End-to-End Delays of Event-Triggered Overlay Networks in a Time-Triggered Architecture

R. Obermaisser
Vienna University of Technology, Austria

Abstract— **Event-triggered overlay networks are a solution for extending time-triggered networks with support for additional event-triggered communication activities. In addition to time-triggered message transmissions, applications can request the dissemination of messages at points in time that need not be statically predefined at design time. Thus, the flexibility is improved and the reuse of event-triggered legacy applications becomes possible. This paper analytically describes the delays of message transmissions on such an event-triggered overlay network on top of a time-triggered physical network. Knowledge about these message delays is significant for the development of new real-time applications, as well as for migrating existing applications from a physical event-triggered network to an event-triggered overlay network. After identifying the determinant parameters, we explain the computation of the worst-case and best-case message delays. The generic analytical description is applied to real-world examples and compared to the message delays that have been observed in actual implementations of event-triggered overlay networks on top of TTP and Time-Triggered Ethernet.**

I. Introduction

FEDERATED system architectures encompass multiple distributed computer systems, each dedicated to a corresponding application subsystem (e.g., powertrain subsystem in a car, cabin pressurization in a plane). This architectural paradigm has lead to increasing numbers of node computers and networks, e.g., in the automotive domain with up to 75 Electronic Control Units (ECUs) and multiple physical networks with different communication protocols (e.g., CAN [1], ByteFlight [2]). In the automotive domain, the upcoming introduction of time-triggered networks [3] enables the shift to an integrated architecture that shares a single time-triggered network with high-bandwidth (e.g., 5 Mbps [4], 25 Mbps [5]) among multiple application subsystems.

A major challenge in such an integrated architecture is the management of the communication resources for multiple application subsystems, while meeting requirements such as composability, support for modular certification, fault isolation, and legacy carry-over [6]. For this purpose, previous work has addressed the establishment of overlay networks on top of a time-triggered physical network [7]. By using a subset of the communication slots of a time-triggered network, each overlay network can serve as an encapsulated communication infrastructure with guaranteed communication resources of a respective application subsystem.

This work has been supported in part by the European IST project ARTIST2 under project No. IST-004527 and the European IST project DECOS under project No. IST-511764.

Furthermore, overlay networks on top of a time-triggered physical network can support both event-triggered and time-triggered control, thus taking into account that both control paradigms possess respective advantages in different application domains. While the predictability of time-triggered control faciliates the construction of safety-critical application subsystems (e.g., X-by-wire), the on-demand communication activities of event-triggered control are superior w.r.t. flexibility in non safety-critical application subsystems (e.g., comfort, multimedia).

However, the event-triggered overlay networks incur an overhead resulting from the need to map the event-triggered control paradigm to the underlying time-triggered physical network. This mapping induces computational delays and communication delays, which need to be considered when constructing real-time applications. Therefore, this paper analyses the temporal properties of event-triggered overlay networks on top of a time-triggered physical network.

The main contribution of this paper is an analytical description of the end-to-end communication delays in applications that exchange messages on such an event-triggered overlay network. Based on a system model of the integrated architecture, we derive significant properties that determine the end-to-end delays. Also, the paper exemplifies the analytical description in realistic examples with existing time-triggered communication protocols (i.e., the Time-Triggered Protocol [5] and Time-Triggered Ethernet [8]).

The paper is structured as follows. A generic system model with support for event-triggered overlay networks on top of a time-triggered physical network is the focus of Section II. Section III analytically describes the end-to-end communication delays of such an event-triggered overlay network. Illustrating examples and a comparison with measurenent results from prototype implementations are the content of Section IV. The paper finishes with a conclusion in Section V.

II. Time-Triggered Architecture with Event-Triggered Overlay Networks

This section presents the system model for the analytical description of the message delays on the event-triggered overlay networks. Physically, the system consists of a set of nodes and a time-triggered physical network that interconnects these nodes (see Figure 1). Each node consists of a host processor and a communication controller. The communication controller executes a time-triggered communication protocol to provide a time-triggered message transport service and a global time base. Any communication protocol that provides

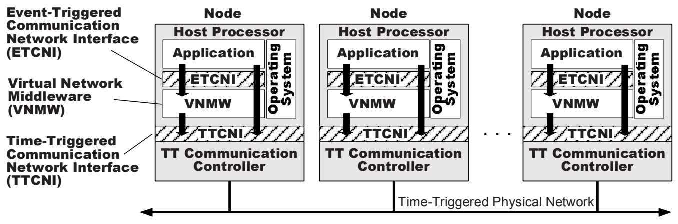
Event-Triggered Communication Network Interface (ETCNI)
Virtual Network Middleware (VNMW)
Time-Triggered Communication Network Interface (TTCNI)
Node
Host Processor
Application
ETCNI
VNMW
Operating System
TTCNI
TT Communication Controller
Time-Triggered Physical Network

Fig. 1. Distributed Computer System with Event-Triggered Overlay Networks on top of a Time-Triggered Physical Network

these services can be used as a basis for the construction of the event-triggered overlay networks. A comprehensive survey of protocols that meet these requirements can be found in [9]. Examples of suitable communication protocols are the Time-Triggered Protocol (TTP) [5], Time-Triggered Ethernet (TTE) [8] and FlexRay [4].

A. Communication Controller

At each node the communication controller (e.g., TTP controller C2 [10], FlexRay Controller MFR4200 [11]) uses Time Division Multiple Access (TDMA) to control the media access to the time-triggered network. The periodicity of the message transmissions is defined within a cluster cycle. Information about the points in time of all message transmissions during a cluster cycle is contained in a static message schedule. The cluster cycle is divided into a sequence of TDMA rounds and each TDMA round is partitioned into time slots. Slots are statically assigned to nodes in a way that allows each of them to send a message during a TDMA round.

At the interface to the host processor, the communication controller provides a memory element with state messages. The memory element contains outgoing state messages that are written by the host processor and read by the communication controller prior to broadcasting them on the time-triggered network at a priori specified instants. In addition, the memory element contains incoming state messages that are read by the host processor and updated by the communication controller with state messages read from the time-triggered network (i.e., information broadcast by other nodes). We denote this memory element, which is the interface between the host processor and the communication controller, as the *Time-Triggered Communication Network Interface (TTCNI)*.

The TTCNI is designed for information with state semantics [12, p. 32], thus a state variable is overwritten whenever a more recent value is available (also called update-in-place). The synchronization for ensuring that only consistent data is read can be guaranteed with a synchronization protocol (e.g., non-blocking write protocol [13]) or using implicit synchronization. The latter solution uses a time-triggered task schedule in the host processor, which is synchronized to the TDMA scheme of the time-triggered network. This time-triggered task schedule can prevent overlapping access operations between the communication controller and the software in the host processor.

B. Host Processor

The software of the host processor in a node (cf. Figure 1) encompasses an *operating system*, *application tasks*, and the so-called *overlay network middleware*. The purpose of the middleware is to exploit the TTCNI provided by the communication controller for the establishment of event-triggered overlay networks.

1) Operating System: The operating system schedules the application tasks and the tasks of the middleware. In the scope of the timing analysis performed in this paper, we distinguish two types of tasks depending on the activation by the operating system:

a) ***Time-triggered tasks****:* The activation of time-triggered tasks occurs at predefined points in time according to a static schedule that is fixed at design time. Time-triggered tasks can be phase-aligned with other time-triggered tasks and with the communication slots of the underlying time-triggered network. Phase-alignment not only enables implicit synchronization, but also allows the construction of real-time transactions with tightly synchronized actions [12, p. 104]. In such a phase-aligned transaction, all communication and computational activities follow densely after each other, e.g., via a sequence that involves a computational delay at a sender (e.g., sensor node), a communication delay on the time-triggered network, and a computational delay at a receiver (e.g., actuator node).

b) ***Event-triggered tasks****:* The activation of event-triggered tasks is triggered by significant events, such as an interrupt. Event-triggered tasks can be more resource efficient for irregular computational activities. A task is only executed on demand, thus avoiding the need to reserve a periodically recurring time interval at design time. Event-triggered tasks can be required, if the CPU overhead incurred by the sampling of an event using a time-triggered task is considered too expensive (e.g., when a short response time to a sporadic event would require a high sampling frequency).

When both time-triggered and event-triggered tasks need to be scheduled by the operating system, contention between these two types of tasks for the CPU needs to resolved. A possible solution is the prioritization of the time-triggered tasks, which is applied as the basic concurrency rule in the TMO programming model [14].

C. Application Tasks

The application software can encompass multiple time-triggered and event-triggered tasks. Time-triggered tasks serve for regular computational activities (e.g., a control loop, the periodic sampling of events). Event-triggered tasks support irregular computational activities for reacting to events as they occur (e.g., an alarm, a user request from the environment).

Regardless of whether the invocation of a task occurs time-triggered or event-triggered, a task can engage in time-triggered and event-triggered communication. The control paradigm for the task invocation is orthogonal to the control paradigm used for communication. For performing time-triggered communication, the application task accesses the TTCNI. For event-triggered communication, a task exploits the services provided at the Event-Triggered Communication

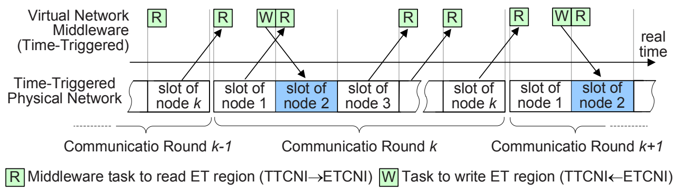

Fig. 2. Scheduling of Middleware Tasks

Network Interface (ETCNI) by the overlay network middleware.

D. Overlay Network Middleware

The purpose of the middleware is the mapping of the TTCNI provided by the communication controller onto the ETCNI. The ETCNI supports on-demand communication activities, i.e., the message transmission requests from the application can occur at a priori unknown instants. In addition, messages can exhibit event semantics, i.e., information that is associated with a particular event (e.g., temperature increase by 2 degrees). In order to reconstruct the current state of a real-time entity from messages with event information, it is essential to process every message exactly once. The loss of a single message with event information can affect state synchronization between a sender and a receiver. Consequently, the ETCNI must support exactly-once delivery semantics and provide message buffers (e.g., queues) for incoming and outgoing messages. Message buffers also allow to handle bursts, i.e., limited intervals of time during which the load of messages from the application exceeds the bandwidth of the communication system.

For mapping the ETCNI to the TTCNI, the state variables in the TTCNI are subdivided. In a node that sends messages on an event-triggered overlay network, we reserve a part of the outgoing state variable that is periodically broadcast on the time-triggered network. This part of the outgoing state variable (denoted as *ET region*) will be used for the dissemination of messages from the ETCNI, while the rest of the state variable remains available for time-triggered communication (denoted as *TT region*). In analogy, each incoming state variable is split into an ET region and a TT region.

The overlay network middleware disseminates the messages in the outgoing buffers of the ETCNI via the bandwidth provided via the ET regions in the TTCNI. Although message transmission requests can occur at arbitrary instants, the dissemination of the messages on the underlying time-triggered network is always performed at the predefined global points in time of the communication slots. The overlay network middleware performs a sampling of the event-triggered message transmission requests and defers their transmission until the next communication slot of the node occurs in the TDMA scheme.

Message fragmentation, i.e., the filling of ET regions with packets from as many messages as possible, ensures that the ET regions are optimally utilized in case of varying message sizes. The overlay network middleware employs periodic tasks that are executed synchronously with the underlying time-triggered communication protocol (cf. Figure 2). After the communication slot of another node, the overlay network middleware is invoked in order to read the data of the ET region from the TTCNI. With the packets from the ET region, the overlay network middleware assembles messages. In case a message is completely assembled, it is stored in the ETCNI.

In addition, the overlay network middleware is invoked prior to the communication slot of the node itself in order to update the ET region in the TTCNI. As long as space is available in the ETCNI, the overlay network middleware reads messages from the ETCNI, fragments them into packets and stores the packets in the TTCNI.

III. End-to-End Delays

This section describes the end-to-end delay of a message transmission over an event-triggered overlay network. A message transmission starts with a transmission request that is issued by the application on the sending node at the *request instant* t_{request}. The obligation of the communication system, i.e., the event-triggered overlay network, is to disseminate the message using the underlying time-triggered communication protocol. The transmission finishes with the delivery of the message to the application on the receiving node at the *delivery instant* t_{deliver}. The duration of the time interval between the request instant and the delivery instant is called the *end-to-end delay*.

As depicted in Figure 3, the end-to-end delay involves task scheduling delays (application tasks, overlay network middleware tasks) at the sender and the receiver, execution times of the overlay network middleware, and the delay for actually transmitting the message on the time-triggered physical network. In detail, these delays which are part of the end-to-end delay are described in the following:

1. Sampling delay: After the application has requested the transmission of a message at the *transmission request instant* t_{request}, a delay occurs until the overlay network middleware processes the transmission request at the next *sampling instant* t_{sample}. This delay, which is called the *sampling delay* d_{sample}, is determined by the transmission request instant relative to the activation time of the overlay network middleware in the time-triggered task schedule (cf. Figure 2). The time-triggered task schedule defines the execution slots of the overlay network middleware, thus controlling the sampling points for the transmission requests.

The best-case sampling delay is zero, in case the transmission request occurs prior to a sampling point. The worst-case sampling delay is d_{round} and results from a transmission request immediately after a sampling point.

$$0 \leq d_{\text{sample}} \leq d_{\text{round}}$$

2. Middleware delay at sender: After the invocation of the overlay network middleware, the message is subject to a delay that is equal to the worst-case execution time of the overlay network middleware. We denote this delay as the *middleware delay at the sender* d_{mw}. During this time, the overlay network middleware reads the message from the ETCNI (e.g., queues) at the interface to the application and writes packets into the TTCNI of the communication controller.

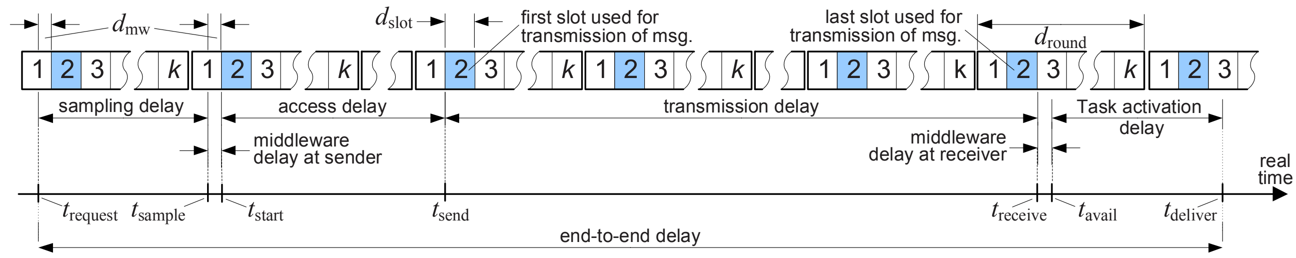

Fig. 3. End-to-End Delay of an Event-Triggered Overlay Network (Depicted for the Node with Slot 2)

As discussed in Section II, the overlay network middleware is phase-aligned with the time-triggered communication schedule. Hence, the reading of the TTCNI through the communication controller and the transmission on the time-triggered physical network follow immediately after the worst-case execution time of the overlay network middleware at the start instant t_{start}. In order to ensure that the TTCNI is updated in time, i.e., before the communication controller reads the TTCNI and broadcasts its contents on the time-triggered physical network, the time-triggered task schedule must select an activation instant t_{sample} that accommodates the worst-case execution time of the overlay network middleware.

3. Access delay: Following the processing through the overlay network middleware, the transmission of the message will start after the access delay d_{access} at the send instant t_{send}. The access delay results from competing message transmission requests in conjunction with limited communication resources. If the transmission of more messages is requested at a node than can be transmitted using a single ET region, then a subset of the messages must be delayed. Since the underlying time-triggered communication protocol assigns via the TDMA scheme a dedicated slot to each node, transmission requests issued at different nodes do not compete in an overlay network. Hence, only the transmission requests issued at the same node determine the access delay.

The access delay depends on the number of bytes n_{priormsg} from competing messages that need to be transferred prior to the send instant. It is determined by the size of the ET region n_{ETregion} and the message scheduling policy (e.g., First-In/First-Out (FIFO), priority-based). The access delay is always a multiple of the TDMA round duration, because the contents of the ET region with outgoing data from a node are broadcast exactly once in each TDMA round.

$$d_{\text{access}} = \lfloor n_{\text{priormsg}} / n_{\text{ETregion}} \rfloor \cdot d_{\text{round}}$$

We round down the number of TDMA rounds, because the remaining bytes from the prior messages that do not consume a complete ET region will be considered in the transmission delay (see below).

A special case with high practical importance is an ETCNI containing a queue with a FIFO policy for scheduling the outgoing messages from the node. In the worst case, the message that is subject to the computation of the maximum end-to-end delay is the last element in the queue.

$$\text{queue of length } n_{\text{queue}} \;\longrightarrow\; n_{\text{priormsg}} = (n_{\text{queue}} - 1) \cdot n_{\text{msg}}$$

4. Transmission delay: After the transmission of a message has started at instant t_{send}, the ET regions in a number of consecutive TDMA rounds are required to relay the message to the receivers. The length of the time interval encompassing these TDMA rounds is called the *transmission delay* $d_{\text{transmission}}$. It depends on the message size n_{msg}, the ET region size n_{ETregion}, and the residue of the prior messages that remains after the access delay.

In general, the ET region of the first TDMA round during the transmission delay is not fully available. The transmission of remaining bytes from the prior messages is required, in case the total size of the prior messages is no multiple of the ET region size. This residue of the prior messages is:

$$n_{\text{residue}} = n_{\text{priormsg}} \bmod n_{\text{ETregion}}$$

By adding the message size n_{msg} to n_{residue}, we gain the number of bytes that are transferred after the send instant. Dividing this value by the ET region size results in the number of required TDMA rounds for the transfer:

$$n_{\text{rounds}} = \lceil ((n_{\text{priormsg}} \bmod n_{\text{ETregion}}) + n_{msg}) / n_{\text{ETregion}} \rceil$$

However, in the last TDMA round, the receiver does not have to wait for the completion of the TDMA round. As depicted in Figure 2, the message under consideration arrives at the receiver after the last slot with data from the ET region (i.e., at the receive instant t_{receive}). In the formula for the transmission delay, this fact is considered by subtracting the duration of one TDMA round and adding the duration of one communication slot (d_{slot}):

$$d_{\text{transmission}} = \left(\left\lceil ((n_{\text{priormsg}} \bmod n_{\text{ETregion}}) + n_{msg}) / n_{\text{ETregion}} \right\rceil - 1 \right) \cdot d_{\text{round}} + d_{\text{slot}}$$

5. Middleware delay at receiver: After the last packet of the message has arrived at the sender at the receive instant t_{receive}, the overlay network middleware can assemble the message and insert it into the ETCNI. We denote this processing delay as the *middleware delay at the receiver*. Like the middleware delay at the sender, it is equal to the worst-case execution time d_{mw} of the overlay network middleware. After the middleware delay (i.e., at instant t_{avail}), the message is ready to be read by the application.

6. Task activation delay: Even though the message is available in the ETCNI at t_{avail}, the message is not delivered to the application before the scheduler activates the application task. Thus, the final summand of the end-to-end delay is the *task activation delay* of the application at the receiver.

A near-zero task activation delay (i.e., only determined by the task switching overhead of the operating system) can be achieved by phase-aligning the activation of the application task with the termination of the preceding middleware task. In such a configuration, the application task is scheduled immediately after the middleware task has assembled the message out of the constituting packets. In case the application task is scheduled once within each TDMA round without phase-alignment, the task activation delay is between 0 and d_{round}.

Using the sum of the worst-case values of these six summands, the maximum end-to-end delay d_{max} can be computed as follows:

$$d_{\text{max}} = \underbrace{d_{\text{round}}}_{\text{sampling delay}} + \underbrace{\mathrm{d}_{\text{mw}}}_{\text{middleware delay at sender}} + \underbrace{\left\lfloor (n_{\text{queue}}-1)\cdot n_{\text{msg}} / n_{\text{ETregion}} \right\rfloor \cdot \mathrm{d}_{\text{round}}}_{\text{access delay}} + \underbrace{\left(\left\lceil (((n_{\text{queue}}-1)\cdot n_{\text{msg}} \bmod n_{\text{ETregion}}) + n_{msg}) / n_{\text{ETregion}} \right\rceil - 1\right)\cdot d_{\text{round}} + d_{\text{slot}}}_{\text{transmission delay}} + \underbrace{\mathrm{d}_{\text{mw}}}_{\text{middleware delay at receiver}} + \underbrace{d_{\text{activate}}}_{\text{task activation delay}}$$

The minimum end-to-end delay d_{min}, on the other hand, assumes the best-case values of the above summands. The sampling delay is 0, because the transmission request coincides with the sampling point. The best-case access delay is also 0, because the buffer at the ETCNI is empty at the instant of the transmission request. In addition, an immediate activation of the application task after the termination of the middleware results in an activation delay of 0.

$$d_{\text{min}} = \underbrace{0}_{\substack{\text{sampling}\\ \text{delay}}} + \underbrace{2\cdot \mathrm{d}_{\text{mw}}}_{\substack{\text{middleware delay at}\\ \text{sender and receiver}}} + \underbrace{0}_{\substack{\text{access}\\ \text{delay}}} + \underbrace{\left[\left(\left\lceil n_{\text{msg}} / n_{\text{etregion}} \right\rceil\right) - 1\right]\cdot \mathrm{d}_{\text{round}} + \mathrm{d}_{\text{slot}}}_{\substack{\text{transmission}\\ \text{delay}}} + \underbrace{0}_{\substack{\text{activation}\\ \text{delay}}}$$

IV. Computation for Exemplary Systems

This section exemplifies the analytical model for event-triggered overlay networks on top of a time-triggered physical network. The required parameters for the computation of the minimum and maximum end-to-end delays are taken from implementations of event-triggered overlay networks using TTP and TTE. In addition, we compare the analytical results with measurements from an experimental evaluation.

A. Overlay Networks on top of TTP

For this example, we have used parameters (see Figure 4) of a prototype implementation of event-triggered overlay networks on top of TTP [15]. This implementation complies to the communication model introduced in Section II. For establishing an overlay network for event-triggered communication, an overlay network middleware exploits a part of each TDMA slot provided by TTP.

The prototype implementation encompasses a cluster with four nodes interconnected by two redundant communication channels. The TTP communication schedule consists of four slots, each with a duration $d_{slot} = 80\,\mu s$ and transporting 64 bytes of user data. The bandwidth of the TTP network is 25 Mbps, while the TDMA round length is $320\,\mu$s.

Each node uses the operating system Linux Real-Time Application Interface (RTAI) and hosts an event-triggered application task. This tasks is activated by the middleware after

Description	Variable	TTP Implementation	DECOS TTE Implementation
slot duration	d_{slot}	80 µs	400 µs
TDMA round duration	d_{round}	320 µs	2 ms
middleware execution time	d_{mw}	32 µs	1 ms
size of an ET region	n_{ETregion}	64 bytes	128 bytes
message size	n_{msg}	14 bytes	14 bytes
queue length	n_{queue}	12 bytes	16 bytes
max. task activation delay	d_{activate}	10 µs	10 µs

Fig. 4. Parameters of Exemplary Event-Triggered Overlay Networks

the reception of a message from the event-triggered overlay network. The maximum task activation delay is $10\,\mu$s and determined by the task switching overhead of Linux RTAI on the PowerPC host processors of the nodes.

By filling in the values from Figure 4 into the generic formula for the end-to-end delay, the maximum end-to-end delay in the TTP-based implementation can be computed as follows:

$$\begin{aligned} d_{\text{max}} &= \underbrace{d_{\text{round}}}_{\text{sampling delay}} + \underbrace{\mathrm{d}_{\text{mw}}}_{\text{middleware delay at sender}} + \underbrace{\left\lfloor (n_{\text{queue}}-1)\cdot n_{\text{msg}} / n_{\text{ETregion}} \right\rfloor \cdot \mathrm{d}_{\text{round}}}_{\text{access delay}} + \underbrace{\left(\left\lceil (((n_{\text{queue}}-1)\cdot n_{\text{msg}} \bmod n_{\text{ETregion}}) + n_{msg}) / n_{\text{ETregion}} \right\rceil - 1\right)\cdot d_{\text{round}} + d_{\text{slot}}}_{\text{transmission delay}} + \underbrace{\mathrm{d}_{\text{mw}}}_{\text{middleware delay at receiver}} + \underbrace{d_{\text{activate}}}_{\text{task activation delay}} \\ &= 320\mu s + 32\mu s + 2\cdot 320\mu s + 80\mu s + 32\mu s + 10\mu s = 1114\mu s \end{aligned}$$

In analogy to the maximum end-to-end delay, the values from Figure 4 in conjunction with the generic formula yield the minimum end-to-end delay in the TTP-based implementation:

$$\begin{aligned} d_{\text{min}} &= \underbrace{0}_{\substack{\text{sampling}\\ \text{delay}}} + \underbrace{2\cdot \mathrm{d}_{\text{mw}}}_{\substack{\text{middleware delay at}\\ \text{sender and receiver}}} + \underbrace{0}_{\substack{\text{access}\\ \text{delay}}} + \underbrace{\left\lfloor\left(\left\lceil n_{\text{msg}} / n_{\text{etregion}} \right\rceil\right) - 1\right\rfloor\cdot \mathrm{d}_{\text{round}} + \mathrm{d}_{\text{slot}}}_{\substack{\text{transmission}\\ \text{delay}}} + \underbrace{0}_{\substack{\text{activation}\\ \text{delay}}} \\ &= 2\cdot 32\mu s + 80\mu s = 144\mu s \end{aligned}$$

The measurements performed in an experimental evaluation described in [15] confirm these analytical results. The minimum observed end-to-end delay has been $147\,\mu$s and the maximum observed end-to-end delay 1.1 ms. These delays match the values derived in the above equations.

B. Overlay Networks on top of TTE

The second example is an implementation of event-triggered overlay networks on top of a Time-Triggered Ethernet (TTE) network [8] with a bandwidth of 100 Mbps. This implementation is part of a prototype for the DECOS system architecture [6] and provides multiple overlay networks as a replacement for the physical networks in today's federated automotive electronic systems. Two event-triggered overlay networks with a bandwidth of 150 kbps serve for the comfort domain. An event-triggered overlay network with a bandwidth of 500 kbps handles the communication needs of the power-train domain. In addition, two time-triggered overlay networks with a bandwidth of more than 1 Mbps aim at multimedia and by-wire functions.

For the analysis of the end-to-end delays, we use the event-triggered overlay network (called PT) with a bandwidth of 500 kbps as an example. The right-most column in Figure 4 summarizes the parameters of overlay network PT that are significant for computing the end-to-end delays. The TTE-based implementation has been configured with a slot duration d_{slot} of 400 μs and a round duration d_{round} of 2 ms. The slot size of the considered overlay network PT is 128 bytes in order to achieve the required bandwidth of 500 kbps equivalent to a high-speed CAN network.

The middleware execution time of 1 ms is significantly longer compared to the event-triggered overlay network on top of TTP due to the provision of multiple overlay networks and additional services within the middleware (e.g., gateways to relay messages from one overlay network to another one). The maximum activation time of 10 μs is identical to the TTP-based implementation, because the same operating system (i.e., Linux RTAI) has been used.

By using the parameters in the generic formula, the following maximum end-to-end delay can be computed:

$$\begin{aligned} d_{\max} &= \underbrace{2000\mu s}_{\substack{\text{sampling}\\ \text{delay}}} + \underbrace{1000\mu s}_{\substack{\text{middleware delay}\\ \text{at sender}}} + \underbrace{1\cdot 2000\mu s}_{\substack{\text{access}\\ \text{delay}}} + \underbrace{400\mu s}_{\substack{\text{transmission}\\ \text{delay}}} + \underbrace{1000\mu s}_{\substack{\text{middleware delay}\\ \text{at receiver}}} + \underbrace{10\mu s}_{\substack{\text{task activation}\\ \text{delay}}} \\ &= 6410\mu s \end{aligned}$$

In analogy to the previous example, the minimum end-to-end delay results from a sampling delay of 0, an activation delay of 0, and the best-case access delay due to an empty queue.

$$d_{\min} = \underbrace{0}_{\substack{\text{sampling}\\ \text{delay}}} + \underbrace{2\cdot 1000\mu s}_{\substack{\text{middleware delay at}\\ \text{sender and receiver}}} + \underbrace{0}_{\substack{\text{access}\\ \text{delay}}} + \underbrace{400\mu s}_{\substack{\text{transmission}\\ \text{delay}}} + \underbrace{0}_{\substack{\text{activation}\\ \text{delay}}} = 2400\mu s$$

The measurements performed as part of the validation of the DECOS architectural services confirm these analytical results [16]. In case of a 100% load of the communication system, an end-to-end delay of 6.1 ms has been observed. Since the occurrence of the worst-case values in all summands (e.g., sampling delay, activation delay) of the formula for the maximum end-to-end delay is a rare event, the measured value is slightly below the computed end-to-end delay. More experiments with varying task activation times and communication loads will have to be performed in order to reach the computed maximum.

V. Conclusion

The extension of a time-triggered physical network with event-triggered overlay networks results in the combination of the respective advantages of time-triggered and event-triggered communication systems. Safety-critical application subsystems can rely on time-triggered communication, i.e., the periodic exchange of messages at predefined points in time. In safety-critical application subsystems, emphasis usually lies on the strengths of time-triggered control like predictability and determinism in order to facilitate fault-tolerance through active redundancy, certification, and rigorous validation. The event-triggered overlay networks aim at non safety-critical application subsystems that are implemented on the same distributed computer system as the safety-critical ones. For the non safety-critical application subsystems, the ability of requesting message transmissions on-demand at a priori unknown points in time improves flexibility.

When building new real-time applications or migrating existing event-triggered applications, knowledge about the delays for message transmissions on the event-triggered overlay networks is required. Therefore, this paper has introduced an analytical description for the delays of messages transmitted on an event-triggered overlay network. This result helps system engineers in determining whether an event-triggered overlay network meets the temporal requirements of their applications. By identifying significant parameters, which determine the delays of an event-triggered overlay network, we also show the effects of design decision, such as the selection of a particular communication schedule for the underlying time-triggered network. Thus, designers become aware of the design alternatives for achieving the required temporal properties of the event-triggered overlay network.

In order to exemplify the computation of the delays, we have also presented practical examples based on the protocols Time-Triggered Ethernet (TTE) and the Time-Triggered Protocol (TTP). These case studies have demonstrated that the analytical results can be applied to real-world systems. In addition, the examples have allowed a comparison of the analytical results with the delays measured in implementations of event-triggered overlay networks.

References

[1] Robert Bosch Gmbh, Stuttgart, Germany. *CAN Specification, Version 2.0*, 1991.

[2] J. Berwanger and M. Peller R. Griessbach. Byteflight a new protocol for safety critical applications. In *Proc. of the FISITA World Automotive Congress*, Seoul, 2000.

[3] Analysis of the european automotive in-vehicle network architecture markets. Technical report, Frost & Sullivan, October 2004.

[4] FlexRay Consortium. *FlexRay Communications System Protocol Specification Version 2.1*, May 2005.

[5] TTTech Computertechnik AG, Schönbrunner Strasse 7, A-1040 Vienna, Austria. *Time-Triggered Protocol TTP/C – High Level Specification Document*, July 2002.

[6] R. Obermaisser, P. Peti, B. Huber, and C. El Salloum. DECOS: An integrated time-triggered architecture. *e&i journal (journal of the Austrian professional institution for electrical and information engineering)*, 3:83–95, March 2006.

[7] R. Obermaisser and P. Peti. Realization of virtual networks in the decos integrated architecture. In *Proc. of the 14th Int. Workshop on Parallel and Distributed Real-Time Systems*, April 2006.

[8] H. Kopetz, A. Ademaj, P. Grillinger, and K. Steinhammer. The Time-Triggered Ethernet (TTE) design. *Proc. of 8th IEEE Int. Symposium on Object-oriented Real-time distributed Computing (ISORC)*, May 2005.

[9] J. Rushby. A comparison of bus architectures for safety-critical embedded systems. Technical report, Computer Science Laboratory, SRI International, September 2001.

[10] TTTech Computertechnik AG, Schönbrunner Strasse 7, A-1040 Vienna, Austria. *TTP/C Controller C2 Controller-Host Interface Description Document, Protocol Version 2.1*, November 2002.

[11] Freescale Semiconductor. MFR4200 datasheet FlexRay communication controllers. Technical report, August 2005.

[12] H. Kopetz. *Real-Time Systems, Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, Boston, Dordrecht, London, 1997.

[13] H. Kopetz and J. Reisinger. The non-blocking write protocol NBW: A solution to a real-time synchronisation problem. In *Proc. of the 14th Real-Time Systems Symposium*, 1993.

[14] K.H. Kim. Object structures for real-time systems and simulators. *IEEE Computer*, pages 62–70, August 1997.

[15] R. Obermaisser and P. Peti. Comparison of the temporal performance of physical and virtual CAN networks. In *Proc. of the IEEE Int. Symposium on Industrial Electronics*, Dubrovnik, Croatia, June 2005.

[16] DECOS. Dependable Embedded Components and Systems. project deliverable D4.2.2 – validation of the DECOS architecture. Technical report, 2006.