Networked Real-Time Embedded Systems 24

Haibo Zeng, Prachi Joshi, Daniel Thiele, Jonas Diemer, Philip Axer, Rolf Ernst, and Petru Eles

Abstract

This chapter gives an overview on various real-time communication protocols, from the Controller Area Network (CAN) that was standardized over twenty years ago but is still popular, to the FlexRay protocol that provides strong predictability and fault tolerance, to the more recent Ethernet-based networks. The design of these protocols including their messaging mechanisms was driven by diversified requirements on bandwidth, real-time predictability, reliability, cost, etc. The chapter provides three examples of real-time communication protocols: CAN as an example of event-triggered communication, FlexRay as

This work was done while Daniel Thiele was with Technische Universität Braunschweig

H. Zeng (✉) • P. Joshi
Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, VA, USA
e-mail: hbzeng@vt.edu; haibo.zeng@gmail.com; prachi@vt.edu

D. Thiele
Elektrobit Automotive GmbH, Erlangen, Germany
e-mail: daniel.thiele@elektrobit.com

J. Diemer
Symtavision, Braunschweig, Germany
e-mail: diemer@symtavision.com

P. Axer
NXP Semiconductors, Hamburg, Germany
e-mail: philip.axer@nxp.com

R. Ernst
Institute of Computer and Network Engineering, Technical University Braunschweig, Braunschweig, Germany
e-mail: ernst@ida.ing.tu-bs.de

P. Eles
Department of Computer and Information Science, Linköping University, Linköping, Sweden
e-mail: petru.eles@liu.se

S. Ha, J. Teich (eds.), *Handbook of Hardware/Software Codesign*,
DOI 10.1007/978-94-017-7267-9_25

a heterogeneous protocol supporting both time-triggered and event-triggered communications, and different incarnations of Ethernet that provide desired temporal guarantees.

Acronyms

ADAS	Advanced Driver Assistance System
AFDX	Avionics Full-Duplex Switched Ethernet
ARQ	Automatic Repeat Request
AVB	Audio/Video Bridging
CAN	Controller Area Network
CPA	Compositional Performance Analysis
CSMA/CD	Carrier Sense Multiple Access/Collision Detection
ECU	Electronic Control Unit
ET	Event-Triggered
FIFO	First-In First-Out
ILP	Integer Linear Program
LIN	Local Interconnect Network
MAC	Media Access Control
MOST	Media Oriented Systems Transport
QoS	Quality of Service
SPNP	Static-Priority Non-Preemptive
TSN	Time-Sensitive Networking
TT-CAN	Time-Triggered CAN
TTEthernet	Time-Triggered Ethernet
TTP	Time-Triggered Protocol
TT	Time-Triggered

Contents

24.1 Introduction

The communication network has been a key enabling technology for the advancement of distributed embedded systems, as evidenced in application domains like automotive, avionics, and industrial automation. With ever-increasing contents (safety, driver assistance, infotainment, etc.) in such systems relying on electronics and software, the supporting architecture is often integrated and interconnected by a complex set of heterogeneous data networks. For example, a modern automobile contains up to 100 Electronic Control Units (ECUs) and several heterogeneous communication buses (such as Controller Area Network (CAN), FlexRay, Media Oriented Systems Transport (MOST), and Local Interconnect Network (LIN)), exchanging thousands of signals [60].

This chapter intends to provide an overview of the real-time communication networks that are mainly categorized according to their messaging mechanism (or link layer protocol): Event-Triggered (ET), Time-Triggered (TT), or the coexistence of the two. In the first category (ET), messages are produced and transmitted based on the significant events occurring in the network, for example, the successful transmission of other messages. This category includes CAN, switched Ethernet, Avionics Full-Duplex Switched Ethernet (AFDX), and Ethernet Audio/Video Bridging (AVB). In the second category (TT), messages are transmitted at predefined time slots. Hence, it is essential to establish a global notion of time among all communication nodes to avoid possible collision. Time-Triggered CAN (TT-CAN) and Time-Triggered Protocol (TTP) belong to this category. In the third category, both time- and event-triggered traffics exist on the same communication network. Examples in this category include FlexRay, Ethernet Time-Sensitive Networking (TSN), and Time-Triggered Ethernet (TTEthernet).

The messaging mechanism in the communication network has a large impact on the temporal performance guarantees it can provide. Specifically, real-time performance metrics on the communication network include message *latency* and *jitter*. Message latency (or response time) is defined as the time that the message is ready for transmission to the time it is successfully received. Meeting the worst-case latency requirement is of fundamental importance to validate the system behavior against message deadlines. Jitter is the difference between the worst-case and best-case latencies, a key metric to the control performance of the features [44].

For time-triggered communication, the bandwidth is assigned to messages according to a time-triggered pattern, and the transmission of a message is triggered exactly when the global time reaches certain time points. Hence, a deterministic

Table 24.1 Summary of real-time communication protocols

Property	CAN	FlexRay	TTP	TT-Ethernet	AFDX	Switched Ethernet	AVB	TSN
Bandwidth	1 Mbps	10 Mbps	25 Mbps	1 Gbps	100 Mbps	100 Mbps–1 Gbps		
Message size (bytes)	0–8	0–254	0–240	46–1500	64–1518	42–1542		
Channels	1	2	2	2	2	1	1	2+
Messaging	ET	TT + ET	TT	TT + ET	ET	ET	ET	ET/ET + TT
Composability	No	Mixed	Yes	Yes	Yes	Yes	Yes	Yes
Medium access	CSMA/CA	TDMA + FTDMA	TDMA	TDMA + SPNP	SPNP + BAG	SPNP	SPNP + CBS	TDMA + SPNP
CRC error detection	15-bit	11-bit Header + 24-bit Tailer	24-bit	32-bit	32-bit	32-bit	32-bit	32-bit

communication pattern on the bus is guaranteed. The message latency/jitter can easily be calculated by checking the predefined transmission slots, which is independent from the rest of the network. On the contrary, in event-triggered communication, network delay occurs due to conflicts on media access and queuing mechanisms in the network. Event-triggered communication can be further classified by the temporal guarantees it can provide. In CAN, the dynamic segment of FlexRay, and switched Ethernet the temporal bounds on latency and jitter can oftentimes be analyzed a priori with its priority-based scheduling. However, changes to one node have an impact on the temporal behavior of messages from other nodes in the network. This makes the system composition difficult. As a remedy, several event-triggered communication protocols provide guaranteed service (bounds for latency and jitter) to a message regardless of the rest of the network, given that the message is produced at a speed that does not exceed the predefined maximum rate. Examples include the bandwidth allocation gap in AFDX and the rate-constrained messages in TTEthernet.

Table 24.1 summarizes the main communication protocols that are currently being deployed or considered in the application domains of real-time embedded systems. (Acronyms in the table: CSMA/CA, carrier sense multiple access with collision avoidance; TDMA, time division multiple access; FTDMA, flexible TDMA; SPNP, static priority non-preemptive; BAG, bandwidth allocation gap; CBS, credit-based shaping.) They all provide some degrees of predictability on temporal behavior. Those that are more suitable for non-real-time application, such as CSMA/CD-based Ethernet, are left out of the comparison. Besides the messaging mechanism and composability, the table also compares their maximum bandwidth, message size, number of (redundant) channels, medium access policy, and error detection.

The rest of the chapter describes in detail three different communication protocols, focusing on their messaging mechanism and timing predictability: CAN as an

example of event-triggered communication, FlexRay as a heterogeneous protocol supporting both time-triggered and event-triggered communications, and different incarnations of Ethernet that provide desired temporal guarantees.

24.2 Event-Triggered Communication: Controller Area Network

Controller Area Network is a broadcast digital bus designed to operate at speeds from 20 kbit/s to 1 Mbit/s, standardized as ISO/DIS 11898 [36]. The transmission rate depends on the bus length and transceiver speed. CAN is an attractive solution for embedded control systems because of its low cost, light protocol management, the deterministic resolution of the contention, and the built-in features for error detection and retransmission. Today CAN networks are widely used in automotive applications, as well as in factory and plant controls, robotics, medical devices, and some avionics systems.

Since its standardization in the mid-1990s, it has attracted a significant amount of research from the real-time systems community. The CAN protocol adopts a collision detection and resolution scheme, where the message to be transmitted is chosen according to its identifier. When multiple nodes need to transmit over the same bus, the lowest identifier message is selected for transmission. This arbitration protocol allows encoding the message priority into the identifier field and implementing priority-based scheduling.

24.2.1 CAN Message Format and Bus Arbitration

The CAN protocol has the message format of Fig. 24.1, where the sizes of the fields are expressed in bits. Priorities are encoded in the message identifier field (ID), which is 11 bits wide for the standard format. The identifier is used not only for bus arbitration but also for describing the data content (identification of the message stream). The CAN protocol requires that all contending messages have a unique identifier (unique priority). The other fields are the start-of-frame (SOF) bit; the control field (CTL) containing information on the type of message; the data field (DATA) containing the actual data to be transmitted, up to a maximum of 8 bytes; the checksum (CRC) used to check the correctness of the message bits; the acknowledge field (ACK) used to acknowledge the reception; the end-of-frame (EOF) delimiter used to signal the end of the message; and the idle space or inter-frame bits (IF) used to separate one frame from the following one.

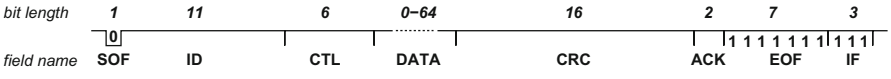

Fig. 24.1 The CAN data frame format

The CAN bus [36] essentially works as wired AND channels connecting all nodes. The two possible states encoded in the physical media are defined as "recessive" and "dominant," where dominant is a logical 0 (actively driven to a voltage by the transmitter) and recessive is a logical 1 (passively returned to a voltage by a resistor). The protocol allows multi-master access to the bus. At the lowest level, if multiple masters try to drive the bus state at the same time, the "dominant" configuration also prevails upon the "recessive."

The CAN arbitration protocol is both priority based and *non-preemptive*, as a message that is being transmitted cannot be preempted by higher-priority messages that are made available at the network adapters after the transmission has started. The media access protocol works by alternating contention and transmission phases. The time axis is divided into slots that must be larger than or equal to the time it takes for the signal to travel back and forth along the channel. The contention and transmission phases take place during the digital transmission of the frame bits.

At any time, if a node wishing to transmit finds the shared medium in an idle state, it waits for the end of the current bit (as defined by its internal oscillator) and then starts an arbitration phase by issuing a start-of-frame bit (a dominant bus state). At this point in time, each node with a message to be transmitted can start the competition to get access to the shared medium. All nodes will synchronize on the SOF bit edge and then, when the *identifier* field starts, they will serially transmit the identifier (priority) bits of the message they want to send, one bit for each slot, starting from the most significant ones. Collisions among identifier bits are resolved by the logical AND semantics, and each node reads the bus state while it transmits.

If a node reads its identifier bits on the medium without any change, it realizes it is the winner of the contention and is granted access to transmit the rest of the message. On the other hand, if one of the bits is changed when reading it back from the medium, this means that there is another node sending a higher-priority (dominant) bit; thus the message is withdrawn by the node that has lost the arbitration. The node stops transmitting the message bits and switches to listening mode only.

Clearly, according to this contention resolution protocol, the node with the lowest identifier is always the winner of the arbitration round and is transmitted next (the transmission stage is actually not even a distinct stage, given that the message frame is transmitted sequentially with the fields following the *Identifier*). All the other nodes switch to a listening mode. In a CAN system, at any time, there can be no two messages with the same identifier (this property is also referred to as *unique purity*). The message identifiers shall be assigned in a centralized fashion (e.g., by the system integrator). The arbitration is deterministic and also *priority* based, as that the message identifier gives in this case an indication of the message priority (the lowest identifier always wins).

24.2.2 Timing Analysis with Ideal Models

To predictably guarantee that the messages transmitting on CAN bus meet their deadlines, Tindell et al. [77] developed the seminal analysis of CAN message worst-case response times. The analysis is derived out of an analogy with CPU scheduling results but adapted to a message system scheduled by priority but without preemption. The result of the original paper influenced the design of on-chip CAN controllers like the Motorola msCAN and was included in the development of the early versions of Volcano's Network Architect tool. Volvo used these tools and the timing analysis results from Tindell's theory to evaluate communication latency in several car models, including the Volvo S80, XC90, S60, V50, and S40 [7]. The analysis was later found to be flawed by Davis et al. [12], where a set of formulas is provided for the exact evaluation and safe approximation of the worst-case message response times.

The analysis methods [12, 77] are based on a number of assumptions at the middleware, driver, and controller levels of the CAN communication stack, including

- The existence of a perfect priority-based software queue at each node for the outgoing messages
- The availability of one output buffer (i.e., transmit object, or simply TxObject) for each message, or preemptability of the TxObjects
- Immediate (zero-time) copy of the highest priority message from the software queue to the TxObjects

However, these idealized assumptions are often violated in practical systems (especially automotive systems) [15].

Under such analyses, each periodic or sporadic message m_i is defined by the tuple

$$m_i = \{id_i, T_i, J_i, C_i, D_i\}$$

where id_i is the CAN identifier, T_i is the period or the minimum inter-arrival time, J_i is the queuing jitter (sometimes also referred to as *release jitter*), C_i is the worst-case transmission time, and D_i is the deadline. The worst-case transmission time C_i is given by the total number of transmitted bits (including the worst-case stuffed bits) divided by the bus transmission rate.

The schedulability analysis [12] starts with the theorem that the worst-case response time is always inside the busy period. The busy period of priority level-i is a contiguous interval of time that starts at the critical instant. During the busy period, any message of priority lower than m_i is unable to start transmission and win arbitration. For m_i, the *critical instant* is the time instant where (1) the contention

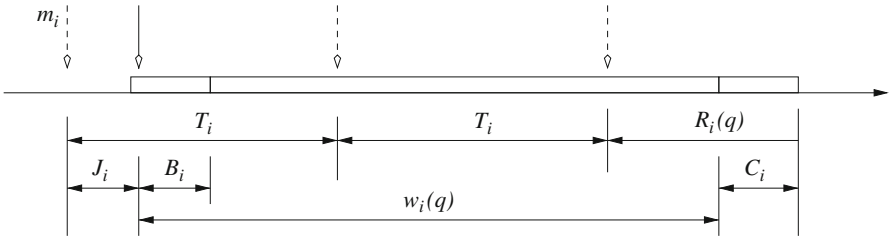

Fig. 24.2 The worst-case busy period and response time of message m_i

on the bus has just won by the longest lower-priority message (if one exists) and (2) all the higher-priority messages $hp(i)$ become simultaneously ready and arrive at their maximum rates thereafter.

Figure 24.2 illustrates the analysis of the worst-case response time for m_i. The dashed downward arrows denote the arrivals of m_i, separated by at least T_i. The solid downward arrow is the queuing time of m_i. The busy period consists of the queuing jitter, the blocking time, the queuing delay, and the transmission time of m_i itself.

To find the correct worst-case response time, the formula to be applied is a small modification to [77] that checks all the instances of message m_i transmitted inside the busy period. The response time of the q-th instance in the hyperperiod is

$$R_i(q) = J_i + w_i(q) - (q-1)T_i + C_i \tag{24.1}$$

where q ranges from 1 to the last instance $q_i^{\max}$ of m_i inside the busy period. The *blocking time*, i.e., the time spent on waiting for the transmission of a lower-priority message already on the bus when m_i becomes ready, is denoted as B_i. The worst-case queuing delay $w_i(q)$ for the q-th instance in the busy period is

$$w_i(q) = B_i + (q-1)C_i + \sum_{k \in hp(i)} \left\lceil \frac{w_i(q) + J_k + \tau_{bit}}{T_k} \right\rceil C_k \tag{24.2}$$

where τ_{bit} is the time to transmit one bit of data on the bus.

In Eq. (24.2), $w_i(q)$ appears on both sides. However, the right hand side is a monotonic nondecreasing function of $w_i(q)$. Hence, $w_i(q)$ can be solved using the iterative procedure defined in the equation below.

$$w_i^{n+1}(q) = B_i + (q-1)C_i + \sum_{k \in hp(i)} \left\lceil \frac{w_i^n(q) + J_k + \tau_{bit}}{T_k} \right\rceil C_k \tag{24.3}$$

The calculation can start with an initial value of $w_i^0(q) = B_i + (q-1)C_i$ and stop if $w_i^{n+1}(q) = w_i^n(q)$ or when $J_i + w_i^{n+1}(q) - (q-1)T_i + C_i > D_i$, the latter condition indicating that m_i is unschedulable.

The length of the longest busy period L_i and the index of the last instance are calculated as

$$\begin{cases} L_i = B_i + \sum_{k=hp(i)\bigcup\{i\}} \left\lceil \frac{L_i + J_k + \tau_{bit}}{T_k} \right\rceil C_k \\ q_i^{\max} = \left\lceil \frac{L_i + J_i}{T_i} \right\rceil \end{cases} \tag{24.4}$$

The worst-case message response time is the maximum among all its instances in the busy period.

$$R_i = \max_{q=1,\ldots,q_i^{\max}} \{R_i(q)\} \tag{24.5}$$

The above analysis (Eq. (24.1), (24.2), (24.3), (24.4), and (24.5)) requires to consider all the $q_i^{\max}$ instances in the busy period. Under the assumption that *the deadline is no greater than the period* ($D_i \leq T_i$), a sufficient but not necessary condition can be derived [12], which only checks the schedulability of the first instance by using its response time upper bound. Davis et al. [12] define the concept of *push through interference*, the largest interference that can be pushed through into the next period of message m_i due to the non-preemptive transmission of the previous instances. Any instance of m_i is subject to either direct blocking from lower-priority messages or indirect blocking from the previous instances of m_i, upper bounded by the push through interference of at most C_i. Hence, the queuing delay of the first instance ($q = 1$) of m_i is bounded by the result of the following equation:

$$w_i = \max(B_i, C_i) + \sum_{k\in hp(i)} \left\lceil \frac{w_i + J_k + \tau_{bit}}{T_k} \right\rceil C_k \tag{24.6}$$

which can be solved in a similar way to Eq. (24.2).

Besides the above analysis technique [12, 77], the schedulability of CAN messages is also addressed with other methods. Sufficient schedulability tests based on system utilization bound are derived for CAN (or in general, for systems under non-preemptive fixed-priority scheduling) with deadline monotonic and rate monotonic priority assignment policies [1, 6]. Navet et al. [62] introduce the concept of worst-case deadline failure probability because of transmission errors. In [5], Broster et al. present the probabilistic analysis of the impact of faults on CAN message latencies in the worst-case scenario. In [64], Nolte et al. extend the worst-case response time analysis using random message transmission times that take into account the probability of a given number of stuff bits. These works [5, 62, 64] still perform the analysis with respect to the *critical instant*, i.e., the worst-case response time scenario. Zeng et al. [81] provide a stochastic analysis framework for CAN message response times. In another work [82], they build a probability mix model to predict the distribution of message response times in CAN. The model is a mixture of the gamma distribution and the degenerate distribution. In [45], Liu et al. present an

extreme value theory-based statistical method to estimate the worst-case response times of CAN messages.

The analysis in [12, 77] has also been extended to other models of CAN messages. [8, 79] studied the worst-case response time for CAN messages that are scheduled with offsets. Mubeen et al. [51] consider the analysis for mixed messages, i.e., those with simultaneous time triggered (periodic) and event triggered (sporadic). They further provide analysis method to support mixed messages that are scheduled with offsets [55, 56]. In addition, the analysis on mixed messages is extended for nonideal CAN networks, including those where some nodes implement FIFO queues [54] or with CAN controllers implementing abortable [53] and non-abortable [52] transmit buffers. Liu et al. [46, 47] consider messages under the multi-frame task model [50] and its generalization [4] and present a sufficient schedulability analysis for systems with mixed message queues.

24.2.3 Analysis with Non-idealized Models

The analysis techniques in Sect. 24.2.2 assume an idealized model for the message queuing and the configuration and management of the peripheral TxObjects. In reality, CAN controllers have a limited number of available transmit buffers (TxObjects). When the number of TxObjects available at the controller is smaller than the number of messages sent by the node (as is the case for automotive gateways and nodes with lots of output information, or when message reception is polling based and a relatively large number of buffers must be reserved to input streams in order to avoid message loss by overwriting), it is necessary to use a *software queue* to hold messages waiting to be copied to a TxObject. Several commercial drivers (including those from Vector [78], probably the most commonly used in automotive systems) allow to put the outgoing messages in software queues as a temporary storage for accessing TxObjects. It is also quite common to use multiple queues, with each queue linked to a single TxObject. When a TxObject is available, a message is extracted from the queue and copied into it.

When software queues are used, the preservation of the priority order of the messages for accessing the CAN bus requires the following:

- **(a)** The messages in the software queue must be sorted by their priority (i.e., by message identifier).
- **(b)** When a TxObject (transmit buffer) becomes free, the highest priority message in the queue is immediately extracted and copied into the TxObject. In addition, messages in the TxObjects must be sent in the order of their CAN identifier.
- **(c)** If a higher-priority message becomes ready and all the TxObjects are used by lower-priority messages, the lowest-priority message in one of the TxObjects must be evicted and put back in the queue to ensure that a TxObject is available for the highest priority message.

When any of these conditions do not hold, priority inversion occurs, and *the worst-case timing analysis* in [12] is not necessarily safe. These issues are discussed in [14, 16] where the impact of transmission by polling (as opposed to interrupt driven) is also outlined. Using FIFO or any work-conserving queuing for messages inside the CAN driver/middleware layers (violating **(a)** in the previous list) is discussed and analyzed in [11, 13, 57]. When the copy time from the message queue to the TxObject cannot be neglected (disobeying **(b)** in the list), the introduced priority inversion is analyzed in [40]. Di Natale and Zeng [15] provides theory for the analysis of systems in which message output at the CAN driver is performed by polling (another break of the rule **(b)**). For the violation to **(c)** in the list, additional delay can be caused by limited number of TxObjects at the CAN controller that are non-abortable. Natale [41] and Khan et al. [61] provide an analysis to these driver configuration issues and controller policies that can lead to (possibly multiple) priority inversions. Di Natale and Zeng [15] provides further insight on the management of TxObjects without preemption and proposes a heuristic for the design of message queuing systems with guaranteed real-time properties. Finally, [58, 59] integrate the effect of these hardware and software limitations with messages that are triggered by both time (periodic) and event (sporadic).

As an example, the analysis for systems with FIFO software queue [13], under the condition that the messages have constrained deadlines ($D \leq T$), is summarized below. The more general case of unconstrained deadlines is discussed in [11]. For a message m_i, the maximum time that it waits in the FIFO queue before it becomes the oldest message (hence ready for priority-based arbitration) is defined as the *buffering delay* of m_i. The buffering delay of m_i is denoted as f_i.

The analysis returns the same worst-case response time bound for all messages $\mathbf{M}$ in the same FIFO queue. It is similar to Eq. (24.6) in that the blocking time is safely bounded to avoid checking multiple instances in the busy period. For each contributing delay to the response time, it makes pessimistic but safe assumptions to derive a correct upper bound. The *first* delay is the blocking time, upper bounded by the maximum between direct blocking time from lower-priority messages or the indirect blocking bounded by the push through interference $C_{\mathbf{M}}^{\max} = \max_{j \in \mathbf{M}} C_j$. The *second* is the interferences from messages from the same FIFO queue. Since messages have a deadline no larger than the period, there can be at most one instance from each message waiting in the queue for a schedulable system. Hence, the maximum interference caused by these messages is upper bounded by $C_{\mathbf{M}}^{\text{SUM}} - C_{\mathbf{M}}^{\min}$ where $C_{\mathbf{M}}^{\text{SUM}} = \sum_{j \in \mathbf{M}} C_j$ and $C_{\mathbf{M}}^{\min} = \min_{j \in \mathbf{M}} C_j$. In this way, among the total transmission time $C_{\mathbf{M}}^{\text{SUM}}$, the maximum amount is also exposed to interference from messages in other queues when m_i has a transmission time $C_i = C_{\mathbf{M}}^{\min}$. The *third* delay is the interferences from messages in other queues. This delay is maximized when we consider the lowest-priority message $m_{L_i} \in \mathbf{M}$ queued in the same FIFO as m_i.

Summating all the above, the queuing delay w_i can be derived from a sufficient condition similar to that in Eq. (24.6) for priority-queued messages:

$$w_i = \max(B_{L_i}, C_{\mathbf{M}}^{\max}) + (C_{\mathbf{M}}^{\mathrm{SUM}} - C_{\mathbf{M}}^{\min}) + \sum_{k \in hp(L_i) \wedge k \notin \mathbf{M}} \left\lceil \frac{w_i + J_k + f_k + \tau_{bit}}{T_k} \right\rceil C_k \tag{24.7}$$

and the response time of m_i is bounded by adding the queuing delay and the transmission time together:

$$R_i = w_i + C_{\mathbf{M}}^{\min} \tag{24.8}$$

In Eq. (24.7), besides the queuing jitter J_k, the buffering delay f_k should be treated as an additional jitter. This is because f_k quantifies the variation from the readiness of m_k to the time it enters the priority-based arbitration (and becomes capable of interfering m_{L_i}). The buffering time f_i of m_i can be bounded as

$$f_i = R_i - C_{\mathbf{M}}^{\min} \tag{24.9}$$

24.3 A Heterogeneous Communication Protocol: FlexRay

24.3.1 Introduction

The FlexRay standard was developed by a consortium including the major car manufacturers and their suppliers, with a stated objective *to support cost-effective deployment of distributed by-wire controls*. It is now defined in a set of ISO standards, ISO 17458-1 to 17458-5 [37]. In addition to the stringent requirements on determinism and short latencies as those for the x-by-wire functions, the definition of FlexRay also was motivated by the large volumes of data traffic from active safety functions.

The upper bound of communication speed in FlexRay is defined as 10 Megabits per second (Mbps), as opposed to 1 Mbps for CAN. Time is divided into communication cycles that are of equal length. Each communication cycle contains up to four segments: *static*, *dynamic*, *symbol window* (to transmit FlexRay-defined symbols for, e.g., maintenance and cold-start cycles), and *network idle time (NIT)* (for clock correction due to, e.g., clock drift), as shown in Fig. 24.3. Clock synchronization is embedded in the standard, using part of the NIT segment.

The *static* segment of the communication cycle enables the transmission of time-critical messages according to a periodic pattern, i.e., with *time-triggered* (TDMA) communication. It is divided into a set of equal-sized time *slots*. The transmission of frames in the static segment is fixed in a given slot, at a given time window. The *dynamic* segment allows for flexible communications. The transmission of frames in the dynamic segment is *event triggered*, arbitrated by their identifiers, where the lowest identifier frames are transmitted first. Frames from different nodes

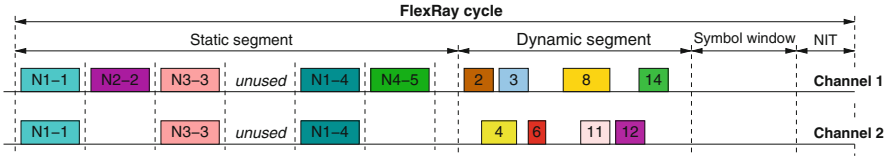

Fig. 24.3 The four segments in a FlexRay communication cycle

can share the same identifier, but they differ in the allowed communication cycle. This flexibility, called *slot multiplexing*, is supported in the most recent FlexRay standard [37]. For increased reliability and timing protection, FlexRay includes specifications of dual channel as well as bus guardians at the node and star level. In a dual-channel configuration, frames for safety-critical communications can be replicated in both channels (as those from node N1 in Fig. 24.3), or the slots can be assigned independently.

The FlexRay bus configuration includes the selection of several parameters, including the length of the communication cycle, the number and length of time slots in the static segment, and the slot time of the dynamic segment. There are several issues that require careful consideration in the definition of the bus configuration: future extensibility, the composability of subsystems, and the possible standardization for reusing of ECU components. Due to the nature of the automotive supply chain and the desire to reuse ECUs on different car platforms, there is a trend of global standardization of these FlexRay bus configuration parameters.

24.3.2 Static Segment

In FlexRay static segment, each node keeps the specification of the time slots for its outgoing and incoming communications in its *local scheduling table*. The local scheduling tables of all nodes shall be consistent (i.e., no two nodes are scheduled to send frames in the same slot of the same communication cycle). In this way, the schedule is *composable* (in the sense that no timing conflicts or interferences arise), each node executes with respect to its own (synchronized) clock, and there is no need for storing a global scheduling table.

Slots that are left free in the (virtual) global table resulting from the composition of the local tables can be used for future extensions. Bus guardians monitor and prevent a node from transmitting outside the allocated time window. This guarantees time protection and isolation from timing faults.

The scheduling of FlexRay systems consists of the scheduling of the task and signal instances in an application cycle. Broadly speaking, there are two possible synchronization patterns between tasks and signals:

- **Asynchronous scheduling.** Such a scheduling model does not require that the job (an instance of the task that produces the signal) and signal schedulers are synchronized. Jobs post data values for the output signals in shared variables.

The communication drivers have the responsibility to pack the signals into frames and fill the registers for the outgoing communication slot. At the receiving side, the received frames are de-packed and written into input registers such that they can be read asynchronously by the reader tasks.

- **Synchronous scheduling.** Different from the asynchronous scheduling model, job executions and frame transmissions are synchronized such that a job must complete before the beginning of the slot that transmits its output signal (with a margin determined by the necessary copy time). It is then necessary to know what job produces the data that is transmitted by a frame and what is the job that reads the data delivered by the frame. It leverages the full benefits of the FlexRay deterministic communication: Scheduling can be arranged to achieve small sampling delays, providing very tight end-to-end latencies and small jitter. Also, equally important, this scheduling model allows to guarantee time determinism and the preservation of the stream of data exchanged over the bus [27].

For synchronous scheduling of signals and tasks, besides packing the signals into frames, designers will need to schedule the software tasks and frames, such that timing constraints including end-to-end deadlines are satisfied. The precedence constraints induced by information passing between tasks and signals and the end-to-end delays associated with control path should be taken into consideration. ILP-based approaches, holistic or two-stage, are studied by Zeng et al. [80], where the tasks are scheduled at job level (each job in the hyperperiod can be scheduled independently). Lukasiewycz et al. [49] provide a framework for scheduling buses and ECUs at the task level, to enable an AUTOSAR compliant system. Besides the timing-related metrics (bandwidth, end-to-end latency), synchronous scheduling is also addressed under other contexts, such as application-level acknowledgment and retransmission scheme [43], robustness to uncertainties in design parameters [24], and message authentication/verification for security enhancement [28]. Also, Han et al. [27] discuss that system-level time-triggered schedules allow the semantics-preserving implementation of distributed control models with a synchronous reactive semantics and develop algorithms for minimizing functional delays to improve control quality.

When considering the asynchronous scheduling, the FlexRay scheduling problem consists in the optimization of the communication scheduling for a set of periodic signal streams, generated at the ECU interface and considered independently from their sender and receiver tasks. For several car manufacturers this is a problem of high practical interest, because the first step in the move to FlexRay is likely to be the remapping of existing CAN communication flows, which are today typically asynchronous with respect to computations. This problem is addressed in a number of papers as follows:

Ding et al. adopt genetic algorithm and its combination with bin packing [22,23]. Schmidt et al. present a two-stage Integer Linear Program (ILP) approach [67], where the first stage packs signals to frames and the second stage determines the schedule from the set of frames. Grenier et al. [26] design a simple heuristic

assuming one signal per frame. Lukasiewycz et al. [48] develop a bin-packing-based approach for allocating signals to slots. Tanasa et al. [69] propose to use message retransmissions on the FlexRay static segment to provide guarantees on reliability. These works focus on the earlier version of FlexRay 2.1 where slot multiplexing is disallowed, hence preventing the share of the same slot across different ECUs. In compliance with the latest FlexRay 3.0 standard, Schenkelaars et al. [66] consider the mapping of frames to slots but assume the signals are already packed to frames. Kang et al. [39] consider the problem of packing signals to frames. Hu et al. [30] adopt a list-scheduling-based approach. Darbandi et al. [9] transform the problem to a strip packing problem. They also propose an ILP-based approach for a direct packing of signals to slots [10].

In the following, the ILP-based approach for synthesizing asynchronous schedule is described. The approach extends the formulation from [48] by considering the slot multiplexing allowed in FlexRay 3.0.

24.3.2.1 ILP-Based Approach for Asynchronous Scheduling

For asynchronous scheduling, the problem the designers are facing is to pack the signals into frames and assign frames to slots such that there is one instance of the signal transmitted within its period, and the number of used slots (the used bandwidth) is possibly minimized.

For the purpose of scheduling in the static segment, it is sufficient to consider the following set of design parameters in the FlexRay bus configuration: (cc, ns, ls), where cc is the number of communication cycles, ns is the number of slots in the static segment of the communication cycle, and ls is the length of the slot in bytes. The application contains a set of ECUs $\mathbb{E}$, each ECU e sending a set of signals $\mathbb{M}_e$. The set of all signals is denoted as $\mathbb{M}$. For each signal $m \in \mathbb{M}$, it is configured with a tuple of parameters (l_m, r_m), where l_m is the length of m in bytes and r_m is the cycle repetition of m. The cycle repetition r_m is essentially the period T_m in the unit of communication cycle: $r_m = T_m / l_{\text{comm}}$, where l_{comm} is the length of the communication cycle in time. r_m shall always be an integer divisor of the number of FlexRay communication cycles cc.

Another option is to evaluate several possible FlexRay configurations in an initial branching of the search procedure. If the number of possible configurations is not very large, it should be possible to explore them and run the optimization framework as an inner loop, comparing the results at the end and choosing the best one with respect to the objective function.

The mapping of signals to slots is encoded in a set of binary variables. For signal m, the base cycle b_m is smaller than r_m ($b_m \in \{0 \ldots r_m - 1\}$); hence the signal to slot mapping is defined as

$$\begin{aligned} &\forall c = 0, \ldots, r_m - 1, \forall s = 0, \ldots, ns - 1, \\ &B_{m,c,s} = \begin{cases} 1, \text{ if } m \text{ is mapped to base cycle } c, \text{ slot } s \\ 0, \text{ otherwise} \end{cases} \end{aligned} \tag{24.10}$$

Another set of binary variables encodes the status of each slot:

$$U_s = \begin{cases} 1, \text{ if slot } s \text{ is used} \\ 0, \text{ otherwise} \end{cases} \tag{24.11}$$

A third set of binary variables encodes the ownership of a slot in a communication cycle:

$$O_{e,c,s} = \begin{cases} 1, \text{ if slot } s \text{ in cycle } c \text{ is owned by ECU } e \\ 0, \text{ otherwise} \end{cases} \tag{24.12}$$

The problem now can be formulated as follows. For the set of constraints, each signal is mapped to one and only one slot in its cycle repetition:

$$\forall m \in \mathbb{M}, \quad \sum_{c=0}^{r_m-1} \sum_{s=0}^{ns-1} B_{m,c,s} = 1 \tag{24.13}$$

The sum of the payloads over all mapped signals within each slot will be upper bounded by the slot size:

$$\forall c = 0, \ldots, cc-1, s = 0, \ldots, ns-1, \quad \sum_{m,\ b \equiv c \pmod{r_m}} l_m \times B_{m,b,s} \leq ls \tag{24.14}$$

If signal m is mapped to communication cycle c and slot s, then m's source ECU e must own that slot:

$$\forall m \in \mathbb{M}_e, c = 0, \ldots, cc-1, b \equiv c \pmod{r_m}, s = 0, \ldots, ns-1, \quad B_{m,b,s} \leq O_{e,c,s} \tag{24.15}$$

If no signal is mapped to slot s in any communication cycle, then the slot ownership is set to null:

$$\forall e \in \mathbb{E}, c = 0, \ldots, cc-1, s = 0, \ldots, ns-1, \quad O_{e,c,s} \leq \sum_{m \in \mathbb{M}_e,\ b \equiv c \pmod{r_m}} B_{m,b,s} \tag{24.16}$$

Each slot in each cycle can only be owned by one ECU:

$$\forall c = 0, \ldots, cc-1, s = 0, \ldots, ns-1, \quad \sum_{e \in \mathbb{E}} O_{e,c,s} \leq 1 \tag{24.17}$$

The slot is used if any of the ECUs owns it:

$$\forall s = 0, \ldots, ns-1, \quad \sum_{e \in \mathbb{E}} \sum_{c=0}^{cc-1} O_{e,c,s} \leq U_s \tag{24.18}$$

The objective is to minimize the number of used slots:

$$\min \sum_{s} U_s \tag{24.19}$$

24.3.3 Dynamic Segment

The FlexRay dynamic segment is partitioned into a number of minislots (MS) that are of equal length. Each frame is assigned a frame identifier (FrameID, or simply ID) within which it can transmit. The dynamic segment is arbitrated in the following way. At the beginning of the dynamic segment, the minislot index is set to one. If there is a ready frame with ID that matches the current slot index, the frame is transmitted. Correspondingly, the dynamic slot is extended to a length equal to the number of minislots needed to transmit the frame, plus one minislot for idle phase (used to separate frame transmissions). Otherwise, the minislot elapses without frame transmission, and the dynamic slot index is incremented before the next minislot starts.

To make sure there is enough time to transmit a frame before the end of the dynamic segment, a parameter $pLatestTx$ is specified for each ECU as the number of minislots in the dynamic segment minus the largest frame size (in minislots). If the current minislot index is larger than $pLatestTx$ (LTx for short in the following), then no frame can start transmission, and all the ready frames are delayed to the next communication cycle.

In the example of Fig. 24.4, five dynamic frames are transmitted over a FlexRay channel. Two frames, m_2 and m_4, share the same ID. At the beginning of the dynamic segment, the dynamic slot index is initialized as 1, and the controller checks whether there is a frame with FrameID 1 ready to be transmitted. m_1, and, consequently, m_2 and m_3, is transmitted in the first communication cycle. However, m_5 with ID 5 cannot be transmitted since there is not enough room to accommodate it and its transmission is delayed to the next cycle (assuming it is allowed to transmit there). In the second communication cycle, there is no frame with ID 1; thus the first dynamic slot is collapsed to one minislot and m_4, which cannot be transmitted

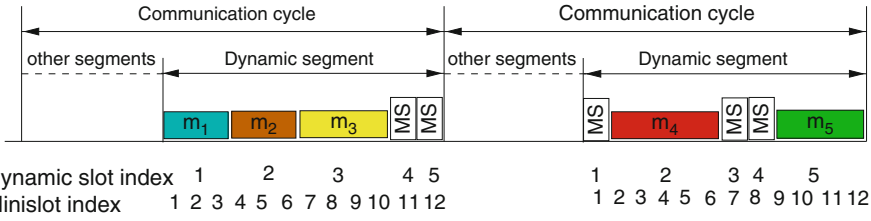

Fig. 24.4 An example of FlexRay dynamic frame transmission

in the first communication cycle because of the transmission of another frame m_2 with the same ID 2, is transmitted, followed by a minislot for indexes 3 and 4 and the transmission of m_5 with ID 5.

For the purpose of scheduling and analysis in the dynamic segment, the FlexRay bus configuration is captured with a list of parameters $(l_{\text{comm}}, l_{\text{ST}}, n_{\text{MS}}, l_{\text{MS}})$, where l_{comm} is the length of the FlexRay communication cycle, l_{ST} is the length of the static segment, n_{MS} is the number of minislots in the dynamic segment, and l_{MS} is the length of the minislot in time (the length of the dynamic segment is $l_{\text{DYN}} = n_{\text{MS}} \cdot l_{\text{MS}}$). In the design flows in the automotive industry, these parameters (in particular the communication cycle and the slot size) are often defined and standardized based on the need to reuse legacy components and standardize configurations, which is likely to induce carmakers to freeze the definition of these parameters for their product lines.

A set of $CC^{\max}$ (of value that is a power of 2, i.e., 1, 2, 4, 8, 16, 32, 64) communication cycles constitute a hyperperiod, for which the scheduling table is specified and repeated. Each dynamic frame $m_i \in \mathbb{M}$ is characterized by a tuple of parameters $\{N_i, T_i, J_i, D_i, C_i\}$, where N_i is its source ECU, T_i is its period, J_i denotes its release jitter (the maximum delay with respect to the periodic activation event), D_i is the deadline, and C_i is the transmission time (the time m_i occupies when transmitted on the bus, including the one minislot needed for the idle phase). For convenience, C_i is defined in terms of the (integer) number of l_{MS}. For example, $C_i = 5 \cdot l_{\text{MS}}$ for frame m_4 in Fig. 24.4. The deadline D_i is assumed to be arbitrary, i.e., it can be smaller, equal, or lager, compared to T_i. The worst-case response time R_i of m_i is the maximum difference between the finish time and the arrival time. For each m_i, LTx_i indicates the latest minislot index in which it is allowed to start transmission. The set of frames with lower ID than m_i is denoted as $lf(m_i) = \{m_j | ID_j < ID_i\}$. Also, the set of frames with lower ID than m_i plus itself is denoted as $le(m_i) = lf(m_i) \bigcup \{i\}$.

In the latest FlexRay 3.0 standard [37], *slot multiplexing* is added as a new feature, which allows different frames sent in different cycles to share the same ID. With slot multiplexing, each frame m_i is assigned with two attributes, base cycle b_i and cycle repetition r_i. Then m_i can only transmit in every r_i-th communication cycle starting at the b_i-th cycle. r_i only assumes a value as a power of 2 that is no greater than $CC^{\max}$, and $b_i \in \{0 \ldots r_i - 1\}$. As a special case, if m_i exclusively occupies the slot (i.e., m_i is the only message allowed to transmit in its slot for all communication cycles), like those without slot multiplexing, and then $b_i = 0$ and $r_i = 1$. Obviously, to avoid infinite waiting for m_i, it shall be $T_i \geq r_i \times l_{\text{comm}}$.

Since the scheduling in FlexRay dynamic segment is not work conserving, i.e., the bus may be left idle even if a message is ready, its timing analysis is inherently more difficult compared to other fixed-priority-based protocols like CAN. *First*, even if there is no message to be sent in a particular slot, it will occupy one minislot. *Second*, if more than one instance of a message is ready, only one of them can be sent in one dynamic segment. *Third*, a message may only be ready after its slot has

started. *Fourth*, there are not enough minislots to transmit a message as its LTx has elapsed. *Lastly*, due to slot multiplexing, the pending messages may not be transmitted in the current communication cycle.

In the following, the timing analysis is discussed in two cases: with or without slot multiplexing. The case of no slot multiplexing is first discussed because of its relative simplicity.

24.3.3.1 Timing Analysis Without Slot Multiplexing

When the feature of slot multiplexing is not used, i.e., the parameters $b_i = 0$ and $r_i = 1$ for all frames m_i, it essentially means that a frame can be transmitted in any cycle.

FlexRay dynamic frames are transmitted in the order of the IDs (priorities) of ready frames and without preemption: If a frame becomes ready while another frame with higher ID is being transmitted (after its slot passed), it cannot preempt the lower-priority frame and must wait until the next cycle. Like other systems with non-preemptive fixed-priority scheduling (such as CAN), the response time analysis is based on the calculation of the *busy period* of level-i, denoted as t_i. The busy period is the worst-case time interval that starts from the critical instant for an instance of m_i queued at $t = 0$ with jitter J_i, where the bus is always busy transmitting frames with priority higher than or equal to m_i except for a possible initial blocking B_i.

Here, B_i denotes the longest blocking delay that happens in the communication cycle when m_i becomes ready. In the worst case, the slot with index i starts as soon as possible in the cycle where no frame with ID lower than i is sent, and m_i is queued right after that. Thus, the worst-case blocking B_i is

$$B_i = l_{\text{comm}} - (l_{\text{ST}} + (ID_i - 1)l_{\text{MS}}) \tag{24.20}$$

Also, for $m_j \in lf(m_i)$, $n_j^{(k)}$ denotes the maximum number of instances of m_j activated in the busy period for the k-th iteration $t_i^{(k)}$. $n_i^{(k)}$ is the number of instances of m_i activated in the busy period (before the one considered for the analysis). The vector $\mathbf{n}^{(k)}$ is defined as $\mathbf{n}^{(k)} = \{n_j^{(k)}\}, j \in le(m_i)$. The function $f_i(\mathbf{n}^{(k)})$ gives the worst-case interference caused by the static segment and the transmission of dynamic frames $m_j \in le(m_i)$ (each activated $n_j^{(k)}$ times). The computation for $f_i(\mathbf{n}^{(k)})$ is discussed in the later part of the section.

The overall length of the busy period can be found as the fixed-point solution of the iterative procedure:

$$\begin{cases} t_i^{(0)} = B_i \\ t_i^{(k+1)} = B_i + C_i + f_i(\mathbf{n}^{(k)}) \end{cases} \tag{24.21}$$

For the iterations in the computation of the busy period t_i, $\mathbf{n}^{(k)}$ is

$$\begin{cases} n_j^{(k)} = \left\lceil \dfrac{J_j + t_i^{(k)}}{T_j} \right\rceil, \forall j \in lf(m_i) \\ n_i^{(k)} = \left\lceil \dfrac{J_i + t_i^{(k)}}{T_i} \right\rceil - 1 \end{cases} \tag{24.22}$$

Inside the busy period t_i, up to $q_i^{\max}$ instances of m_i are ready for transmission:

$$q_i^{\max} = \left\lceil \frac{J_i + t_i}{T_i} \right\rceil \tag{24.23}$$

The worst-case response time of m_i is the maximum among those $q_i^{\max}$ instances in the busy period. These instances are indexed as $q = 1, \ldots, q_i^{\max}$. The longest time from the start of the busy period to the time the q-th instance starts transmission is calculated by the following iterative formula:

$$\begin{cases} w_i^{(0)}(q) = B_i \\ w_i^{(k+1)}(q) = B_i + f_i(\mathbf{n}^{(k)}) \end{cases} \tag{24.24}$$

where $\mathbf{n}^{(k)}$ is given as

$$\begin{cases} n_j^{(k)} = \left\lceil \dfrac{w_i^{(k)}(q) + J_j}{T_j} \right\rceil, \forall j \in lf(m_i) \\ n_i^{(k)} = q - 1 \end{cases} \tag{24.25}$$

The response time of the q-th instance is

$$R_i(q) = J_i + w_i(q) - (q-1)T_i + (C_i - l_{\mathrm{MS}}) \tag{24.26}$$

where l_{MS} is subtracted from C_i as a frame is considered received before the idle phase (whose length is one minislot).

Go back to the computation of $f_i(\mathbf{n}^{(k)})$, the function computing the worst-case delay caused by frames in $le(m_i)$ with known vector $\mathbf{n}^{(k)}$ of activated instances. Since each previous instance of m_i delays the transmission of the following instance of m_i by at least one cycle, their occurrences will produce a delay equal to at least $n_i^{(k)}$ communication cycles. Consider the frames in $lf(m_i)$. Let $s_{\mathrm{cycle}}^{(k)}$ be the number of communication cycles the $(n_i^{(k)} + 1)$-th instance of m_i has to wait because of interference from frames in $lf(m_i)$ and $r_{\mathrm{cycle}}^{(k)}$ be the time from the start of the last cycle to the beginning of the transmission of the $(n_i^{(k)}+1)$-th instance of m_i. $f_i(\mathbf{n}^{(k)})$ can be expressed as

$$f_i(\mathbf{n}^{(k)}) = (n_i^{(k)} + s_{\text{cycle}}^{(k)}) l_{\text{comm}} + r_{\text{cycle}}^{(k)} \tag{24.27}$$

The computation of s_{cycle} and r_{cycle} is demonstrated to be NP hard [65, 68]. *For simplicity, from now on the iteration index k is dropped from* $\mathbf{n}$, s_{cycle} *and* r_{cycle}.

Next, an overview is given on the calculation of the exact value of $f_i(\mathbf{n})$. The calculation is based on the solution of two ILPs [65], which is generally time-consuming especially for large problem sizes. Hence, a load-based heuristic and a heuristic leveraging the results on bin-covering problem are summarized afterward.

The calculation of $f_i(\mathbf{n})$ can be viewed as a constrained version of the *bin-covering problem*. A bin-covering problem is to maximize the number of bins using a given list of items with known weights, such that each bin is at least filled to a minimum capacity (the sum of the weights from items packed to the bin is no smaller than the minimum bin capacity). To calculate $f_i(\mathbf{n})$, after the number of instances n_j from each frame $m_j \in lf(m_i)$ is calculated, these instances are used to cover bins (communication cycles) with minimum capacity $LTx_i \cdot l_{\text{MS}}$. Each frame m_j has n_j instances, and its weight is C_j. The problem also contains additional constraints such as (24.34) and (24.35) below.

Exact Solution for $f_i(\mathbf{n})$

Below a brief summary is given on the ILP optimization formulation proposed in [65] to find the exact solutions. The number of busy communication cycles s_{cycle} can be bounded by the total number of frame instances in $lf(m_i)$:

$$s_{\text{cycle}} \le s_{\text{cycle}}^{\text{ub}} = \sum_{j \in lf(m_i)} n_j \tag{24.28}$$

or better, by using the heuristic in (24.41) presented in the following subsections. The number of binary variables in the ILP formulation depends linearly on the maximum number of busy communication cycles; thus using a tighter bound like (24.41) can greatly reduce the complexity.

A set of binary variables defines the transmission of the instances of frames in $lf(m_i)$ in the cycles:

$$\forall j \in lf(m_i), n \le n_j, s \le s_{\text{cycle}}^{\text{ub}}, \quad x_{j,n,s} = \begin{cases} 1 & \text{if the } n\text{-th instance of } m_j \\ & \text{is sent in cycle } s \\ 0 & \text{otherwise} \end{cases} \tag{24.29}$$

Another set of binary variables denotes whether cycle s is busy transmitting frames $\in lf(m_i)$ or not:

$$\forall s \le s_{\text{cycle}}^{\text{ub}}, \quad y_s = \begin{cases} 1 & \text{if the } s\text{-th cycle is busy} \\ 0 & \text{otherwise} \end{cases} \tag{24.30}$$

The total load $N_{i,s}$ from $lf(m_i)$ and idle minislots with slot index $< ID_i$ in cycle s is

$$N_{i,s} = \sum_{j \in lf(m_i)} \sum_{n \le n_j} x_{j,n,s} \cdot C_j + \sum_{j < ID_i} (1 - \sum_{n \le n_j} x_{j,n,s}) l_{\mathrm{MS}} \tag{24.31}$$

A set of constraints encodes the FlexRay protocol requirements. First, in any busy cycle s, $N_{i,s}$ must be no smaller than the length of LTx_i minislots:

$$\forall s \le s^{\mathrm{ub}}_{\mathrm{cycle}}, \quad N_{i,s} \ge LTx_i \cdot l_{\mathrm{MS}} \cdot y_s \tag{24.32}$$

Second, each frame instance is transmitted at most once:

$$\forall j \in lf(m_i), n \le n_j, \quad \sum_{s \le s^{\mathrm{ub}}_{\mathrm{cycle}}} x_{j,n,s} \le 1 \tag{24.33}$$

and each frame ID is transmitted at most once in each cycle

$$\forall j \in lf(m_i), s \le s^{\mathrm{ub}}_{\mathrm{cycle}}, \quad \sum_{n \le n_j} x_{j,n,s} \le 1 \tag{24.34}$$

Third, each instance of m_j is sent no later than the minislot of index LTx_j, formulated using the "big-M" formulation:

$$\forall j \in lf(m_i), n \le n_j, s \le s^{\mathrm{ub}}_{\mathrm{cycle}}, \; N_{j,s} < LTx_j \cdot l_{\mathrm{MS}} \cdot y_s + M(1 - x_{j,n,s}) \tag{24.35}$$

Here M is a large-enough constant, e.g., l_{DYN}.

To find the maximum number of busy communication cycles s_{cycle}, an optimization problem is solved, with objective function in (24.36) and constraints in (24.32), (24.33), (24.34), and (24.35):

$$s_{\mathrm{cycle}} = \max(\sum_{s \le s^{\mathrm{ub}}_{\mathrm{cycle}}} y_s) \tag{24.36}$$

With the solution to s_{cycle}, the value of r_{cycle} can be determined by maximizing the communication load in the $(s_{\mathrm{cycle}} + 1)$-th cycle after filling up the first s_{cycle} cycles:

$$r_{\mathrm{cycle}} = \max(l_{\mathrm{ST}} + N_{i,s_{\mathrm{cycle}}+1}) \tag{24.37}$$

Load-based Heuristic Solution for $f_i(\mathbf{n})$.

The basic idea to approximate $f_i(\mathbf{n})$ is to assume that the s_{cycle} communication cycles are always filled with the minimum amount of load from frames in $lf(m_i)$.

Hence, the concept of the *minimum serviced load* is defined as the minimum transmission time that is needed in addition to the idle minislots to fill a communication cycle bin. It can be calculated as

$$\begin{aligned} p_i &= LTx_i \cdot l_{\mathrm{MS}} - (ID_i - 1)l_{\mathrm{MS}} \\ &= (LTx_i - ID_i + 1)l_{\mathrm{MS}} \end{aligned} \tag{24.38}$$

This may also be derived by manipulating constraint (24.32):

$$\begin{aligned} N_{i,s} &= \sum_{j \in lf(m_i)} \sum_{n \le n_j} x_{j,n,s} C_j + \sum_{j < ID_i, j \notin lf(m_i)} l_{\mathrm{MS}} + \sum_{j \in lf(m_i)} (1 - \sum_{n \le n_j} x_{j,n,s}) l_{\mathrm{MS}} \\ &= \sum_{j \in lf(m_i)} \sum_{n \le n_j} x_{j,n,s}(C_j - l_{\mathrm{MS}}) + (ID_i - 1)l_{\mathrm{MS}} \\ &\ge LTx_i \cdot l_{\mathrm{MS}} \cdot y_s \end{aligned} \tag{24.39}$$

With $C_i' = C_i - l_{\mathrm{MS}}$, constraint (24.32) is equivalent to

$$\sum_{j \in lf(m_i)} \sum_{n \le n_j} x_{j,n,s} \cdot C_j' \ge p_i \cdot y_s \tag{24.40}$$

An upper bound on s_{cycle} can be derived by considering a bin-packing problem formulation, where (24.32) (or equivalently (24.40)) and (24.33) are respected but constraints (24.34) and (24.35) are ignored [65]. With the notation K_i as

$$K_i = \frac{\sum_{j \in lf(m_i)} (n_j \cdot C_j')}{p_i} \tag{24.41}$$

it is

$$s_{\mathrm{cycle}} \le \lfloor K_i \rfloor \tag{24.42}$$

The upper bound on $f_i(\mathbf{n})$ is

$$f_i(\mathbf{n}) \le (n_i + \lfloor K_i \rfloor) l_{\mathrm{comm}} + l_{\mathrm{ST}} + (ID_i - 1)l_{\mathrm{MS}} + (K_i - \lfloor K_i \rfloor) p_i \tag{24.43}$$

However, the upper bound in Eq. (24.43) is in general loose, since constraint (24.34) that requires each frame identifier is used at most once in each cycle is ignored. With this observation, constraint (24.34) is brought back into consideration: at most one instance of any frame $m_j \in lf(m_i)$ can be packed in each communication cycle.

Algorithm 1 reflects this idea and gives a tighter upper bound $f_i^H(\mathbf{n})$ on $f_i(\mathbf{n})$ than Eq. (24.43). It uses a *load variable* L to denote the available requested transmission time within a communication cycle. The loop from Lines 2–15

implements the iterative procedure to calculate L and s. It tries to fill each bin (communication cycle) starting from cycle $s = 0$. In each cycle s, one instance from each $m_j \in lf(m_i)$ that has $n_j > 0$ is added to get the load variable L (Lines 3–8). L is the maximum amount of time that is available for transmission in the communication cycle s. By adding only a term $C'_j = C_j - l_{\text{MS}}$ for each $m_j \in lf(m_i)$, at most one instance from each frame m_j can be transmitted in this cycle. If $L \geq p_i$, the cycle s is filled, the bin capacity p_i is subtracted from the load variable L, and the iteration continues to the next cycle (Lines 9–11). The reason is that in the worst-case scenario, only p_i of these loads is transmitted in the communication cycle and the maximum amount of *remaining loads* is delayed to the next cycles. If $L < p_i$, there is not enough load to fill the bin, and m_i is transmitted in the current communication cycle, and the iteration stops (Lines 12–13). Then, s is assigned to s^H_{cycle}, and r^H_{cycle} is calculated as $l_{\text{ST}} + (ID_i - 1)l_{\text{MS}} + L$, assuming L as the additional load in the communication cycle in which the $(n_i + 1)$-th instance of m_i is transmitted (Line 16). Finally, $f_i(\mathbf{n})$ is calculated as the length of $(n_i + s_{\text{cycle}})$ communication cycles plus r_{cycle} (Line 17).

Algorithm 1 Algorithm to compute the upper bound $f_i^H(\mathbf{n})$ on $f_i(\mathbf{n})$

1: $p_i = (LTx_i - ID_i + 1)l_{\text{MS}}$, $L = 0$, $s = 0$
2: **while** true **do**
3: **for** each frame $j \in lf(m_i)$ **do**
4: **if** $n_j > 0$ **then**
5: $L = L + C_j$
6: $n_j = n_j - 1$
7: **end if**
8: **end for**
9: **if** $L \geq p_i$ **then**
10: $s = s + 1$
11: $L = L - p_i$
12: **else**
13: break
14: **end if**
15: **end while**
16: $s^H_{\text{cycle}} = s$, $r^H_{\text{cycle}} = l_{\text{ST}} + (ID_i - 1)l_{\text{MS}} + L$
17: $f_i^H(\mathbf{n}) = (n_i + s^H_{\text{cycle}})l_{\text{comm}} + r^H_{\text{cycle}}$

The value of $f_i^H(\mathbf{n})$ returned by Algorithm 1 is proven to be no smaller than the exact value $f_i(\mathbf{n})$ [83]. Essentially s^H_{cycle} is solved with a more relaxed problem than the exact solution s_{cycle}. It ignores the constraints (24.32) and (24.35). The complete proof is documented in [83]. Hence, the algorithm is a pessimistic but safe procedure in that the resulted worst-case response time R_i is always an upper bound for the actual response time.

Bin-Covering-Based Heuristic Solution for f_i(n).
Alternatively, Tanasa et al. [68] apply recent theoretical advances [38] to approximate the upper bounds on the optimal solution for the bin-covering problem. It takes an input parameter ϵ to define the precision of the results. The details of how the bin-covering heuristic is solved can be found in [68]. It is reported that [68] the approach provides improvement over the load-based heuristic when $\epsilon \leq 1/16$ and is comparable when $\epsilon = 1/8$. Hence, a value between $1/16$ and $1/8$ for ϵ can provide the right balance between efficiency and quality.

24.3.3.2 Extension to Slot Multiplexing

The generalization of the two techniques to slot multiplexing is now discussed. For this purpose, p_i is extended with an additional parameter cc, i.e., $p_i(cc)$, to denote the minimum serviced load for communication cycle cc. Similarly, $lf(m_i, cc)$ denotes the set of messages in $lf(m_i)$ which can be transmitted in cc, i.e., $lf(m_i, cc) = \{m_j | m_j \in lf(m_i), cc \equiv b_j \pmod{r_j}\}$.

Consider an example as presented in [68], which contains five frames with their cycle repetition and base cycle in Table 24.2. Figure 24.5 illustrates the communication cycles that each frame is allowed to transmit.

Load-Based Heuristic
Slot multiplexing provides flexibility and efficiency for scheduling but also introduces new challenges to the timing analysis. Each communication cycle should be regarded differently, due to the facts that the message m_i under analysis and those with lower ID (in $lf(m_i)$) cannot be transmitted in every cycle.

Table 24.2 Frame parameters of an example

	m_1	m_2	m_3	m_4	m_5
Cycle repetition	1	2	4	8	2
Base cycle	0	0	0	0	0

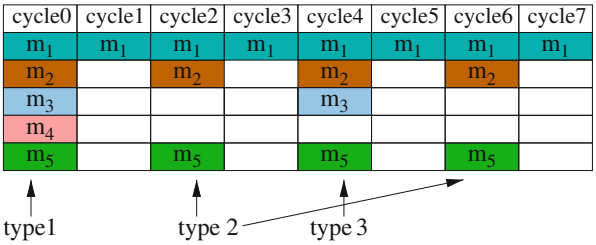

Fig. 24.5 The allowed communication cycles for the frames in Table 24.2

- First, the first complete communication cycle cc_0 after the start of the busy window can be any one in the hyperperiod. Hence, in the analysis, it is sufficient to enumerate the possible values $\{0...CC^{\max}-1\}$ for cc_0.
- Second, it is insufficient to use a single load variable since the load accumulated in one cycle may not be delayed to the next cycles. This can be illustrated with a simple example that m_i can be transmitted in every cycle, and each message m_j in $lf(m_i)$ has $r_j = 2$ and $b_j = 0$. Even if the accumulated load from $lf(m_i)$ in cycle 0 can be larger than $2p_i$, it is not deferrable to cycle 1, and m_i can be sent in cycle 1. The solution to this issue is explained later.
- Third, the minimum serviced load p_i in each cycle is different as m_i may not be transmitted in certain cycles. A conservative estimate is that for those communication cycle $cc \mod r_i \neq b_i$, $p_i(cc)$ is set to be 0. This can be improved to $p_i(cc) = \min_{j \in lf(m_i,cc)} p_j(cc)$, since any of the messages in $lf(m_i)$ could still start transmission if less load was serviced [63].

To analyze the effects of slot multiplexing on the interference that messages in $lf(m_i)$ may introduce (the second challenge above), a set of load variables $L_{b,r}$ is added to denote the cycle-dependent load, where b is the base cycle and r is the repetition factor. At a given cycle cc during the analysis, the transmission time C_j of m_j is added to the load variable L_{b_j,r_j} only if $cc \equiv b_j \pmod{r_j}$, to reflect the fact that m_j is only allowed to transmit in such communication cycles. Hence, the total available load $L(cc)$ at cc is

$$L(cc) = \sum_{b,r:\ cc \equiv b \pmod{r}} L_{b,r} \tag{24.44}$$

If $L(cc) < p_i(cc)$, then there is not enough load to further delay m_i, and m_i will be sent in the earliest cycle $\geq cc$ that it is permitted to transmit. Otherwise, m_i will be delayed, and the minimum serviced load should be subtracted.

Since $L(cc)$ is in general composed of loads from several suitable $L_{b,r}$ variables, there is an additional question about which $L_{b,r}$ should $p_i(cc)$ be subtracted from. Neukirchner et al. [63] observe that the repetition factors are only allowed to be a power of 2, which helps to simplify the problem. Because of this restriction in the FlexRay specification, any load variable coincides with several load variables of a lower repetition factor. For example, $L_{3,4}$ always coincides with $L_{1,2}$ and $L_{0,1}$. To maximize the $L(cc)$ for every cc, it is sufficient to maximize the $L_{b,r}$ variable with the smallest cycle repetition r. This can be achieved by subtracting the minimal serviced load $p_i(cc)$ first from the available $L_{b,r}$ with the highest cycle repetition.

Bin-Covering-Based Heuristic

The heuristic based on bin-covering approximation algorithm can also be extended to slot multiplexing. However, the problem is no longer a traditional bin-covering problem. Rather, the problem becomes what was called bin-covering problem with conflicts [68], as not all messages (items) can be transmitted in every

communication cycle (bin). The number of types of bins P is determined by the distinct communication cycles in a hyperperiod, where two communication cycles are considered distinct if the set of messages they can carry are different. With this understanding, the problem then can be reformulated as bin-covering problem with specific requirements on the number of bins to be packed for each of the P types. For example, in Fig. 24.5, there are three types of bins for the purpose of analyzing m_5's response time: type 1 containing cycle 0 where all higher-priority messages m_1–m_4 can transmit, type 2 for cycles 2 and 6, and type 3 for cycle 4.

24.4 Packet-Switched Networks: Ethernet

24.4.1 Introduction

In addition to traditional buses such as CAN or FlexRay, packet-switched Ethernet will be used in next-generation automotive communication architectures. Ethernet's superior bandwidth and flexibility make it ideal to address the high communication demands of, for example, Advanced Driver Assistance Systems (ADASs), infotainment systems, and ECU flashing. As a switched network, Ethernet provides a scalable, high-speed, and cost-effective communication platform, which allows arbitrary topologies.

Ethernet evolved from a shared bus communication medium with Carrier Sense Multiple Access/Collision Detection (CSMA/CD)-based link access scheme to a switched network. Frame collisions in CSMA/CD were resolved by a binary exponential backoff algorithm which picked a random delay until a retransmission could be started after a collision. This deemed CSMA/CD unsuitable for real-time systems with tight latency or jitter requirements. Switched Ethernet made CSMA/CD obsolete. In switched Ethernet, contention is moved into the switches, where a scheduler has full control over each output port. This enables the implementation of elaborate link schedulers, which allow the derivation of real-time guarantees. Today, Ethernet installations (including the automotive domain) are almost always switched. Hence, in the following, we will refer to switched Ethernet as standard Ethernet.

In the automotive context, Ethernet is anticipated to serve as an in-vehicle communication backbone, where it must be able to transport traffic streams of mixed criticality. This requires Quality of Service (QoS) mechanisms, in order to provide deterministic timing guarantees for critical traffic. Standard Ethernet (IEEE 802.1Q) introduced eight traffic classes. These classes can be used to prioritize traffic, which is typically implemented by a Static-Priority Non-Preemptive (SPNP) scheduler at each output port in each switch and end point. This limited number of classes requires that multiple traffic streams share a class, making streams of equal priority indistinguishable to the scheduler. Traffic within a shared class is usually scheduled in First-In First-Out (FIFO) order.

Compared to CAN or FlexRay, Ethernet exhibits complex timing behavior, as each switch output port is a point of arbitration, which adds delay to the overall

end-to-end latency. While mature formal performance analysis techniques have been established for CAN and FlexRay, such techniques are even more required for Ethernet before it can be used in timing- and safety-critical systems. This will become even more important in the context of highly automated and autonomous driving. In this section, we use Compositional Performance Analysis (CPA) (see ▸ Chap. 23, "CPA: Compositional Performance Analysis" and [29]) to derive worst-case performance bounds for Ethernet.

24.4.2 Modeling Ethernet Networks for Performance Analysis

Before timing guarantees can be derived for an Ethernet network, the components of this network must be mapped to the CPA system model (cf. Sect. 2 in ▸ Chap. 23, "CPA: Compositional Performance Analysis"). This mapping process is explained in detail in [19]. Here, a brief summary covering the essential steps, using Figs. 24.6 and 24.7 as illustration, is presented.

Figure 24.6 shows an Ethernet model comprising two switches and four ECUs. A sequence of related Ethernet frames between a source and one (or more) destination(s) is called an Ethernet traffic stream. There are two traffic streams in the network: a unicast stream from ECU0 to ECU3 and a multicast stream from ECU2 to ECU1 and ECU3.

In order to map the Ethernet model to the CPA system model, resources, tasks, and event models must be identified. Resources model points of contention. In Ethernet, contention between individual frames happens at the switches. Inside a switch there are several delay sources. At the input port, there is *input queuing delay*, the switch fabric adds *forwarding delay*, and at the output port, there is *output queuing delay*. Contemporary switches are fast enough that input queuing delay and forwarding delay only have a negligible impact on the overall timing guarantees. Hence, these delays can be ignored or approximated by constant terms. The output queuing delay considers the time it takes to transmit a given frame, including the interference from other frames. Consequently, switch output ports are modeled by CPA resources. The scheduling policy of these resources is determined by the switch port's scheduling mechanism. Additionally, here is *transmission delay* on the link between switches. This delay corresponds to the propagation delay of electric signals on the link's wire and can also be modeled by a constant term.

The transmission of a frame via an output port of a switch is modeled in CPA as the execution of a task on the port's resource. An Ethernet traffic stream is modeled as a chain of dependent frames (tasks) according to its path through the network

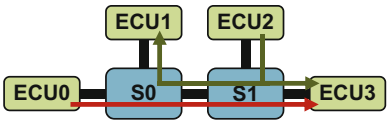

Fig. 24.6 Ethernet model

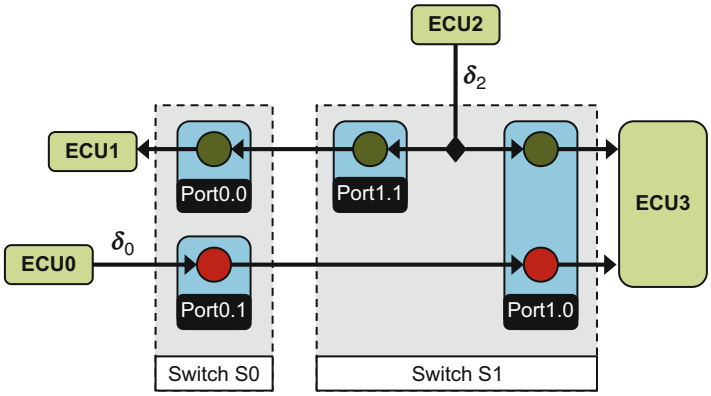

Fig. 24.7 CPA system model for the Ethernet model from Fig. 24.6

(see Fig. 24.7). This chain may fork to model multicast or broadcast trees. On each resource, a task consumes service according to its execution time bounds, which are derived from the best-case and worst-case transmission times of its corresponding Ethernet frame. The transmission time of a frame is defined to be the time it takes the frame to be transmitted without any interference from other frames. For a frame of traffic stream i with maximum/minimum payload $p_i^{-/+}$, the best-case and worst-case transmission times C_i^- and C_i^+ can be computed to

$$C_i^{+/-} = \frac{42\,\text{bytes} + \max\left\{42\,\text{bytes}, p_i^{+/-}\right\}}{r_{TX}} \tag{24.45}$$

where r_{TX} is the transmission rate of the port that transmits the frame. The constant terms correspond to the protocol overhead. The first 42 bytes account for preamble (7 bytes), start of frame delimiter (1 byte), destination and source Media Access Control (MAC) address (both 6 bytes), IEEE 802.1Q tag (4 bytes), EtherType (2 bytes), frame check sequence (4 bytes), and inter-frame gap (12 bytes). The second 42 bytes account for the fact that there is a minimum Ethernet frame size of 84 bytes and that the payload must be padded if necessary.

Frame arrivals (and emissions) are modeled by event models. These models come from either external sources or from dependent frames.

Figure 24.7 shows the corresponding CPA model of the Ethernet model from Fig. 24.6. As can be seen, the output ports of both switches are modeled as resources (light blue boxes with rounded corners). Both traffic streams are modeled by a chain of tasks (red and green circles) reflecting their paths through the network. Notice that the green path originating at ECU2 splits into a multicast tree. ECU0 and ECU2 inject frames into the network according to the event models δ_0 and δ_2 (respectively). This model can then be analyzed with CPA's iterative approach.

24.4.3 Analysis of Standard Ethernet (IEEE 802.1Q)

In order to derive upper bounds on the worst-case performance of Ethernet networks, an analysis which captures all delay effects on the CPA resources that model the switch output ports must be developed. This analysis will then be used in the local analysis step of the CPA loop to derive worst-case frame transmission latencies on each output port.

Definition 1. A frame's transmission latency is the time interval, which starts when the frame has been received at an input port and ends when it has been transmitted entirely from an output port. The transmission latency includes all timing effects from interfering traffic streams.

In the context of the model transformation from Sect. 24.4.2, the transmission latency of a frame corresponds to the response time of a task.

When deriving formal performance guarantees, the worst-case transmission latency of the frames of a given traffic stream i (among all for stream i's possible frames transmission latencies) is of particular interest. For non-preemptive scheduling (such as standard Ethernet), it has been shown that, in order to find this worst-case transmission latency, the transmission latencies of all frames of stream i in its longest scheduling horizon must be evaluated (cf. [12]). The scheduling horizon of a traffic stream i is the time a switch port is busy processing frames of stream i, including interference from frames of other traffic streams (cf. Sect. 2.2.1 in ▸ Chap. 23, "CPA: Compositional Performance Analysis" and [17]). Particularly, the worst-case transmission latency of the q-th frame of traffic stream i can be derived from its worst-case multiple activation queuing delay $Q_i(q, a_i^q)$ (cf. Eq. (9) in ▸ Chap. 23, "CPA: Compositional Performance Analysis").

Definition 2. Assuming that the q-th frame of a traffic stream i arrives at time a_i^q at a switch output port, its worst-case multiple activation queuing delay $Q_i(q, a_i^q)$ is the time interval, which starts with the arrival of the first frame of stream i that initiates the scheduling horizon and ends when the q-th frame can be transmitted (i.e., it does not include the transmission of the q-th frame).

Note that, in contrast to the multiple activation queuing delay $Q_i(q)$ introduced in Sect. 2.2.1 in ▸ Chap. 23, "CPA: Compositional Performance Analysis", the queuing delay in the Ethernet context additionally depends on the arrival time a_i^q of the q-th frame. This is due to the FIFO scheduling of frames with equal priority and will be explained later in this section. The arrival time a_i^q of the q-th frame of stream i is measured relative to the beginning of the scheduling horizon.

As stated in Sect. 24.4.2, it is assumed that the queuing delay of a given frame at a switch output port accounts for all delays induced by interfering traffic streams. The amount of interference from other traffic streams depends on the output port's scheduling policy. In standard Ethernet, traffic streams are categorized into (up to)

eight traffic classes, which correspond to priority levels. Inside each output port there is a set of FIFO queues, one for each traffic class. These FIFO queues are served by an SPNP scheduler. Consequently, to calculate the worst-case queuing delay $Q_i(q, a_i^q)$ in standard Ethernet, all blocking effects, which can occur in this combination of FIFO and SPNP scheduling, must be considered.

Lower-priority blocking: In non-preemptive scheduling, a frame which started transmitting is guaranteed to finish without interruption. Hence, a frame of traffic stream i can experience blocking from at most one lower-priority frame, if this lower-priority frame started transmitting just before the arrival of the first frame of traffic stream i [21]:

$$I_i^{LPB} = \max_{j \in lp(i)} \left\{ C_j^+ \right\} \tag{24.46}$$

where $lp(i)$ is a function yielding the set of all traffic streams whose priority is lower than that of stream i.

Higher-priority blocking: In any time interval of length Δt, a frame of traffic stream i can experience blocking from all frames of higher-priority streams, which arrive during Δt, i.e., before the frame of stream i can be transmitted [21]:

$$I_i^{HPB}(\Delta t) = \sum_{j \in hp(i)} \eta_j^+(\Delta t + \epsilon) C_j^+ \tag{24.47}$$

where $hp(i)$ is a function yielding the set all traffic streams whose priority is higher than that of stream i. Recall from Sect. 2.1.2 in ▸ Chap. 23, "CPA: Compositional Performance Analysis" that $\eta^+(\Delta t)$ yields an upper bound on the number of events, i.e., frame arrivals, in any half open time interval of length Δt. As the multiple activation queuing delay $Q_i(q, a_i^q)$ covers the time *until* the q-th frame can be transmitted, higher-priority frames arriving exactly at the end for Δt can also interfere with the q-th frame. We model this by adding an infinitesimal small time ϵ to Δt to cover the *closed* time interval $[t, t + \Delta t]$. In practice, ϵ corresponds to a bit time.

Same-priority blocking: As frames of identical priority are processed in FIFO order, frames of traffic stream i can experience blocking from frames of other traffic streams with the same priority as stream i. Hence, if the q-th frame of traffic stream i arrives at time a_i^q, it must wait for all frames from other streams with identical priority, which arrived before or at a_i^q, as well as wait for its own $q - 1$ predecessor frames to finish [21]:

$$I_i^{SPB}(q, a_i^q) = (q-1) C_i^+ + \sum_{j \in sp(i)} \eta_j^+(a_i^q + \epsilon) C_j^+ \tag{24.48}$$

Here, $sp(i)$ is a function yielding the set of all traffic streams whose priority is equal to that of stream i (excluding stream i). In the worst case, any same-priority frames arriving concurrently at exactly a_i^q are assumed to interfere with the q-frame of stream i. Again, an infinitesimal small time ϵ is added to Δt to cover this case.

In [21] it is shown that FIFO scheduling requires a candidate search in order to determine the worst-case blocking. The reason for this candidate search is that if frame q arrives early (within its jitter bounds), it might experience additional blocking from some of its own $q-1$ queued predecessors. However, if it arrives late (within its jitter bounds), it might experience additional blocking from previously queued frames of interfering same-priority streams. The set of arrival candidates A_i^q can be reduced to points in time where the candidates a_i^q coincide with the earliest arrivals of interfering frames from same-priority traffic streams [21]. Consequently, all candidates for the arrival of the q-th frame of stream i can be found by investigating the arrivals of interfering frames between the earliest arrival $\delta_i^-(q)$ of the q-th frame and its q-activation scheduling horizon $S_i(q)$, which is the time a switch port is busy processing q frames of stream i, including interfering frames from other traffic streams (cf. Eq. (4) in ▸ Chap. 23, "CPA: Compositional Performance Analysis"):

$$A_i^q = \bigcup_{j \in sp(i)} \left\{ \delta_j^-(n) | \delta_i^-(q) \leq \delta_j^-(n) < S_i(q) \right\}_{n \geq 1} \tag{24.49}$$

where, in the context of standard Ethernet, $S_i(q)$ can be computed as follows:

$$S_i(q) = \max_{j \in lp(i)} \left\{ C_j^+ \right\} + qC_i^+ + \sum_{j \in sp(i) \cup hp(i)} \eta_j^+ \left(S_i(q) \right) C_j^+ \tag{24.50}$$

Note that the computation of the q-activation scheduling horizon does not require a candidate search for the same-priority interference, as it is only concerned about the time when the port is busy. As $S_i(q)$ occurs on both sides, Eq. (24.50) cannot be solved directly. However, it represents an integer fixed-point problem, which can be solved by iteration, as all terms are monotonically increasing (cf. [29]). A valid starting point is, e.g., $S_i(q) = \max_{j \in lp(i)} \{C_j^+\} + qC_i^+$.

In order to compute the worst-case queuing delay $Q_i(q, a_i^q)$ of the q-th frame arrival of traffic stream i, which arrived at time a_i^q, all presented blocking effects must be considered:

$$Q_i(q, a_i^q) = I_i^{LPB} + I_i^{SPB}(q, a_i^q) + I_i^{HPB}\left(Q_i(q, a_i^q) \right) \tag{24.51}$$

Again, $Q_i(q, a_i^q)$ occurs on both sides, and Eq. (24.51) cannot be solved directly. Like the integer fixed-point problem in Eq. (24.50), it can be solved by iteration with, e.g., $Q_i(q, a_i^q) = (q-1)C_i^+$ as a starting point.

Now, the largest transmission latency $R_i(q)$ for the q-th frame arrival of traffic stream i can be computed by adding the transmission time C_i^+ of this q-th frame to its worst-case queuing delay and accounting for the fact that the frame arrived at time a_i^q (see, e.g., [3]). This is illustrated in Fig. 24.8:

$$R_i(q) = \max_{a_i^q \in A_i^q} \left\{ Q_i(q, a_i^q) + C_i^+ - a_i^q \right\} \tag{24.52}$$

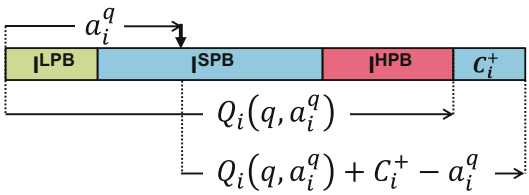

Fig. 24.8 Example queuing delay and transmission latency computation (cf. [75])

By taking the maximum overall $R_i(q)$, the worst-case frame transmission latency for a frame of stream i can be computed:

$$R_i^+ = \max_{1 \le q \le q_i^+} \{R_i(q)\} \tag{24.53}$$

As mentioned before, in order to derive the worst-case frame transmission latency of stream i, all frame arrivals of stream i in its longest scheduling horizon must be evaluated. Let q_i^+ be the maximum number of these frame arrivals. It can be derived by computing the maximum number of frames, which arrive during the scheduling horizon of their respective predecessors (cf. [18]):

$$q_i^+ = \max_{q \ge 1} \{q | \delta_i^-(q) \le S_i(q-1)\} \tag{24.54}$$

Now, as established in Sects. 2.2.1 and 2.2.2 in ▸ Chap. 23, "CPA: Compositional Performance Analysis", the worst-case bounds on the maximum path latency and the maximum frame backlog can be derived from the maximum frame transmission latencies and maximum q-activation processing times.

24.4.3.1 End-to-End Latency Bounds

From the individual worst-case transmission latencies of the frames along the path of a traffic stream through the network, the worst-case end-to-end latency of the stream can be derived. Let $Path(i)$ be the path of stream i through the network. Now, the time it takes to transmit q frames of stream i, i.e., its worst-case q-activation end-to-end latency, can be bounded by (cf. [20]):

$$L_i^+(q) = \delta_i^-(q) + \sum_{j \in Path(i)} R_j^+ \tag{24.55}$$

Here, the frames of stream i are injected into the network at their maximum rate (i.e., with minimum inter-arrival times $\delta_i^-(q)$) to induce maximum load on the system's resources. Along any given path, frames of a traffic stream are processed in order, i.e., they cannot overtake each other. Equation (24.55) assumes that the last of the q frames experiences the worst-case transmission latency on all its ports. Due to in-order processing, all $q-1$ previously sent frames must have arrived by then.

Obviously, for $q = 1$, Eq. (24.55) yields the worst-case end-to-end latency of a single frame (recall that $\delta_i^-(1) = 0$). Larger q are convenient in cases where, for example, a large IP packet is distributed over multiple Ethernet frames.

24.4.3.2 Buffer Size Bounds

Apart from timing guarantees, buffer size requirements are also important, as actual systems (e.g. ,switches) only have limited memory resources (buffer space). Insufficient buffer space can lead to frame drop, which is highly undesirable for (time) critical traffic.

The maximum activation backlog of a traffic stream i is an upper bound on the number of frames from i that can be queued at a resource at any given time. It can be derived by computing, for each q-th frame, the maximum number of frames, which arrived until the q-th frame has been transmitted, and subtracting from this number the $q - 1$ frames that must have been transmitted prior to the q-th one (cf. [21]):

$$b_i^+ = \max_{1 \le q \le q_i^+} \left\{ \eta_i^+ \left(B_i^+(q) \right) - q + 1 \right\} \tag{24.56}$$

where $B_i^+(q)$ is the multiple activation processing time. Given q consecutive frames of a traffic stream i, the multiple activation processing time is the longest time interval between the arrival of the first frame and end of the transmission of the q-th frame (cf. Eq. (8) in ▸ Chap. 23, "CPA: Compositional Performance Analysis"):

$$B_i^+(q) = Q_i(q) + C_i^+ \tag{24.57}$$

In the Ethernet context, it can be bounded by the multiple activation queuing delay under the assumption that all event arrive as soon as possible (i.e., we do not need to consider different event arrivals as in Eq. (24.51)) by adding the frames worst-case transmission time C_i^+ (cf. Eq. (9) in ▸ Chap. 23, "CPA: Compositional Performance Analysis"):

$$Q_i(q) = I_i^{LPB} + I_i^{SPB}(q, Q_i(q)) + I_i^{HPB}(Q_i(q)) \tag{24.58}$$

From the maximum activation backlogs, the maximum buffer size requirements can be derived. Typically, the memory in Ethernet switches can only be allocated block wise, e.g., in blocks of 128 or 256 bytes. This must be taken into account when deriving the maximum buffer size requirements. Assuming that a switch only allows the allocation of memory blocks of size m and that only the destination and source MAC addresses, the IEEE 802.1Q tag, the EtherType, the maximum payload p^+, and the frame check sequence of an Ethernet frame must be stored in switch memory, the buffer size requirement (in bytes) for a traffic stream i can be bounded by (cf. Sect. 24.4.2):

$$\hat{b}_i^+ = b_i^+ \left\lceil \frac{22\,\text{bytes} + \max\{42\,\text{bytes}, p^+\}}{m} \right\rceil m \tag{24.59}$$

The buffer size requirement per port can be computed by summing the buffer size requirements of all streams passing this port, and the buffer size requirement of a switch can be computed by summing up the requirements of each of its ports.

24.4.4 Analysis Extensions

24.4.4.1 Other Ethernet Schedulers

As Ethernet strives to cover a wide range of application domains, it supports many different schedulers and shapers to forward frames, each of which has a different impact on the queuing delay at a switch's output port. For the most prominent ones, CPA-based analyses are available.

Ethernet AVB [31] introduced standardized traffic shaping in the form of credit-based shaping on top of standard Ethernet. The motivation is to shape higher-priority traffic streams to bound their interference on lower-priority ones, e.g., to prevent starvation. However, as any form of traffic shaping introduces additional delays, a careful timing analysis is required to evaluate Ethernet AVB's applicability for real-time applications. Formal analyses for Ethernet AVB in the context of the CPA framework are presented in [21] and [3].

Ethernet TSN defines a set of Ethernet standards, which were designed with real-time requirements in mind. Some of these standards specify new link arbitration mechanisms. Namely, IEEE 802.1Qbv [35] introduces time-triggered frame forwarding to Ethernet, i.e., frames of time-triggered traffic classes are scheduled at predefined points in time such that they do not experience interference from other traffic classes. IEEE 802.1Qbv relies on so-called guard bands to block non-time-triggered traffic early enough to prevent interference with time-triggered traffic. In IEEE 802.1Qch [32], cyclic frame forwarding is defined, i.e., frame forwarding is based on alternating time intervals and frames received in one interval will be sent in the next interval etc. A new credit-based shaper, which aims to improve the forwarding of bursts, is discussed in [25]. Formal analyses for these shapers are presented in [75] and [74].

Although not explicitly standardized by the IEEE, weighted round robin scheduling can be implemented as an IEEE 802.1Q enhanced transmission selection algorithm. A CPA-compatible formal analysis for weighted round robin scheduling in the Ethernet context has been presented in [71].

In order to improve the timing of critical traffic, frame preemption has been introduced to Ethernet via the IEEE 802.3br [33] and IEEE 802.1Qbu [34] standards. A CPA-compatible formal analysis for frame preemption has been presented in [73].

24.4.4.2 Analysis Improvements

This section covered the fundamental approach to derive timing guarantees for Ethernet networks in CPA. The presented baseline analysis has been improved and extended in many directions.

Different analysis optimizations to exploit various kinds of correlations between Ethernet traffic streams have been proposed in [3] and [70]. Axer et al. [3] exploits

the fact that both Ethernet links and Ethernet AVB's traffic shapers limit the amount of workload, which can pass them in a given time interval. This property can be used to limit the interference during the computation of the worst-case frame transmission latencies. In [70], the authors show how FIFO scheduling can be exploited to reduce the interference a frame can experience from its same-priority predecessors.

24.4.4.3 Higher-Layer Protocols

Ethernet only defines frame forwarding on layer 2 of the ISO/OSI model. Higher-layer protocols often have additional timing implications. In [2] and [72] analyses to determine a bound on the worst-case timing impact of Automatic Repeat Requests (ARQs) and software-defined networking (SDN) [42] are presented.

Due to the compositional nature of CPA, the Ethernet analysis can be easily combined with other analyses from the CPA framework to derive system-wide performance guarantees. In [76], this has been done to compute end-to-end latency bounds for CAN-over-Ethernet traffic, where Ethernet ports are modeled as described in this section, but CAN buses and gateway processors are modeled according to their respective scheduling policies.

24.5 Conclusion

This chapter gives the overview on three representative communication protocols for real-time embedded systems, focusing on their timing related design principles and analysis. We note that real-time embedded systems are increasingly equipped with more sophisticated features (such as autonomous driving) that require high adaptivity and large volume data exchange. As a consequence, we envision that their future supporting communication networks will provide better extensibility and higher bandwidth while still keeping their behavior predictable.

Acknowledgments The contribution *Packet-Switched Networks: Ethernet* has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 644080.

References

1. Andersson B, Tovar E (2009) The utilization bound of non-preemptive rate-monotonic scheduling in controller area networks is 25%. In: 2009 IEEE international symposium on industrial embedded systems, pp 11–18
2. Axer P, Thiele D, Ernst R (2014) Formal timing analysis of automatic repeat request for switched real-time networks. In: Proceedings of the SIES, Pisa
3. Axer P, Thiele D, Ernst R, Diemer J (2014) Exploiting shaper context to improve performance bounds of Ethernet AVB Networks. In: Proceedings of the DAC, San Francisco
4. Baruah S, Chen D, Gorinsky S, Mok A (1999) Generalized multiframe tasks. Real-Time Syst 17(1):5–22

5. Broster I, Burns A, Rodriguez-Navas G (2002) Probabilistic analysis of can with faults. In: 23rd IEEE real-time systems symposium, pp 269–278
6. von der Bruggen G, Chen JJ, Huang WH (2015) Schedulability and optimization analysis for non-preemptive static priority scheduling based on task utilization and blocking factors. In: 2015 27th Euromicro conference on real-time systems (ECRTS). IEEE, pp 90–101
7. Casparsson L, Rajnak A, Tindell K, Malmberg P (1998) Volcano revolution in on-board communications. Technical report, Volvo
8. Chen Y, Kurachi R, Takada H, Zeng G (2011) Schedulability comparison for can message with offset: priority queue versus FIFO queue. In: 19th international conference on real-time and network systems, pp 181–192
9. Darbandi A, Kim MK (2014) Schedule optimization of static messages with precedence relations in FlexRay. In: Sixth international conference on ubiquitous and future networks, pp 495–500
10. Darbandi A, Kwon S, Kim MK (2014)Scheduling of time triggered messages in static segment of FlexRay. Int J Softw Eng Appl 8(6):195–208
11. Davis R, Navet N (2012) Controller area network (CAN) schedulability analysis for messages with arbitrary deadlines in FIFO and work-conserving queues. In: 9th IEEE international workshop on factory communication systems, pp 33–42
12. Davis RI, Burns A, Bril RJ, Lukkien JJ (2007) Controller area network (CAN) schedulability analysis: refuted, revisited and revised. Real-Time Syst 35(3):239–272
13. Davis RI, Kollmann S, Pollex V, Slomka F (2013) Schedulability analysis for controller area network (CAN) with FIFO queues priority queues and gateways. Real-Time Syst 49(1): 73–116
14. Di Natale M, Zeng H (2010) System identification and extraction of timing properties from controller area network (CAN) message traces. In: IEEE conference on emerging technologies and factory automation, pp 1–8
15. Di Natale M, Zeng H (2013) Practical issues with the timing analysis of the controller area network. In: 18th IEEE conference on emerging technologies factory automation, pp 1–8
16. Di Natale M, Zeng H, Giusto P, Ghosal A (2012) Understanding and using the controller area network communication protocol: theory and practice. Springer Science & Business Media, New York
17. Diemer J (To appear) Predictable network-on-chip for general-purpose processors – formal worst-case guarantees for on-chip interconnects. Ph.D. thesis, Technische Universität Braunschweig, Braunschweig. N/A
18. Diemer J, Axer P, Ernst R (2012) Compositional performance analysis in python with pyCPA. In: International workshop on analysis tools and methodologies for embedded and real-time systems
19. Diemer J, Rox J, Ernst R (2012) Modeling of Ethernet AVB networks for worst-case timing analysis. In: MATHMOD – Vienna international conference on mathematical modelling, Vienna
20. Diemer J, Rox J, Negrean M, Stein S, Ernst R (2011) Real-time communication analysis for networks with two-stage arbitration. In: Proceedings of the ninth ACM international conference on embedded software (EMSOFT 2011). ACM, Taipei, pp 243–252
21. Diemer J, Thiele D, Ernst R (2012) Formal worst-case timing analysis of ethernet topologies with strict-priority and AVB switching. In: IEEE international symposium on industrial embedded systems. Invited Paper
22. Ding S (2010) Scheduling approach for static segment using hybrid genetic algorithm in FlexRay systems. In: 10th IEEE international conference on computer and information technology, pp 2355–2360
23. Ding S, Murakami N, Tomiyama H, Takada H (2005) A ga-based scheduling method for FlexRay systems. In: 5th ACM international conference on embedded software, pp 110–113
24. Ghosal A, Zeng H, Di Natale M, Ben-Haim Y (2010) Computing robustness of FlexRay schedules to uncertainties in design parameters. In: Proceedings of the conference on design, automation and test in Europe, pp 550–555

25. Götz FJ (2013) Alternative shaper for scheduled traffic in time sensitive networks. In: IEEE 802.1 TSN TG meeting, Vancouver
26. Grenier M, Havet L, Navet N (2008) Configuring the communication on FlexRay-the case of the static segment. In: 4th European congress on embedded real time software
27. Han G, Di Natale M, Zeng H, Liu X, Dou W (2013) Optimizing the implementation of real-time simulink models onto distributed automotive architectures. J Syst Archit 59(10): 1115–1127
28. Han G, Zeng H, Li Y, Dou W (2014) SAFE: security-aware FlexRay scheduling engine. In: Design, automation and test in Europe conference and exhibition
29. Henia R, Hamann A, Jersak M, Racu R, Richter K, Ernst R (2005) System level performance analysis – the SymTA/S approach. In: IEE proceedings computers and digital techniques
30. Hu M, Luo J, Wang Y, Lukasiewycz M, Zeng Z (2014) Holistic scheduling of real-time applications in time-triggered in-vehicle networks. IEEE Trans Ind Inf 10(3): 1817–1828
31. IEEE Audio Video Bridging Task Group (2010) 802.1Qav – forwarding and queuing enhancements for time-sensitive streams. http://www.ieee802.org/1/pages/802.1av.html
32. IEEE Audio Video Bridging Task Group (2016) 802.1Qch – cyclic queuing and forwarding. http://www.ieee802.org/1/pages/802.1ch.html
33. IEEE P802.3br Interspersing Express Traffic Task Force. P802.3br – standard for ethernet amendment specification and management parameters for interspersing express traffic. https://standards.ieee.org/develop/project/802.3br.html
34. IEEE Time-Sensitive Networking Task Group. 802.1Qbu – frame preemption. http://www.ieee802.org/1/pages/802.1bu.html
35. IEEE Time-Sensitive Networking Task Group (2015) P802.1Qbv (Draft 3.0) – enhancements for scheduled traffic. http://www.ieee802.org/1/pages/802.1bv.html
36. International Standards Organisation (ISO) (1993) ISO 11898-1. Road vehicles – interchange of digital information – controller area network (CAN) for high-speed communication. ISO Standard-11898
37. International Standards Organisation (ISO) (2013) Road vehicles – FlexRay communications system – part 1: general information and use case definition. ISO Standard-17458
38. Jansen K, Solis-Oba R (2003) An asymptotic fully polynomial time approximation scheme for bin covering. Theor Comput Sci 306(1):543–551
39. Kang M, Park K, Jeong MK (2013) Frame packing for minimizing the bandwidth consumption of the FlexRay static segment. IEEE Trans Ind Electron 60(9):4001–4008
40. Khan D, Bril R, Navet N (2010) Integrating hardware limitations in can schedulability analysis. In: 8th IEEE international workshop on factory communication systems, pp 207–210
41. Khan D, Davis R, Navet N (2011) Schedulability analysis of can with non-abortable transmission requests. In: 16th IEEE conference on emerging technologies factory automation, pp 1–8
42. Kreutz D, Ramos F, Esteves Verissimo P, Esteve Rothenberg C, Azodolmolky S, Uhlig S (2015) Software-defined networking: a comprehensive survey. Proc IEEE 103(1):14–76
43. Li W, Di Natale M, Zheng W, Giusto P, Sangiovanni-Vincentelli A, Seshia S (2009) Optimizations of an application-level protocol for enhanced dependability in flexray. In: Design, automation test in Europe conference exhibition (DATE 2009), pp 1076–1081
44. Lincoln B, Cervin A (2002) Jitterbug: a tool for analysis of real-time control performance. In: Proceedings of the 41st IEEE conference on decision and control, vol 2, pp 1319–1324
45. Liu M, Behnam M, Nolte T (2013) An EVT-based worst-case response time analysis of complex real-time systems. In: 8th IEEE international symposium on industrial embedded systems, pp 249–258
46. Liu M, Behnam M, Nolte T (2013) Schedulability analysis of multi-frame messages over controller area networks with mixed-queues. In: 18th IEEE conference on emerging technologies factory automation, pp 1–6
47. Liu M, Behnam M, Nolte T (2014) Schedulability analysis of GMF-modeled messages over controller area networks with mixed-queues. In: 10th IEEE workshop on factory communication systems, pp 1–10

48. Lukasiewycz M, Glaß M. Teich J, Milbredt P (2009) FlexRay schedule optimization of the static segment. In: 7th IEEE/ACM international conference on hardware/software codesign and system synthesis, pp 363–372
49. Lukasiewycz M, Schneider R, Goswami D, Chakraborty S (2012) Modular scheduling of distributed heterogeneous time-triggered automotive systems. In: 17th Asia and South Pacific design automation conference, pp 665–670
50. Mok A, Chen D (1996) A multiframe model for real-time tasks. In: 17th IEEE real-time systems symposium, pp 22–29
51. Mubeen S, Mäki-Turja J, Sjödin M (2011) Extending schedulability analysis of controller area network (CAN) for mixed (periodic/sporadic) messages. In: 16th IEEE conference on emerging technologies factory automation, pp 1–10
52. Mubeen S, Mäki-Turja J, Sjödin M (2012) Extending response-time analysis of mixed messages in can with controllers implementing non-abortable transmit buffers. In: 17th IEEE conference on emerging technologies factory automation, pp 1–4
53. Mubeen S, Mäki-Turja J, Sjödin M (2012) Response time analysis for mixed messages in can supporting transmission abort requests. In: 7th IEEE international symposium on industrial embedded systems, pp 291–294
54. Mubeen S, Mäki-Turja J, Sjödin M (2012) Response-time analysis of mixed messages in controller area network with priority- and FIFO-queued nodes. In: 9th IEEE international workshop on factory communication systems, pp 23–32
55. Mubeen S, Mäki-Turja J, Sjödin M (2012) Worst-case response-time analysis for mixed messages with offsets in controller area network. In: 17th IEEE conference on emerging technologies factory automation, pp 1–10
56. Mubeen S, Mäki-Turja J, Sjödin M (2013) Extending offset-based response-time analysis for mixed messages in controller area network. In: 18th IEEE conference on emerging technologies factory automation, pp 1–10
57. Mubeen S, Mäki-Turja J, Sjödin M (2014) Extending worst case response-time analysis for mixed messages in controller area network with priority and FIFO queues. IEEE Access 2: 365–380
58. Mubeen S, Mäki-Turja J, Sjödin M (2014) Response time analysis with offsets for mixed messages in can supporting transmission abort requests. In: Emerging technology and factory automation (ETFA 2014). IEEE, pp 1–10
59. Mubeen S, Mäki-Turja J, Sjödin M (2015) Integrating mixed transmission and practical limitations with the worst-case response-time analysis for controller area network. J Syst Softw 99:66–84
60. Mundhenk P, Steinhorst S, Lukasiewycz M, Fahmy SA, Chakraborty S (2015) Security analysis of automotive architectures using probabilistic model checking. In: 52nd ACM/IEEE design automation conference (DAC), pp 1–6
61. Natale MD (2006) Evaluating message transmission times in controller area networks without buffer preemption. In: 8th Brazilian workshop on real-time systems
62. Navet N, Song YQ, Simonot F (2000) Worst-case deadline failure probability in real-time applications distributed over controller area network. J Syst Archit 46(7):607–617
63. Neukirchner M, Negrean M, Ernst R, Bone TT (2012) Response-time analysis of the FlexRay dynamic segment under consideration of slot-multiplexing. In: 7th IEEE international symposium on industrial embedded systems, pp 21–30
64. Nolte T, Hansson H, Norstrom C (2003) Probabilistic worst-case response-time analysis for the controller area network. In: 9th IEEE real-time and embedded technology and applications symposium, pp 200–207
65. Pop T, Pop P, Eles P, Peng Z, Andrei A (2008) Timing analysis of the FlexRay communication protocol. Real-Time Syst 39(1–3):205–235
66. Schenkelaars T, Vermeulen B, Goossens K (2011) Optimal scheduling of switched FlexRay networks. In: Design, automation test in Europe conference exhibition, pp 1–6
67. Schmidt K, Schmidt E (2009) Message scheduling for the FlexRay protocol: the static segment. IEEE Trans Veh Technol 58(5):2170–2179

68. Tanasa B, Bordoloi UD, Kosuch S, Eles P, Peng Z (2012) Schedulability analysis for the dynamic segment of FlexRay: a generalization to slot multiplexing. In: 18th IEEE real-time and embedded technology and applications symposium, pp 185–194
69. Tanasa B, Dutta Bordoloi U, Eles P, Peng Z (2011) Reliability-aware frame packing for the static segment of FlexRay. In: Proceedings of the ninth ACM international conference on embedded software, pp 175–184
70. Thiele D, Axer P, Ernst R (2015) Improving formal timing analysis of switched ethernet by exploiting FIFO scheduling. In: Design automation conference (DAC), San Francisco
71. Thiele D, Diemer J, Axer P, Ernst R, Seyler J (2013) Improved formal worst-case timing analysis of weighted round robin scheduling for ethernet. In: Proceedings of the CODES+ISSS, Montreal
72. Thiele D, Ernst R (2016) Formal analysis based evaluation of software defined networking for time-sensitive ethernet. In: Proceedings of the design, automation, and test in Europe (DATE), Dresden
73. Thiele D, Ernst R (2016) Formal worst-case performance analysis of time-sensitive Ethernet with frame preemption. In: Proceedings of emerging technologies and factory automation (ETFA), Berlin, p 9
74. Thiele D, Ernst R (2016) Formal worst-case timing analysis of Ethernet TSN's burst-limiting shaper. In: Proceedings of the design, automation, and test in Europe (DATE), Dresden
75. Thiele D, Ernst R, Diemer J (2015) Formal worst-case timing analysis of Ethernet TSN's time-aware and peristaltic shapers. In: IEEE vehicular networking conference (VNC)
76. Thiele D, Schlatow J, Axer P, Ernst R (2015) Formal timing analysis of can-to-ethernet gateway strategies in automotive networks. Real-Time Syst. http://dx.doi.org/10.1007/s11241-015-9243-y
77. Tindell K, Hansson H, Wellings A (1994) Analysing real-time communications: controller area network (CAN). In: IEEE real-time systems symposium, pp 259–263
78. Vector. CANbedded interaction layer. [Online] http://www.vector.com
79. Yomsi P, Bertrand D, Navet N, Davis R (2012) Controller area network (CAN): response time analysis with offsets. In: 9th IEEE international workshop on factory communication systems, pp 43–52
80. Zeng H, Di Natale M, Ghosal A, Sangiovanni-Vincentelli A (2011) Schedule optimization of time-triggered systems communicating over the FlexRay static segment. IEEE Transactions on Industrial Informatics 7(1):1–17
81. Zeng H, Di Natale M, Giusto P, Sangiovanni-Vincentelli A (2009) Stochastic analysis of CAN-based real-time automotive systems. IEEE Transactions on Industrial Informatics 5(4):388–401
82. Zeng H, Di Natale M, Giusto P, Sangiovanni-Vincentelli A (2010) Using statistical methods to compute the probability distribution of message response time in controller area network. IEEE Transactions on Industrial Informatics 6(4):678–691
83. Zeng H, Ghosal A, Di Natale M (2010) Timing analysis and optimization of FlexRay dynamic segment. In: 7th IEEE international conference on embedded software and systems, pp 1932–1939