systems Symposium*, pages 3–12, 2011.

[9] S. Baruah, H. Li, and L. Stougie. Towards the design of certifiable mixed-criticality systems. In *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*, 2010.

[10] S. K. Baruah, A. Burns, and R. I. Davis. Response-Time Analysis for Mixed Criticality Systems. *Proceedings of the Real-Time Systems Symposium*, pages 34–43, 2011.

[11] H. Bauer, J.-L. Scharbarg, and C. Fraboul. Applying and optimizing trajectory approach for performance evaluation of AFDX avionics network. In *Proceedings of the Conference on Emerging Technologies Factory Automation*, pages 1 –8, 2009.

[12] H. Bauer, J.-L. Scharbarg, and C. Fraboul. Worst-case end-to-end delay analysis of an avionics AFDX network. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1220–1224, 2010.

[13] A. Bouillard, L. Jouhet, and E. Thierry. Tight Performance Bounds in the Worst-Case Analysis of Feed-Forward Networks. In *Proceedings of INFOCOM*, pages 1–9, 2010.

[14] M. Boyer and C. Fraboul. Tightening end to end delay upper bound for AFDX network calculus with rate latency FIFO servers using network calculus. In *Proceedings of the International Workshop on Factory Communication Systems*, pages 11 –20, 2008.

[15] R. Ernst. Certification of Trusted MPSoC Platforms. 10th International Forum on Embedded MPSoC and Multicore, 2010.

[16] R. N. et al. Steiner tree based distributed multicast routing in networks. In X. Cheng and D.-Z. Du, editors, *Steiner Trees in Industry*. Springer, 2002.

[17] E. N. Gilbert and H. O. Pollak. Steiner minimal trees. *SIAM Journal on Applied Mathematics*, 16(1):1–29, 1968.

[18] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[19] K. Hoyme and K. Driscoll. SAFEbus. *IEEE Aerospace Electronic Systems Magazine*, 8:34–39, 1993.

[20] V. Izosimov, P. Pop, P. Eles, and Z. Peng. Scheduling of fault-tolerant embedded systems with soft and hard timing constraints. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 915–920, 2008.

[21] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Springer, 2011.

[22] B. Leiner, M. Schlager, R. Obermaisser, and B. Huber. A Comparison of Partitioning Operating Systems for Integrated Systems. *Computer Safety, Reliability, and Security*, pages 342–355, 2007.

[23] H. Lonn and J. Axelsson. A comparison of fixed-priority and static cyclic scheduling for distributed automotive control applications. In *Proceedings of the Euromicro Conference on Real-Time Systems*, pages 142–149. IEEE, 1999.

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

[25] R. Obermaisser. *Time-Triggered Communication*. CRC Press, Inc., 2011.

[26] P. Pop, P. Eles, and Z. Peng. Scheduling with optimized communication for time-triggered embedded systems. In *Proceedings of the International Workshop on Hardware/Software Codesign*, pages 178–182, 1999.

[27] P. Pop, P. Eles, and Z. Peng. *Analysis and Synthesis of Communication-Intensive Heterogenous Real-Time Systems*. Kluwer Academic Publishers, 2004.

[28] P. Pop, P. Eles, and Z. Peng. Schedulability-driven frame packing for multicluster distributed embedded systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 4(1):112–140, 2005.

[29] T. Pop, P. Pop, P. Eles, and Z. Peng. Analysis and Optimisation of Hierarchically Scheduled Multiprocessor Embedded Systems. *International Journal of Parallel Programming*, 36(1):37–67, feb 2008.

[30] T. Pop, P. Pop, P. Eles, Z. Peng, and A. Andrei. Timing analysis of the FlexRay communication protocol. In *Proceedings of the Euromicro Conference on Real-Time Systems*, pages 11 pp. –216, 2006.

[31] J. Rushby. Partitioning for Avionics Architectures: Requirements, Mechanisms, and Assurance. NASA Contractor Report CR-1999-209347, NASA Langley Research Center, 1999.

[32] I. Saha and S. Roy. A Finite State Modeling of AFDX Frame Management Using Spin. In L. Brim, B. Haverkort, M. Leucker, and J. van de Pol, editors, *Formal Methods: Applications and Technology*, volume 4346 of *Lecture Notes in Computer Science*, pages 227–243. Springer, 2007.

[33] P. K. Saraswat, P. Pop, and J. Madsen. Task Mapping and Bandwidth Reservation for Mixed Hard/Soft Fault-Tolerant Embedded Systems. *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*, pages 89–98, 2010.

[34] W. Steiner. An Evaluation of SMT-based Schedule Synthesis For Time-Triggered Multi-Hop Networks. In *Proceedings of the Real-Time Systems Symposium*, pages 375–384, 2010.

[35] W. Steiner. 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*, pages 11 –18, 2011.

[36] W. Steiner, G. Bauer, B. Hall, M. Paulitsch, and S. Varadarajan. TTEthernet Dataflow Concept. In *Proceedings of the International Symposium on Network Computing and Applications*, pages 319 –322, 2009.

[37] D. Tămaş-Selicean and P. Pop. Design Optimization of Mixed-Criticality Real-Time Applications on Cost-Constrained Partitioned Architectures. In *Proceedings of the Real-Time Systems Symposium*, pages 24–33, 2011.

[38] J. D. Ullman. NP-complete scheduling problems. *J. Comput. Syst. Sci.*, 10(3):384–393, 1975.

[39] J. Xu and D. L. Parnas. On Satisfying Timing Constraints in Hard-Real-Time Systems. *IEEE Transactions on Software Engineer*, 19(1):70–84, 1993.