A Quantitative Analysis of Interfaces to Time-Triggered Communication Buses

Raimund Kirner, *Senior Member, IEEE*, and Peter Puschner

***Abstract*— Nodes connected to a time-triggered (TT) network can access the network interface in two different ways, synchronously or asynchronously, which greatly impacts communication timing and message lifespans (i.e., the time from writing a message to its send buffer till the time when the message is read by the receiver). In this paper we present a clear timing model to reason about the timing variation possible with TT interfaces. This model facilitates the quantitative analysis of the message lifespans of synchronous and asynchronous TT interfaces. Further, we develop a tool to search for node and network configurations that minimise or maximise message lifespans. We show that choosing the right configuration for synchronous interface access can reduce message lifespan significantly (we observed a factor of 9 even for small scenarios). While industrial practice typically is to choose a slot allocation a priory, we show that optimising the slot allocation in coordination with task scheduling gives an extra edge in obtaining minimal message lifespans. For nodes with synchronous interface access, the tool determines the parameters needed to obtain minimal message lifespan and jitter.**

***Index Terms*— Real-time systems, networks, time-triggered communication.**

I. Introduction

TIME-TRIGGERED networks provide a communication service for dependable real-time systems. This communication service offers time predictability at its interfaces, as message transmission is under full control of the TT network with its global clock and pre-defined transmission schedule. The message transmission rate as well as send and receive times of all messages are clearly determined before runtime. As a consequence, events observed by communicating nodes at runtime have no influence on the points in time and rate at which messages are sent or received, and sending nodes cannot exercise any control pressure on and across the communication network, which protects both the network and the receiving nodes from load changes or even overload and allows system designers to give guarantees for message-transmission deadlines.

While the TT communication network delivers its communication service with precisely defined timing, the degree to which applications leverage the timing properties of the network depends on how the nodes connected to the TT network configure and access the network interface. We distinguish two different strategies that nodes use to access the TT communication interface, asynchronous and synchronous access [1]. Both access strategies allow for the construction of composable and time-predictable real-time applications. However, the timing parameters and the response-time guarantees achievable with each of the two access strategies differ significantly.

In this article we therefore investigate into the timing of applications that use the two different strategies of interfacing to TT systems. To this end, we extend the work on asynchronous and synchronous TT interfaces [1]. We propose a precise timing model that allows us to analyse the *lifespan* of messages (i.e., the time interval between a message write operation by a sending task and the message read operation by its receiving task) in TT communication and evaluate the effect of the TT network-interfacing strategies on message lifespans. The contributions of this article are:

- Development of a timing model to reason about the timing variation possible with TT interfaces. This model facilitates the quantitative analysis of the message lifespans of synchronous and asynchronous TT interfaces.
- Implementation of the timing model in a tool in order either to mimimise the maximum message lifespan in the system or to minimize the sum of the lifespans of all messages in the system.
- We explain how to use this timing model to analyse the possible lifespan values in systems with asynchronous TT interfaces.
- We explain how to use this timing model to configure systems with synchronous TT interfaces in order to obtain a minimised message lifespan with a jitter limited to the precision of clock synchronisation.
- We provide a detailed quantitative analysis to show that the slot allocation on the TT bus can significantly influence the achievable message-lifespan minimum. While industrial practice is to allocate TT slots a-priori, our results indicate that optimising the TT slot allocation together with local task activation times is crucial to arrive at minimal message lifespans.

While research has been done on optimising the bus timing, no research has been done on the quantitative effects of TT interfaces.

The article is structured as follows: Section II presents an overview of related work. The foundations of time-triggered communication systems are reviewed in Section III. The timing model for TT interfaces is described in Section IV. Section V presents an implementation of the timing model used to optimise the message lifespan (i.e., either the

Manuscript received September 23, 2020; revised February 12, 2021 and March 25, 2021; accepted March 26, 2021; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor C. F. Chiasserini. *(Corresponding author: Raimund Kirner.)*

Raimund Kirner was with the Department of Computer Science, University of Hertfordshire, Hatfield AL10 9AB, U.K. (e-mail: r.kirner@herts.ac.uk).

Peter Puschner was with the Faculty of Computer Science, TU Wien, 1040 Vienna, Austria (e-mail: peter@vmars.tuwien.ac.at).

Digital Object Identifier 10.1109/TNET.2021.3073460

maximum or the sum of all message lifespans). Experiments exemplify the quantitative effect of message lifespans. Finally, Section VI concludes this article.

II. Related Work

Time-triggered message communication is highly attractive for real-time communication networks, as message transfer is pre-planned and therefore temporally predictable. The most prominent protocol for time-triggered message communication is the *Time-Triggered Protocol (TTP)* [2]. Besides time-predictable, cyclic message transport, TTP provides support for precise clock synchronization, redundant message transmission, error detection, and clique avoidance that make TTP the preferred choice for safety-critical real-time applications [3]. The TTEthernet protocol [4] makes time-triggered communication available to the world of switched Ethernet. TTEthernet uses standard Ethernet messages with a dedicated TTEthernet message type. The protocol integrates the real-time features of TTP with Ethernet compatibility, i.e., by means of TTEthernet switches time-triggered Ethernet messages and standard event-triggered Ethernet traffic can be merged on the same network. Another variant of a TT communication bus is FlexRay, which is used in the automotive domain [5].

TT protocols provide an infrastructure for time-predictable communication, but do not dictate when applications must access the network interface. I.e., applications may operate in synchrony or asynchronously to the operation of the TT communication protocol. Giotto [6] is a framework for designing and validating applications according to the paradigm of synchronous operation, i.e., all computation steps are clearly defined together with the synchronous points in time when communication takes place and state changes occur. Applications developed with Giotto can be compiled to utilize the synchronous operation of TT communication to maintain the synchrony of tasks and communication in the final application. This makes the whole applications time-predictable and time-repeatable.

TT communication has been also used inside silicon chips, i.e., in so-called network-on-chips [7]. Even more, TT communication has not only been used with wired communication, but also in wireless communication, e.g., with ZigBee by Koubaa *et al.* [8] and Ahmad and Hanzalek [9].

Network behaviour simulation is an established topic, e.g. the simulation framework `OMNeT++` is often used [10]. `OMNeT++` is a framework to build simulators; network protocol models have to be provided or developed separately. For example, for communication networks with event-based semantics, queuing theory [11] can be used to model the temporal behaviour of the network. The contribution of this article is the development of a behaviour model and analysis methods for TT networks.

The asynchronous and synchronous TT interfaces have been described by Puschner and Kirner [1]. The authors provided a bound for the jitter of the message lifespan, i.e., the time between writing a message by the sender task and reading the message by the receiver task.

For asynchronous TT interfaces there had been interface access protocols proposed that aim to avoid concurrency-based access delays. Kopetz and Reisinger proposed the original non-blocking write protocol, which guaranteed that the writer has no delay, only readers might have additional delay in case of an access contention [12]. Puschner and Frömel converted the original NBW protocol into a single-path variant, for which also the readers have no execution-time jitter but instead wait for the worst-case delay [13]. Puschner and Kirner proposed the *Rate-bounded Non-Blocking Communication Protocol* (RNBC), which has less execution time overhead than NBW and at the same time guarantees that both writer and reader are free of concurrency-based execution-time jitter [1].

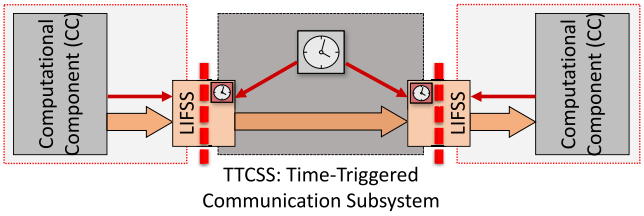

Fig. 1. Time-triggered system model.

There has been also work done in building optimised schedules for systems with TT networks. Pop *et al.* did an optimisation of slot allocations and task offsets for the TTP [2] protocol [14]. Their experiments compare two heuristics against a reference optimisation based on simulated annealing. It is not described in detail what optimisation criteria is used in their work to denote the 'best schedule'. Craciunas *et al.* described an optimisation framework for systems with TT networks using integer-linear programming [15]. The authors assume a given slot allocation on the TT bus and search for task offsets to yield an optimal scheduling table for TTEthernet [16]. Minaeva *et al.* describe the search of task offsets for TT systems with hyper periods [17]. The authors focus on finding a feasible schedule with limited jitter within a hyper period, in contrast to the typical zero-jitter scheduling.

The described work on searching TT schedules does not cover the quantitative analysis of the TT communication interface. None of the research focuses studies the effect of using a given slot allocation versus searching for a minimal one. Also none of the research analyses the difference between synchronous TT interfaces and asynchronous TT interfaces. Besides showing the effect of slot allocation and task alignment, in our work we also show that there are different optimisation criteria possible when optimising the message lifespans of TT buses.

III. Time Triggered Communication

Following the time-triggered approach, a distributed real-time computer system consists of the *time-triggered communication system (TTCS)* and the *computational components (CCs)* or *nodes* that exchange data via the TTCS. CCs connect to the TTCS via the *linking interface subsystem (LIFSS)*, see Figure 1.

The TTCS is an autonomous subsystem of the distributed real-time computer system that transports messages in a time-predictable way. It transfers time-triggered state messages from the LIFSS interfaces of the sending nodes to one or more

receiver nodes of the distributed computer system [18]. Typically, the TTCS transports messages periodically, according to a static message-transmission schedule that is constructed at design time.

The clock synchronization service of the TTCS provides a global clock to the distributed system. Communication end points of the TTCS, the LIFSS of the nodes, use this global clock to maintain a uniform view about the progress of time and to coordinate their message send and receive operations according to the message schedule. Further, the TTCS offers access to the virtual global clock and a clock-interrupt service. The computational components of the nodes can use these services to synchronize their operation to the progression of global time. (see e.g., [19]).

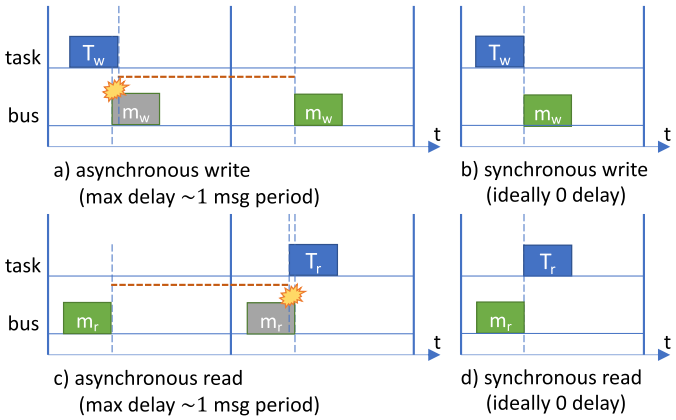

Fig. 2. Data delay in asynchronous and synchronous TT interfaces.

A. Synchronous and Asynchronous Interfaces

Time-triggered communication allows components of multi-component systems to autonomously control their timing behavior [20]. Reading from or writing to the LIFSS of a component is similar to reading or writing volatile program variables that are periodically updated. In contrast to event-triggered interfaces, where every received message has to be read and consumed in order to keep the receiving component in a consistent state (i.e., the receiver must read/process every message before its validity time expires), time-triggered communication does not impose such a control pressure on components that read interface data from the LIFSS [21].

Depending on whether computing components use only the data-sharing semantics of the LIFSS or also utilize its global notion of time to synchronize their actions to the progression of time and message send and receive times, we distinguish two interfacing strategies at the LIFSS [1]:

In the first approach, computational components access the LIFSS *asynchronously*, and do not take the timing of the TTCS into consideration. This has the advantage that the respective components maintain full control over the pace of their activities. Even mutual-exclusion access conflicts are masked and do not impact the temporal control of the component. On the other hand, this loose temporal coupling between the activities of the CCs and the TTCS prolongates message lifespans, i.e., the intervals between the creation and the consumption of messages by tasks on the CC.

In the second approach, computing components access the LIFSS *in synchrony* with the TTCS. I.e., components are aware of global time and the timing of the message transport service of the TTCS wrt. global time. They synchronize to the global time of the communication system (thus accepting some restriction of their temporal autonomy) and take advantage of the statically available information about the send and receive times of messages to optimize response times (i.e., minimize message lifespan) and avoid control conflicts when accessing the LIFSS. Further, synchronizing component activities with global time yields a high degree of overall temporal predictability and facilitates the realization of replica determinism and therefore fault-tolerance mechanisms in distributed real-time applications [1].

Figure 2 visualises different message-lifespan scenarios of asynchronous and synchronous TT interfaces. Figure 2.a and Figure 2.c show that without optimised alignment of read/write of tasks with the CC can cause data delays for both read and write of one message period, resulting in a total message lifespan of two message periods plus one communication slot lengths. Figure 2.b and Figure 2.d show that with synchronised TT interfaces the read/write delay can be potentially zero, resulting in a total message lifespan of just one communication slot length. As shown later, the minimal lifespan of multiple messages can be significantly higher than zero due to their relationships with tasks.

In the rest of the paper we will investigate in further detail how the two LIFSS access strategies influence message lifespan, i.e., what minimal and maximum message lifespans can occur. We will also show, that synchronizing the activities of computational components to the message transport of the TTCS leads to minimum message lifespans, thus facilitating the construction of real-time systems with short response times.

IV. Timing Model for Message Lifespan in Time-Triggered Communication

In this section we introduce a formal timing model for the TTCS. While TTCSs focus on periodic and predictable communication, these systems can also support multi-rate communication by constructing a hyper-period [22] that is the least common multiple of all the message periods in the system. In this article we focus on systems with all message periods being equal, i.e., the hyper-period is equal to the period of the messages.

A TTCS can be expressed as the following tuple:

$$TTCS = \langle T_{tt}, t_s, M, B \rangle$$

T_{tt} …length of a TT communication round (also known as communication period or TDMA round).

t_s …length of one TT communication slot within T_{tt}. We enumerate the slot numbers in a T_{tt} starting with 0.

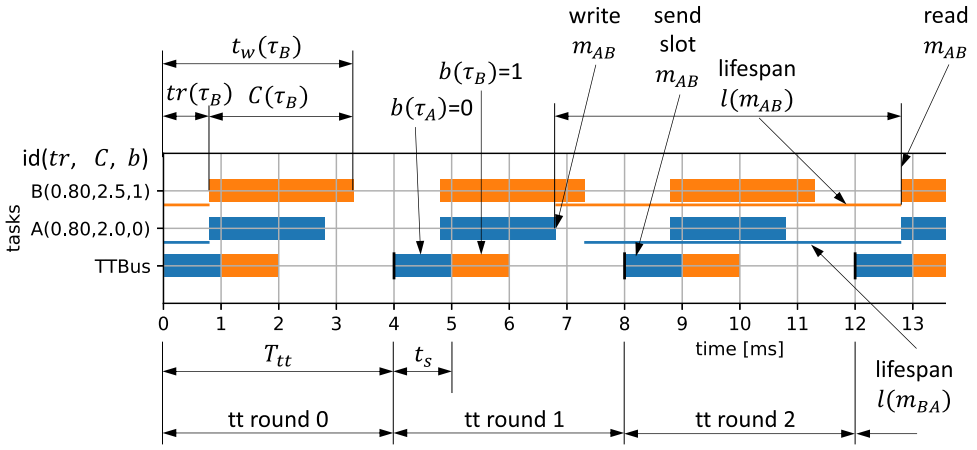

Fig. 3. Timing model for TT communication.

M …set of TT messages sent via the TTCS. Assuming only one message is sent per TT slot, it must hold that: $|M| \cdot t_s \leq T_{tt}$.

B …bus configuration is a mapping of the form $M \Rightarrow SNum$ with $SNum = [0, \frac{T_{tt}}{t_s} - 1]$, i.e., each message is assigned to a TT slot number. $B(m)$ is the TT slot number of a message m.

It shall be noted that this model allows for the size of the set of TT messages $|M|$ to be less than the number of available TT slots. In this case the remaining TT slots could be used for other forms of communication, like event-based communication. E.g., the FlexRay protocol uses such a hybrid communication system [23].

Each real-time task $\tau \in T$ running at each TT node can be described as a tuple:

$$\tau = \langle tr, C, M_R, m_W \rangle$$

T …set of all tasks

tr …the trigger time of the task relative to the beginning of a TT round. This parameter is only specified when using a synchronous TT interface.

C …the WCET of the task

M_R …the set of TT messages this task reads: $M_R \subseteq M$

m_W …the TT message this task writes: $m_W \in M$.

For simplification, we use only one output message per task. This is also the typical use case of a TT system, as the output is written at the end of a task's execution, which then can be read by all other nodes connected to the TTCS. To formulate the timing model of the TTCS, we use the following additional notation:

$C(\tau)$ …WCET of task τ

$b(\tau)$ …relative TT slot number within a TT communication round for task τ to send a message

$tr(\tau)$ …relative start time of task τ (relative to begin of a TT round)

$t_w(\tau)$ …writing time of output message of task τ relative to begin of a TT round: $t_w(\tau) = tr(\tau) + C(\tau)$ (for simplicity we assume the execution time of τ is the WCET of τ)

$S(m)$ …sender task of message m

$R(m)$ …set of receiver tasks of message m

To give a further intuition to the above definitions, Figure 3 shows a simple TT system with two tasks τ_A, τ_B. The message sent from τ_A to τ_B is denoted as m_{AB}. The length of a TT communication round T_{tt} is 4ms, consisting of 4 TT slots of length $t_s = 1ms$. The two top rows show the execution of tasks τ_A (blue) and τ_B (orange). The bottom row shows the communication timing of the TTCS, with the slot color matching the color of the writer task, e.g., in TT slot 0 (blue) the message of task τ_A is sent.

Based on the introduced notation we calculate the lifespan $l(m)$ of a message m, which is the time from the termination of the sender task $S(m)$ (writing the message to the LIFSS) till the start of receiver task $R(m)$ (reading the message from the LIFSS; when $R(m)$ contains more than one task we get different lifespans for the receivers). We have to take into account that the writing, communicating, and reading of a message can take place in different TT rounds, which is why we use this extra notation:

$r^S(m)$ …number of TT round in which message m is transmitted by the TTCS relative to the TT round it was written (e.g., $r^S(m) = 1$ if the message is transmitted in the TT round after it was written)

$r^R(m)$ …number of TT round where message m is read by the receiver task relative to the TT round it was written

$l(m)$ …lifespan of message m, i.e., the time span between its write by task $S(m)$ and its read by task $R(m)$.

To calculate the lifespan $l(m)$ of a message m we use the following abbreviations: $\tau_S{=}S(m)$, $\tau_R{\in}R(m)$. Note that $R(m)$ can contain more than one task, in which case we calculate the message lifespan separately for each case. In addition, Equation 1 introduces the notation of a ceiling function based on arbitrary multiples (y) rather than 1:

$$\lceil x \rceil^y = \lceil \frac{x}{y} \rceil \cdot y \tag{1}$$

Based on that, the communication round $r^S(m)$ can be calculated using the following minimisation problem:

$$r^S(m) = k \mid \min_k \begin{pmatrix} \lceil t_w(\tau_S) \rceil^{ts} \leq \\ k \cdot T_{tt} + b(\tau_S) \cdot ts \end{pmatrix} \tag{2}$$

In simple terms, $r^S(m)$ is the minimum number of multiples of T_{tt} that have to be added to the relative beginning of message m's communication slot ($b(\tau_S)$) to be equal or more than the write time of m ($t_w(m)$). Referring to the example in Figure 3, $r^S(m_{AB})$ is 1, see the labels "write m_{AB}" and "send m_{AB}" which are one communication round apart.

Based on $r^S(m)$, $r^R(m)$ can be calculated using another minimisation problem:

$$r^R(m) = k \mid \min_k \begin{pmatrix} r^S(m) \cdot T_{tt} + (b(\tau_S)+1) \cdot t_s \leq \\ k \cdot T_{tt} + tr(\tau_R) \end{pmatrix} \quad (3)$$

In other words, $r^R(m)$ is the minimum number of multiples of T_{tt} that have to be added to the relative beginning of message m's read time ($tr(\tau_R)$) to be equal or more than the time for the completed communication of m ($r^S(m) \cdot T_{tt} + (b(\tau_S) + 1) \cdot t_s$). Referring to the example in Figure 3, $r^R(m_{AB})$ is 2, see the labels "write m_{AB}" and "read m_{AB}" which are one communication round apart.

Based on $r^R(m)$ the lifespan $l(m)$ of a message m can be calculated as given in Equation 4:

$$l(m) = r^R(m) \cdot T_{tt} + tr(\tau_R) - t_w(\tau_S) \quad (4)$$

The above equation calculates the difference between message m's write time ($t_w(\tau_S)$) by task $S(m)$ and m's read time ($r^R(m) \cdot T_{tt} + tr(\tau_R)$) by task $R(m)$. Going back to the example of Figure 3, we can see that the lifespan $l(m_{AB})$ of message m_{AB} is 6.0ms, which is 1.5 times the length of the TT communication round T_{tt}.

A. Timing of Asynchronous TT Interfaces

In our timing model we have introduced the real-time tasks to be tuples of the following form $\tau = \langle tr, c, M_R, m_W \rangle$. However, in the case of asynchronous TT interfaces the value of tr is not specified, since the execution of tasks inside nodes and the communication timing of the TTCS are asynchronous. Thus, tr can be any value within the range $0 \leq tr \leq T_{tt}$. The case $tr > T_{tt}$ is irrelevant, as due to the periodic nature of the TTSC, this case is equal to the case (tr modulo T_{tt}). For the asynchronous TT interface, the lifespan of a message m is therefore not a single value, but may vary depending on the values tr of writer and reader task.

To get the minimum and maximum of the lifespan $l(m)$ of a message m for the asynchronous TT interface with $\tau_S{=}S(m)$ and any $\tau_R{\in}R(m)$ we need to search over the value range of both $tr(\tau_S)$ and $tr(\tau_R)$:

$$\begin{aligned} l^{min}_{async}(m) &= \min_{tr(\tau_S), tr(\tau_R)} l(m) \\ l^{max}_{async}(m) &= \max_{tr(\tau_S), tr(\tau_R)} l(m) \end{aligned} \quad (5)$$

Since the executions of all tasks are not only asynchronous to the TTCS, but also against each other, the variations of the lifespan per message add up to get the variation of the lifespan of all messages.

$$\begin{aligned} l^{min}_{async}(M) &= \sum_{m \in M} l^{min}_{async}(m) \\ l^{max}_{async}(M) &= \sum_{m \in M} l^{max}_{async}(m) \end{aligned} \quad (6)$$

The message-validity jitter for asynchronous TT interfaces is for each message theoretically maximal 2x the TT communication round T_{tt} [1]. Equation 6 calculates the total message-validity jitter specifically for a given system.

B. Timing of Synchronous TT Interfaces

With *synchronous* TT interfaces we have the possibility to choose the start time tr of each task relative to the beginning of a TT communication round. This allows us to optimise the timing of a TT system. We introduce two optimisation methods. The first method is to find a configuration of relative task start times such that the sum of the lifespans of all messages is minimized. The second method aims at minimising the maximum lifespan among all messages.

Equation 7 shows the optimisation of synchronous TT interfaces for the average lifespan of all messages $m{\in}M$. The resulting TR_{avg} consists of task tuples $\tau{\in}T$ and their optimised tr value (τ, tr_{min}). Optimising for the average lifespan is equivalent to optimising for the sum of lifespans.

$$\begin{aligned} TR_{avg} = \{(\tau, tr_{min}) \mid \tau{\in}T \wedge tr_{min}{=}tr(\tau)\} \\ \mid \min_{tr(\tau \in T)} \left(\sum_{m \in M} l(m) \right) \end{aligned} \quad (7)$$

The optimisation of synchronous TT interfaces for the maximum lifespan among all messages $m{\in}M$ is given in Equation 8. The resulting TR_{max} consists of tuples of tasks $\tau{\in}T$ and their optimised tr value.

$$\begin{aligned} TR_{max} = \{(\tau, tr_{min}) \mid \tau{\in}T \wedge tr_{min}{=}tr(\tau)\} \\ \mid \min_{tr(\tau \in T)} (\max(\{l(m) \mid m{\in}M\})) \end{aligned} \quad (8)$$

The preferable optimisation method depends on the application requirements. Synchronous TT interfaces are meant for hard real-time systems, where the minimisation of the maximum lifespan of messages is usually central.

1) Logical Execution Time (LET): The optimisation models given in Equation 7 and Equation 8 allow to configure the system in a way that the lifespan jitter of TT messages is minimal. Basically, the lifespan jitter could be reduced to the precision of the clock synchronisation in the distributed TT system. However, to really achieve this minimal lifespan jitter, we have to discuss the task execution times in more detail. In the given timing model the execution time $et(\tau)$ of a task is modelled by its WCET, denoted as $C(\tau)$. However, it can happen that the task execution time $et(\tau)$ is smaller than $C(\tau)$. This can happen due to multiple reasons: the execution time might be variable due to different initial HW states before a task's start or dependency on input data. As another reason, $C(\tau)$ might be an overestimation of the real WCET, i.e., $C(\tau)$ is a safe but pessimistic upper bound of the real WCET. In order to achieve a predictable timing behaviour, the system would have to implement the concept of *logical execution times* (LET) [6]. The basic idea is that a LET is assigned to a task, with the important property that the LET is larger or equal than the real WCET of a task. By using a LET, the produced output of a task is held back at the end of its execution and is only forwarded to the LIFSS of the TTSC

TABLE I
TASKSETS USED FOR QUANTITATIVE ANALYSIS OF TT INTERFACES (VALUES IN MS)

		WCET (τ.c)								
Taskset	*Task Structure*	τ_A	τ_B	τ_C	τ_D	τ_E	τ_F	τ_G	τ_H	T_{tt}
TS1	|AB|	2.0	2.5							4.0
TS2	|ABA|	2.0	2.5							4.0
TS3	|ABA|CDC|	2.0	2.5	1.5	1.5					5.0
TS4	|ABA|CDC|EFE|	2.0	2.5	1.5	1.5	1.0	0.5			7.0
TS5	|ABA|CDC|EFE|GHG	2.0	2.5	1.5	1.5	1.0	0.5	1.0	0.5	9.0
TS6	|ABC|	2.0	2.5	1.5						5.0
TS7	|ABCA|	2.0	2.5	1.5						5.0
TS8	|ABCDA|	2.0	2.5	1.5	1.5					5.0

once the end of the task's LET has arrived. This way, from the output side, it appears that the task has a constant execution time. Assuming that $C(\tau)$ is a safe upper bound of a task's real WCET, we can set $LET(\tau) = C(\tau)$.

However, if a low message lifespan jitter is not a requirement for the concrete system, but rather only a minimised maximum message lifespan is needed, then the LET concept does not have to be implemented. In this case, the message lifespan jitter might be significantly higher than the precision of clock synchronisation. Nevertheless, one would still get a minimised message lifespan for the system with synchronous TT interfaces, either for the maximum message lifespan (TR_{max}) or the sum of message lifespans (TR_{avg}).

V. QUANTITATIVE ANALYSIS

In this section we describe concrete experiments based on Equation 5 of the TT timing model developed in Section IV.

A. Description of Analysis Method

To demonstrate the difference between the synchronous and the asynchronous TT interface, we developed a tool that implements the timing model for message lifespans given in Equation 5. The tool computes the a) maximum lifespan $l(m)$ for all $m \in M$, b) the sum of the lifespan of all messages, and c) the average lifespan of all messages of the TTCS. It can be configured to iterate over different TT bus configurations ($b(\tau)$ for all $\tau \in T$) and different task start configurations ($tr(\tau)$ for all $\tau \in T$). Based on that, the tool facilitates the following optimisations:

- OMAX: search for the configuration to find the maximum jitter of the maximum lifespan among the messages of the TTCS.
- OSUM: search for the configuration to find the maximum jitter of the sum of the lifespan of all messages in the TTCS.

Each of these optimisations shows on one hand the maximum jitter possible in case of the asynchronous TT interface, and on the other hand provides the system configuration to obtain a minimised lifespan (OMAX or OSUM) in case of the synchronous TT interface. The tool allows its user to visualise the scheduling behaviour for the different optimisation goals, using the format explained in Figure 3. As shown in Figure 3, the tool shows the lifespan of each message using a horizontal line from the termination of the sender task till the start of the receiver task. The lifespan of individual messages from the same task may overlap. In order to draw message lifespans of unambiguous lengths in case of overlaps, in the figures the message lifespan for only each 3^{rd} TT period is drawn as a horizontal line.

As mentioned above, the tool can search over both, the possible task offsets as well as over the possible slot assignment on the bus. By searching over possible slot assignments we are able to demonstrate that the slot assignment on the bus ($b(\tau)$ for all $\tau \in T$) influences the lifespan of messages. This is particularly interesting, as the industrial practice starts with the bus-slot assignment, and only thereafter suppliers develop their individual tasks. We demonstrate that one has to break with the established industrial practice of determining the slot assignment before the task development. To get the best timing optimisation on the bus, the slot assignment on the bus and the task offsets need to be optimised simultaneously.

Table I shows the basic communication structure of task sets we have used for our experiments. The syntax |ABA|CDC| is that "|" designates separate communication groups, i.e., tasks τ_A,τ_B communicate with each other and tasks τ_C,τ_D communicate with each other. A sequence of two letters within a communication group means that the 2^{nd} task reads the message written by the 1^{st} task. For example, |ABA| means that task τ_B reads from task τ_A (pattern AB) and that task τ_A reads from task τ_B (pattern BA).

For each task $\tau = \langle tr, c, M_R, m_W \rangle$, the parameters M_R, m_W are given by the described communication structure, while the WCET c chosen with some arbitrary value to fit within a communication period T_{tt}. The first task parameter tr is the result of the performed optimisation method, to find the maximum jitter or the configuration for the minimum message lifespan (max or sum). The bus configuration $b(\tau)$ is also an output of the performed optimisation method.

1) Complexity of the Analysis Method: The OMAX/OSUM searches in general are quite computationally costly with larger systems. For example, the number of different possible bus configurations grows exponentially with the number of message slots. Also the search of task offsets can grow exponentially with the number of tasks.

To cope with that potentially huge search space, we have implemented a numerical search method that uses effective simplifications. Together with an automatic adjustment of the search step size of task offsets, we include systematic placements based on the patterns given in Figure 2.a-d to find the extreme cases for single message lifespans.

Another optimisation splits the task set into task-communication groups. A task communication group is the transitive closure of all tasks communicating with

TABLE II
OMAX OPTIMISATION OF MESSAGE LIFESPAN (VALUES IN MS)

Omax, Smin	max lifespan				sum lifespan			
Taskset	ls: Tmin	ls: Tmax	ls: jitter	ls: jitter rel	ls: Tmin	ls: Tmax	ls: jitter	ls: jitter rel
|AB|	1.00	8.98	7.98	898%	1.00	8.98	7.98	898%
|ABA|	1.80	8.98	7.18	499%	3.50	11.50	8.00	329%
|ABA|CDC|	3.50	10.98	7.48	314%	12.50	32.50	20.00	260%
|ABA|CDC|EFE|	2.76	14.98	12.22	542%	12.00	54.00	42.00	450%
|ABA|CDC|EFE|GHG|	3.80	18.98	15.18	499%	25.50	97.50	72.00	382%
|ABC|	1.36	10.98	9.62	805%	2.50	16.48	13.98	659%
|ABCA|	1.50	10.98	9.48	732%	4.00	19.00	15.00	475%
|ABCDA|	2.00	10.98	8.98	549%	7.50	27.50	20.00	367%

Omax, Smax	max lifespan				sum lifespan			
Taskset	ls: Tmin	ls: Tmax	ls: jitter	ls: jitter rel	ls: Tmin	ls: Tmax	ls: jitter	ls: jitter rel
|AB|	1.00	8.98	7.98	898%	1.00	8.98	7.98	898%
|ABA|	3.80	8.98	5.18	236%	7.50	15.50	8.00	207%
|ABA|CDC|	3.50	10.98	7.48	314%	12.50	32.50	20.00	260%
|ABA|CDC|EFE|	6.41	14.98	8.57	234%	33.00	75.00	42.00	227%
|ABA|CDC|EFE|GHG|	8.50	18.98	10.48	223%	61.50	133.50	72.00	217%
|ABC|	2.77	10.98	8.21	396%	5.50	18.49	12.99	336%
|ABCA|	4.82	10.98	6.16	228%	14.00	24.00	10.00	171%
|ABCDA|	4.64	10.98	6.34	236%	17.50	27.50	10.00	157%

other tasks. By splitting the task set this way, the search for the OMAX/OSUM optimisation can be done separately for each task communication group. While the size of each task communication group still poses an exponential search space, the number of task communication groups now is only a linear search space complexity.

B. Experimental Results for OMAX

The results of the OMAX optimisation with the maximum message lifespan are given in Table II. The table has two parts: the top part "**Smin**" describes the results for the search of the bus configuration with the minimised maximum message lifespan, and the bottom part "**Smax**" describes the results for the search of the bus configuration with the maximised maximum message lifespan.

The columns "**max lifespan**" show the optimisation results for the maximum message lifespan. The columns "**sum lifespan**" show what results the OMAX optimisations provided as a side-effect for the sum of all message lifespans, without optimising for it. The columns "**ls: Tmin**" and "**ls: Tmax**" show the minimum and maximum observed lifespans among the different task start configurations.

Column "**ls: jitter**" calculates the lifespan jitter: "**ls: Tmax**" − "**ls: Tmin**". The column "**ls: jitter rel**" shows the relative lifespan jitter: "**ls: Tmax**" / "**ls: Tmin**" as percentage. In the OMAX optimisation, the information for "**sum lifespan**" is given, to be compared with the corresponding results of the OSUM optimsation. Comparing the top part of Table II with the bottom part, we see that the bus configuration in most cases has a significant influence on the possible minimisation of the maximum message lifespan. The only tasksets where the slot configurations did not show an influence were '|AB|' and '|ABA|CDC|'.

As an example, let us look at the results for taskset |ABA| in Table II. For example, we see that the maximum lifespan of the messages of taskset |ABA| varies between 1.80 and 8.98 due to changing the task offsets. The upper value is by a factor of 4.99 higher than the lower maximum lifespan. However, the table also shows that the slot configuration also has a significant influence on the bus timing, as it shows that the minimum observable maximum message lifespan varies between 1.80 and 3.50. Thus, the bus configuration has in this example an influence of a factor of $3.8/1.8 = 2.1$.

Figure 4 shows the analysis result for the OMAX optimisation for the taskset '|ABA|'. Figure 4.a and Figure 4.b show the results for the bus configuration with the minimum lifespan (Smin), and Figure 4.c and Figure 4.d show it for the maximum lifespan configuration (Smax). Looking at Figure 4.a and Figure 4.b we see that maximum message lifespan of message m_{AB} varies between $1.8ms$ and $8.98ms$ due to the difference in the task offsets. First of all, this shows that the optimisation of the task offsets by using the synchronous TT interface has a significant influence on the message lifespan. We have to note that in Figure 4.a the message lifespan of message m_{AB} is not the minimum possible, as the result in this figure is the minimisation of the maximum message lifespan of message m_{AB} and message m_{BA}. Nevertheless, task τ_A reads immediately after the message m_{AB} has been sent in slot 1, just from task τ_B writing the message and finally sending the message, an additional delay of $0.8ms$ happen. We observe the found start times $tr(\tau_A) = 2.00$ and $tr(\tau_B) = 1.80$. However, in Figure 4.b we have an overall maximum message lifespan for message m_{BA} of $8.98ms$. The figure also illustrates why this message lifespan is so long. We observe that this case is produced by the task start times $tr(\tau_A) = 2.01$ and $tr(\tau_B) = 0.99$. So task τ_A starts execution

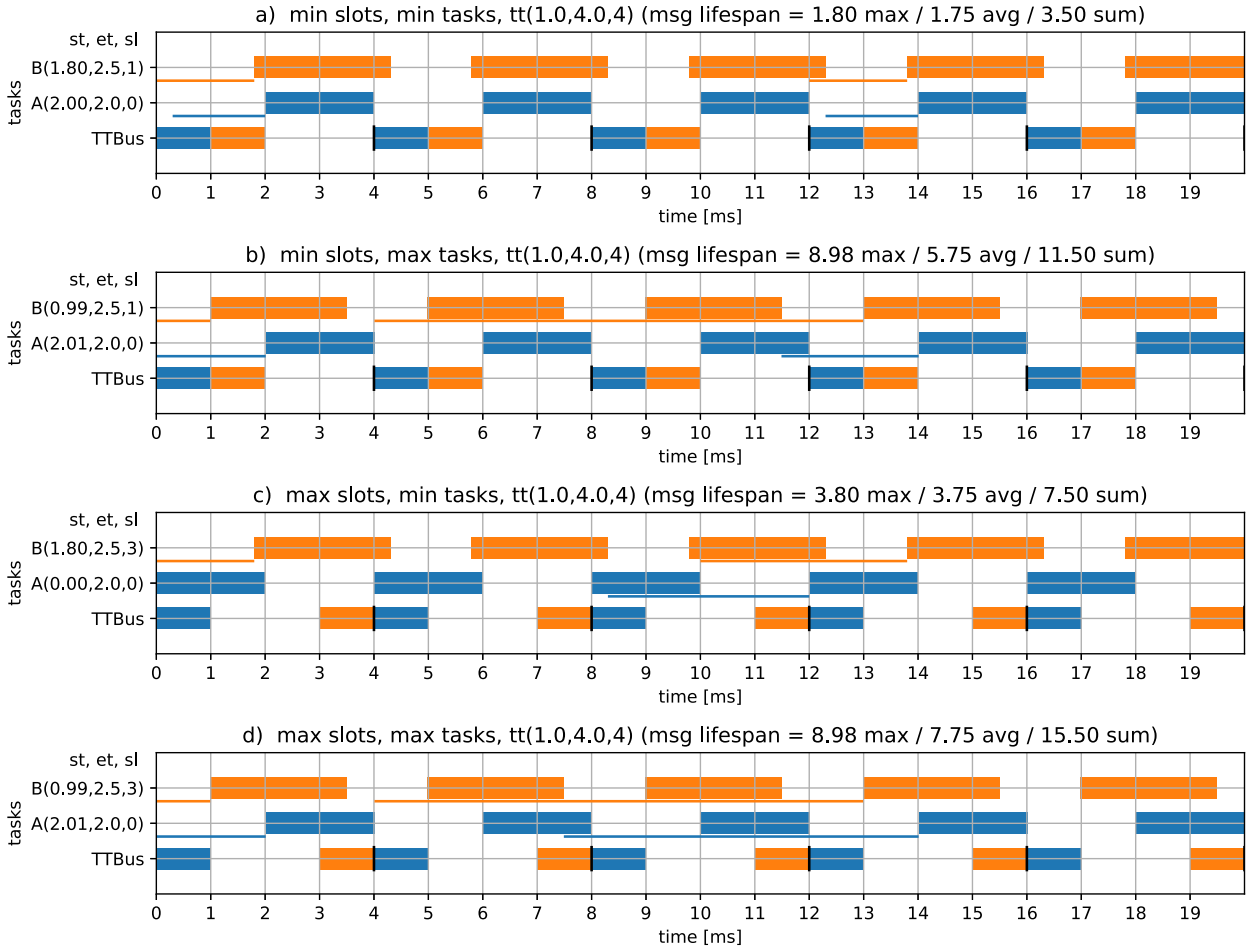

Fig. 4. OMAX optimisation of message lifespan (Taskset '|ABA|').

at time $2.01ms$ and writes its output message m_{AB} at time $4.01ms$ ($tr(\tau_A) + c(\tau_A)$), which misses τ_A's communication slot starting at time $4.00ms$. Thus the message only gets communicated in the next round, finishing its communication at time $9.00ms$. Task τ_B has a relative start $tr(\tau_B) = 0.99$, thus its execution start at time $2 \cdot 4ms + 0.99ms = 8.99ms$ is just too early to get the message, so we have to consider τ_B's execution in the next communication round, i.e., $3 \cdot 4ms + 0.99ms = 12.99ms$. The resulting lifespan of message m_{AB} is $12.99ms - 4.01ms = 8.98ms$. Compare this with the lowest maximum lifespan given in Figure 4.a, which has the task start times $tr(\tau_A) = 2.00$ and $tr(\tau_B) = 1.80$. In this case m_{AB} still has the maximum lifespan, but now the lifespan $l(m_{AB})$ is only $1.80ms$. Note that in Figure 4.a one might still be able to reduce the lifespan $l(m_{AB})$ further. The given analysis shows the minimum of the maximum lifespan of all messages in the system. When comparing Figure 4.a and Figure 4.c we see that the maximised bus configuration Figure 4.c allows for a less optimal maximum message lifespan with the bus configuration $b(\tau_A) = 0$ of $b(\tau_B) = 3$. This difference can be explained by the fact that in Figure 4.a between the end of $b(\tau_B) = 2ms$ and the begin of $b(\tau_A) = 4ms$ the execution time $c(\tau_A) = 2ms$ was fitting exactly in. However, in Figure 4.c between the end of $b(\tau_A) = 1ms$ and the begin of $b(\tau_B) = 3ms$ the execution time $c(\tau_B) = 2.5ms$ does not fit in.

As another example of the OMAX optimisation, Figure 5 shows the results for the taskset '|ABA|CDC|EFE|'. This example shows in Figure 5.a and Figure 5.b as well that the task start times have a significant influence, namely between $2.76ms$ to $14.98ms$. However, this examples also shows with Figure 5.a and Figure 5.c that the bus configuration also has a significant influence, namely between $2.76ms$ to $6.41ms$ for the minimum of the maximal message lifespans.

C. Experimental Results for OSUM

Table III shows the results for OSUM optimisation. For the OSUM optimisation, the bus configuration ($b(\tau)$ for all $\tau \in T$) also shows a difference, given in the top part the results for the bus configuration with the minimum lifespan sum, and the bottom part the results for the bus configuration with the maximum lifespan sum. For example, for taskset '|ABA|CDC|EFE|' the jitter of the lifespan sum for Smin is $54.00ms - 12.00ms = 42.00ms$, and for Smax it is $75.00ms - 33.00ms = 42.00ms$. While the jitter of the lifespan sum here is the same, the Tmin shows a significant change: for Smin it is $12.00ms$, and for Smax it is $33.00ms$. This shows that the slot optimisation has a significant influence on the minimum achievable message lifespan sum. Overall, for all task sets except '|AB|' that has a single message, Table III shows a significant improvement of the lifespan sum if one optimises the bus configuration in parallel with the task schedules.

Figure 6 shows the analysis result for the OSUM optimisation for the taskset '|ABA|'. Figure 6.a and Figure 6.b focus

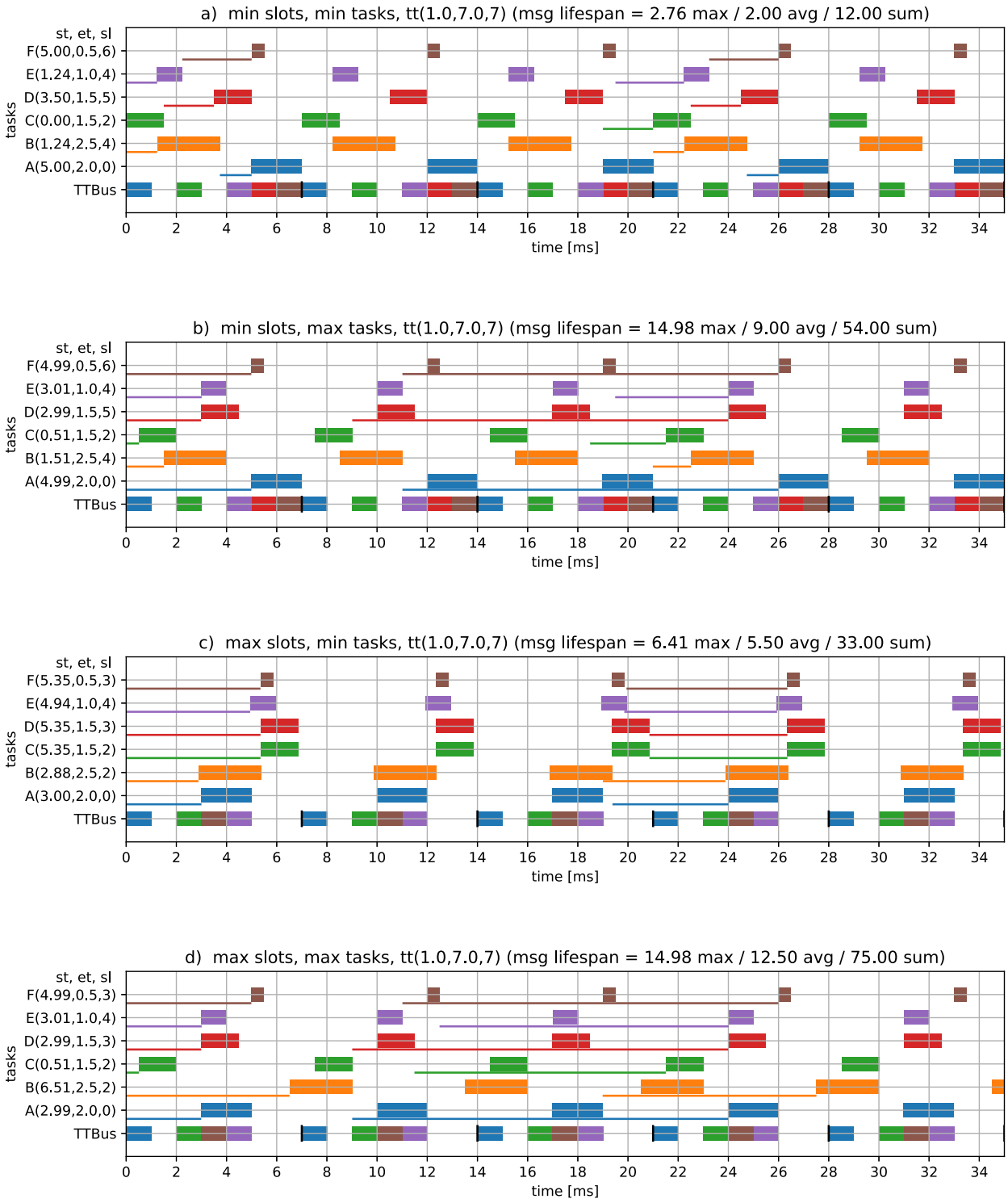

Fig. 5. OMAX optimisation of message lifespan (Taskset '|ABA|CDC|EFE|').

on the bus configuration with the minimum lifespan, Figure 6.c and Figure 6.d consider the maximum lifespan configuration. With the OSUM optimisation, the bus configuration influences the message lifespan sum. For the minimum bus configuration in Figure 6.a and Figure 6.b $b(\tau_A) = 0$ and $b(\tau_B) = 1$, for the maximum bus configuration in Figure 6.c and Figure 6.d $b(\tau_A) = 0$ and $b(\tau_B) = 3$.

As another example of the OSUM optimisation, Figure 7 shows the results for the taskset '|ABA|CDC|EFE|'. Figure 7.a and Figure 7.c show that the bus configuration has a significant influence on the possible minimisation of the message lifespan sum. The change factor is $33.00ms/12.00ms = 2.75$. While the change due to the tasks offsets (Tmin, Tmax) is important to optimise, nevertheless, also the bus configuration has an important contribution to the minimum achievable lifespan sum.

D. Discussion

The experimental results for both OMAX and OSUM optimisation have shown that the consideration of how to

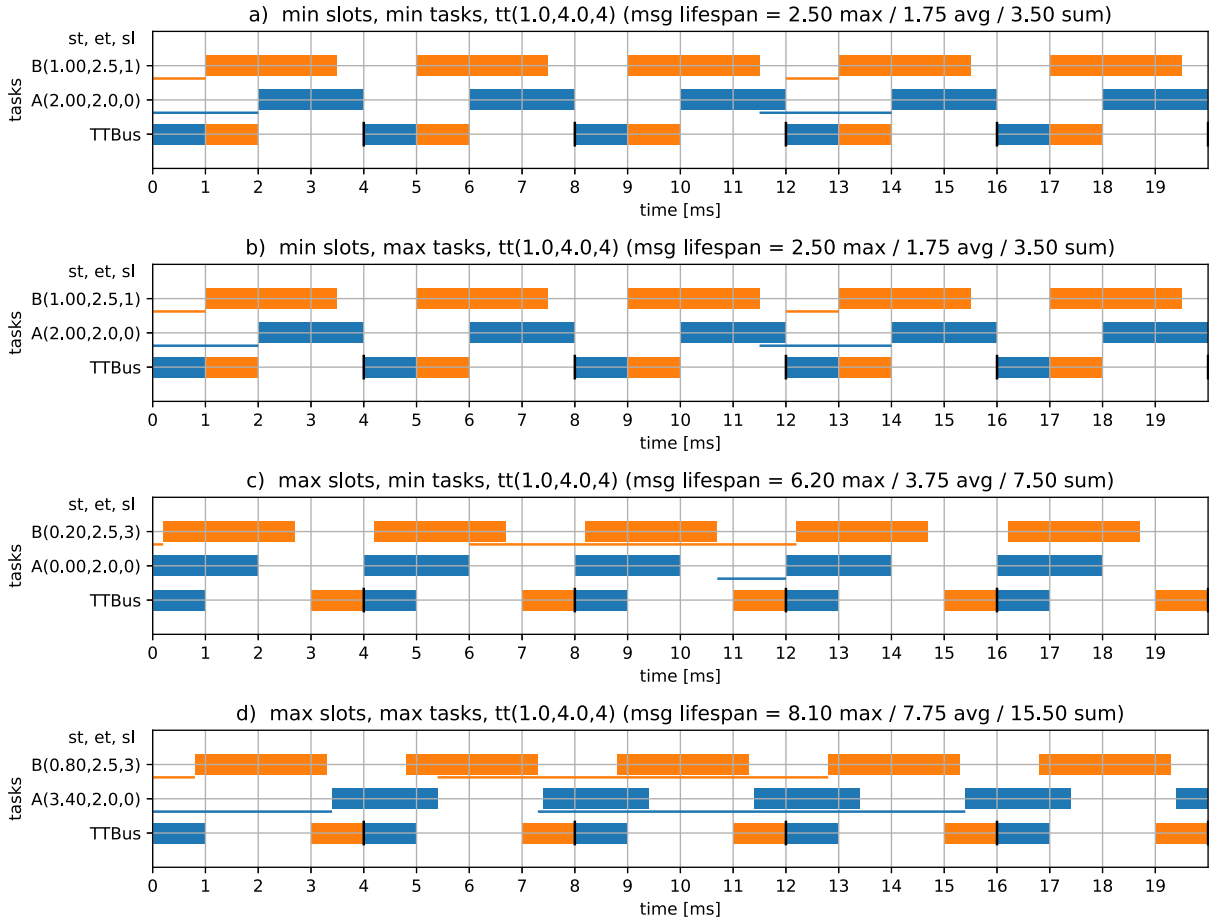

Fig. 6. OSUM optimisation of message lifespan (Taskset '|ABA|').

TABLE III
OSUM OPTIMISATION OF MESSAGE LIFESPAN (VALUES IN MS)

Osum, Smin	sum lifespan				max lifespan			
Taskset	ls: Tmin	ls: Tmax	ls: jitter	ls: jitter rel	ls: Tmin	ls: Tmax	ls: jitter	ls: jitter rel
|AB|	1.00	8.98	7.98	898%	1.00	8.98	7.98	898%
|ABA|	3.50	11.50	8.00	329%	2.50	6.00	3.50	240%
|ABA|CDC|	12.50	32.50	20.00	260%	4.21	9.55	5.34	227%
|ABA|CDC|EFE|	12.00	54.00	42.00	450%	3.12	12.00	8.88	385%
|ABA|CDC|EFE|GHG|	25.50	97.50	72.00	382%	5.60	18.20	12.60	325%
|ABC|	2.50	17.48	14.98	699%	1.36	10.22	8.85	749%
|ABCA|	4.00	19.00	15.00	475%	1.59	9.41	7.82	591%
|ABCDA|	7.50	27.50	20.00	367%	3.21	8.50	5.29	264%

Osum, Smax	sum lifespan				max lifespan			
Taskset	ls: Tmin	ls: Tmax	ls: jitter	ls: jitter rel	ls: Tmin	ls: Tmax	ls: jitter	ls: jitter rel
|AB|	1.00	8.98	7.98	898%	1.00	8.98	7.98	898%
|ABA|	7.50	15.50	8.00	207%	6.20	8.10	1.90	131%
|ABA|CDC|	17.50	37.50	20.00	214%	6.42	10.37	3.95	161%
|ABA|CDC|EFE|	33.00	75.00	42.00	227%	8.47	14.46	5.99	171%
|ABA|CDC|EFE|GHG|	61.50	133.50	72.00	217%	14.60	18.98	4.38	130%
|ABC|	6.50	21.48	14.98	330%	4.09	10.90	6.81	266%
|ABCA|	14.00	29.00	15.00	207%	5.32	9.86	4.55	185%
|ABCDA|	17.50	37.50	20.00	214%	5.36	10.98	5.62	205%

interface with TT systems does have a significant influence on the message lifespan (and as such on the latency of real-time control). The results have shown that the jitter of the message lifespan for asynchronous TT interfaces can come close to the limit of $2{\cdot}T_{tt}$ as described in [1]. With the taskset '$|\mathtt{AB}|$' it basically reached that limit. To avoid this high message

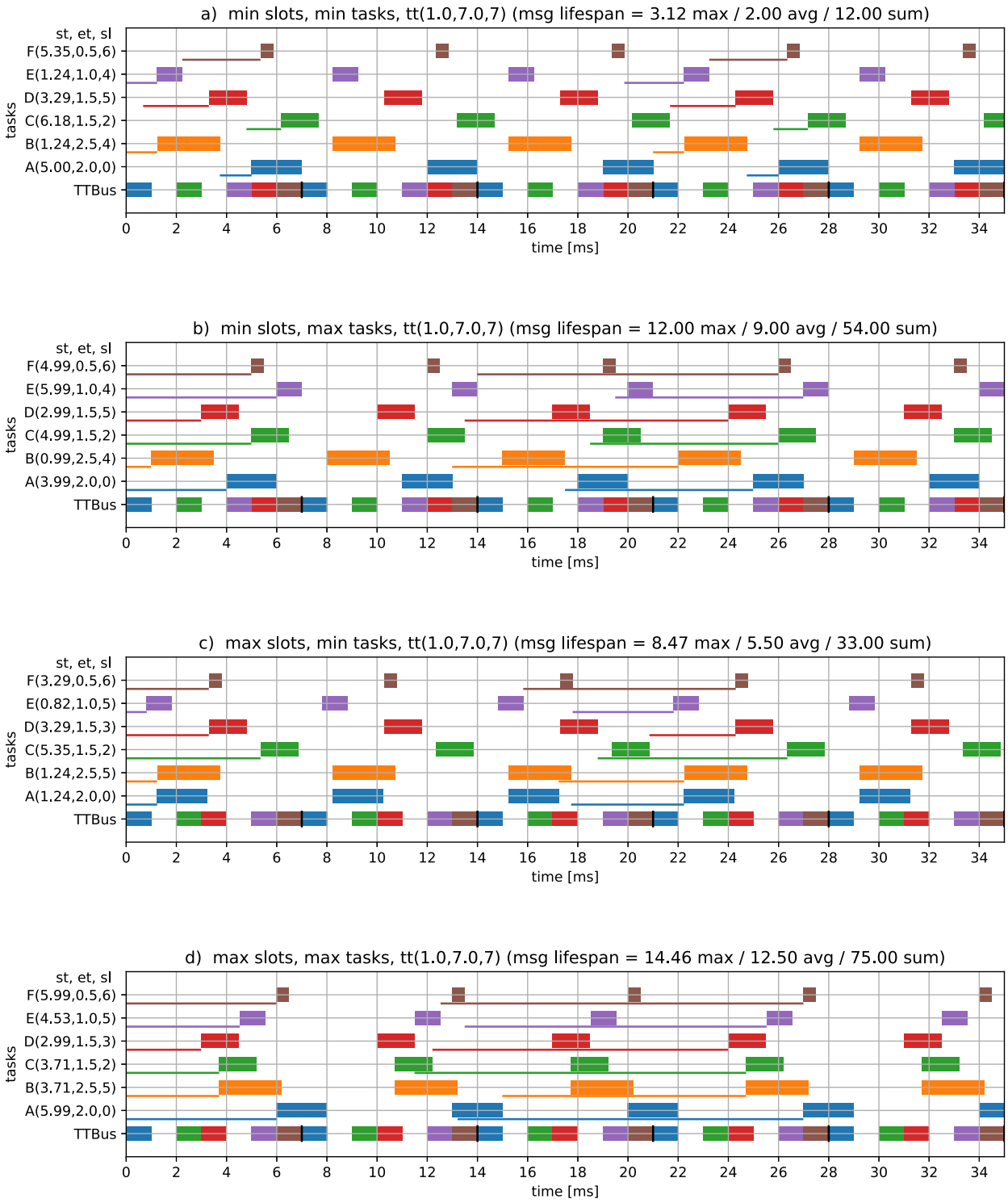

Fig. 7. SUM optimisation of message lifespan (Taskset '|ABA|CDC|EFE|').

lifespan jitter, one should consider the extra effort required to build a synchronous TT interface with the tasks start times aligned to the communication schedule.

As described before, the OMAX optimisation to reduce the maximum message lifespan is most suitable for hard real-time systems. However, the OSUM optimisation to reduce the message lifespan sum of all messages together comes fairly close to the OMAX optimisation in terms of minimising also the maximum message lifespan, but brings in significant further reductions of message lifespans, making the real-time control system in overall more agile.

With respect to bus configuration ($b(\tau)$ for all $\tau \in T$) it has been shown with the given tasksets that the bus configuration has a significant influence on the Tmin as well as the Tmax values of the OMAX as well as the OSUM optimisation.

VI. Summary and Conclusion

This article focuses on the timing behaviour of time-triggered (TT) communication by either using asynchronous or synchronous TT interfaces. To do so, we have developed a timing model for the TT communication, which allows to calculate the time from writing a message till the time of reading a message, which we call message lifespan. For real-time systems we want the message lifespans to be low and predictable. To get a predictable message lifespan we require the message lifespan jitter to be low.

We have implemented that timing model in a tool that allows to search for TT configuration parameters (relative task start times and assignment of communication slots) that yield the maximum and minimum of the message lifespan. We have done this optimisation for both, the maximum lifespan of any message in the system and for the sum of the lifespans of all messages in the system. The tool also allows to plot example schedules to visually explain the calculated message lifespans.

This timing model can be used for asynchronous TT interfaces to obtain a precise bound of the possible message lifespan (latency) and lifespan jitter. In addition, the timing model can be used to configure synchronous TT systems to obtain a minimal message lifespan with the jitter limited to that of clock synchronisation. One important insight from the analysis is that the bus configuration (slot assignment) also has an important contribution to the minimisation of the bus timing. While in industry the pragmatic approach for the synchronous TT interface is to make a bus configuration at the beginning and then only optimise the task offsets, our results show that a joint optimisation of bus configuration and task offsets could bring a significant improvement of the message lifespan, in some cases more than a factor of 2.

References

[1] P. Puschner and R. Kirner, "Asynchronous vs. synchronous interfacing to time-triggered communication systems," *J. Syst. Archit.*, vol. 103, Feb. 2020, Art. no. 101690. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1383762119304977

[2] H. Kopetz and G. Grunsteidl, "TTP—A time-triggered protocol for fault-tolerant real-time systems," in *Proc. 23rd Int. Symp. Fault-Tolerant Comput. (FTCS)*, 1993, pp. 524–533.

[3] H. Kopetz and G. Bauer, "The time-triggered architecture," *Proc. IEEE*, vol. 91, no. 1, pp. 112–126, Jan. 2003.

[4] H. Kopetz, A. Ademaj, P. Grillinger, and K. Steinhammer, "The time-triggered Ethernet (TTE) design," in *Proc. 8th IEEE Int. Symp. Object-Oriented Real-Time Distrib. Comput. (ISORC)*, 2005, pp. 22–33.

[5] *ISO 17458-1:2013 Road Vehicles—FlexRay Communications System—Part 1: General Information and Use Case Definition*, Standard ISO/TC 22/SC 31, ISO 17458-1 to 17458-5, reviewed and confirmed in 2019, International Standards Organisation, New York, NY, USA, Feb. 2013. [Online]. Available: https://www.iso.org/standard/59804.html

[6] T. A. Henzinger, B. Horowitz, and C. M. Kirsch, "Giotto: A time-triggered language for embedded programming," *Proc. IEEE*, vol. 91, no. 1, pp. 84–99, Jan. 2003.

[7] J. Sparsø, E. Kasapaki, and M. Schoeberl, "An area-efficient network interface for a TDM-based network-on-chip," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*. San Jose, CA, USA: EDA Consortium, 2013, p. 1044.

[8] A. Koubâa, A. Cunha, M. Alves, and E. Tovar, "TDBS: A time division beacon scheduling mechanism for ZigBee cluster-tree wireless sensor networks," *Real-Time Syst.*, vol. 40, no. 3, pp. 321–354, Dec. 2008, doi: 10.1007/s11241-008-9063-4.

[9] A. Ahmad and Z. Hanzalek, "An energy-efficient distributed TDMA scheduling algorithm for ZigBee-like cluster-tree WSNs," *ACM Trans. Sensor Netw.*, vol. 16, no. 1, pp. 1–41, Feb. 2020, doi: 10.1145/3360722.

[10] A. Varga, *Recent Advances in Network Simulation* (EAI/Springer Innovations in Communication and Computing (EAISICC)). London, U.K.: Springer, May 2019, pp. 3–51.

[11] D. A. Menasce, L. W. Dowdy, and V. A. F. Almeida, *Performance by Design: Computer Capacity Planning By Example*. Upper Saddle River, NJ, USA: Prentice-Hall, 2004.

[12] H. Kopetz and J. Reisinger, "The non-blocking write protocol NBW: A solution to a real-time synchronization problem," in *Proc. Real-Time Syst. Symp.*, Dec. 1993, pp. 131–137.

[13] P. Puschner and B. Frömel, "Composable component interfaces for time-triggered systems," in *Proc. 8th Medit. Conf. Embedded Comput. (MECO)*, Jun. 2019, pp. 1–4.

[14] P. Pop, P. Eles, and Z. Peng, "Scheduling with optimized communication for time-triggered embedded systems," in *Proc. 7th Int. Workshop Hardw./Softw. Codesign (CODES)*, May 1999, pp. 178–182.

[15] S. S. Craciunas, R. S. Oliver, and V. Ecker, "Optimal static scheduling of real-time tasks on distributed time-triggered networked systems," in *Proc. IEEE Emerg. Technol. Factory Autom. (ETFA)*. Barcelona, Spain: IEEE Computer Society, Sep. 2014.

[16] W. Steiner, G. Bauer, B. Hall, and M. Paulitsch, *Time-Triggered Communication*. Boca Raton, FL, USA: CRC Press, Aug. 2011.

[17] A. Minaeva, B. Akesson, Z. Hanzálek, and D. Dasari, "Time-triggered co-scheduling of computation and communication with jitter requirements," *IEEE Trans. Comput.*, vol. 67, no. 1, pp. 115–129, Jan. 2018.

[18] H. Kopetz, "The time-triggered model of computation," in *Proc. 19th IEEE Real-Time Syst. Symp.*, Dec. 1998, pp. 168–177.

[19] P. Puschner and R. Kirner, "From time-triggered to time-deterministic real-time systems," in *Proc. 5th Work. Conf. Distrib. Parallel Embedded Syst. (IFIP)*, Braga, Portugal, Oct. 2006, pp. 115–124.

[20] H. Kopetz, *Real-Time Systems*, 2nd ed. London, U.K.: Springer, 2011.

[21] H. Kopetz and R. Nossal, "Temporal firewalls in large distributed real-time systems," in *Proc. 6th IEEE Comput. Soc. Workshop Future Trends Distrib. Comput. Syst.*, Oct. 1997, pp. 310–315.

[22] I. Ripoll and R. Ballester-Ripoll, "Period selection for minimal hyper-period in periodic task systems," *IEEE Trans. Comput.*, vol. 62, no. 9, pp. 1813–1822, Sep. 2013.

[23] *FlexRay Communications System—Protocol Specification, Version 2.0*, 2nd ed., FlexRay Consortium, Jun. 2004.