Reuse of CAN-based Legacy Applications in Time-Triggered Architectures

R. Obermaisser
Vienna University of Technology, Austria

Abstract

Upcoming car series will be deployed with time-triggered communication protocols due to benefits with respect to bandwidth, predictability, dependability, and system integration. In present day automotive networks, CAN is the most widely used communication protocol. Today, up to five CAN buses and several private CAN networks result from the bandwidth limits of CAN in conjunction with constraints concerning bus utilization aimed at controlling transmission latencies. In this context, the upcoming introduction of time-triggered networks into series production offers the potential to reduce the number of CAN networks by exploiting the high-bandwidth of the time-triggered network instead of CAN buses. Due to the elimination of CAN buses, the resulting reduction of wiring and connectors promises a significant reduction in hardware cost and reliability improvements. In order to support the reuse of existing CAN-based application software, this paper presents a solution for the emulation of a CAN communication service on top of an underlying time-triggered network. By providing to CAN-based applications the same interface as in a conventional CAN system, redevelopment efforts for CAN-based legacy software are minimized. For this purpose, a CAN emulation middleware operates between a time-triggered operating system and the CAN-based applications. In a first step, the middleware establishes event channels on top of the time-triggered communication network in order to support on-demand transmission requests at a priori unknown points in time. The middleware then emulates the CSMA/CA media access protocol of a physical CAN network for passing messages received via event channels to the application in the correct temporal order. Finally, the API of the widely-used HIS/VectorCAN driver provides a handle-based programming interface with support for message filtering and callbacks. A validation setup with a TTP cluster demonstrates that the CAN emulation can handle CAN-based legacy software and a real-world communication matrix provided by the automotive industry.

Index Terms

Computer network performance, distributed algorithms, legacy systems, real-time systems, road vehicle electronics

I. Introduction

Time-Triggered (TT) and Event-Triggered (ET) control are two different paradigms for the construction of the communication service of a distributed real-time system [1]. Communication protocols with ET control (e.g., Controller Area Network (CAN) [2]) are prevalent in current automotive networks, because they offer high flexibility and resource efficiency. However, with the increasing criticality of automotive computer systems (e.g., advanced driver assistance systems [3], X-by-wire [4]), there is an upcoming shift from ET to TT communication protocols [5]. Communication protocols with TT control (e.g., TTP [6], SafeBus [7], FlexRay [8]) excel with respect to predictability, composability, error detection and error

containment. For this reason, in-vehicle electronic systems will be deployed with TT communication protocols beginning in 2007 [9].

At present, CAN [2] is the most widely used automotive protocol with a 100% market penetration. Major advantages of CAN include its high flexibility (e.g., migration transparency, no need to change a communication schedule when adding Electronic Control Units (ECUs)), resource efficiency through the sharing of bandwidth between ECUs, low cost of CAN hardware, and high availability of CAN-based tools (e.g., [10]) and engineers with CAN know-how (e.g., experience in production process, existing maintenance chain, field experience).

In the context of upcoming TT technology in the automotive domain, the availability of a TT communication network with high bandwidth (e.g., 10 Mbps in FlexRay [8]) enables the elimination of one or more of the physical CAN networks deployed in present day cars. The communication resources of a single TT network can be shared for the exchange of both TT messages and CAN messages. In conjunction with integrated ECUs, i.e., ECUs for both CAN-based application software and application software based on TT communication, the sharing of communication and computational resources not only reduces the number of ECUs, but also results in fewer connectors and wires. Fewer connectors and wires not only decrease hardware cost, but also lead to improved reliability. Field data from automotive environments has shown that more than 30% of electrical failures are attributed to connector problems [11].

Nevertheless, despite the shift to TT communication protocols, previous investments motivate the reuse of CAN-based legacy applications. In addition, this strategy allows to retain software with conceivably low field failure rates, i.e., for those applications that have functioned correctly in a large set of cars. In order to minimize redevelopment efforts of legacy software (e.g., in comfort domain, powertrain domain), when migrating existing CAN-based applications to the TT system, there is the need to reestablish the services of the platform, for which the legacy software has been developed for. Among these services of the legacy platform are the operating system services (e.g., OSEK/VDX [12]) and the CAN communication service. The focus of this paper is the establishment of the CAN communication service, i.e., the application's interface to the underlying communication system. For this CAN communication service, three main requirements can be identified, which need to be satisfied for ensuring the correctness of CAN-based legacy applications after the migration to the TT computer system:

1) **Support for ET Control.** CAN legacy software can request message transmissions on-demand at a priori unknown instants. In order to use a common network for both TT messages and CAN message exchanges, both TT and ET control need to be supported by the communication system.
2) **Temporal Properties.** In general, the correct behavior of CAN legacy software depends on the temporal properties of a CAN communication system. Relevant properties include the bandwidth, the transmission latencies, and the temporal order of messages.
3) **Application Programming Interface (API).** Legacy CAN software accesses the communication system through a specific API. The establishment of this API is thus a prerequisite for reuse with a minimum of redevelopment efforts.

This paper presents a solution for the emulation of a CAN network on top of an underlying TT communication protocol. This so-called *Virtual CAN Network (VCN)* can be established on top of different TT communication protocols (e.g., TTP, FlexRay). In contrast to other solutions for ET/TT integration (see Section II), this solution addresses all three requirements identified above. An *event service* maps TT control to ET control by realizing event channels on top of an underlying TT communication protocol for the on-demand transmission of messages at a priori unknown points in time. Event channels support the temporal properties of a physical CAN network through a predefined relationship between the communication schedule of the TT physical network and the bandwidth and latencies of a VCN. In addition, an emulation (denoted as *protocol emulation service*) of the Carrier Sense Multiple Access Collision Avoidance (CSMA/CA) media access control protocol ensures that the temporal message order is identical to the one of a physical CAN network. Finally, the *front-end* is a service that maps the VCN onto the API expected by legacy applications (e.g., API of HIS/VectorCAN driver).

A prototype implementation using the Time-Triggered Protocol (TTP) [6] serves as a target for the validation of the devised solution. The validation activities have included measurements of the transmission latencies, bandwidth, and temporal message order of VCNs with a communication matrix from a series-car provided by an automotive manufacturer as inputs. These results have been compared to the transmission latencies, bandwidths, and temporal message orders of a physical CAN network, as indicated by a MATLAB/Simulink simulation framework of a physical CAN network.

The paper is structured as follows. Section II gives an overview of related work on the integration of ET/TT communication protocols. The system model of a TT computer system with support for VCNs is the focus of Section III. The following sections are devoted to explaining the realization of the architectural services introduced in this system model. Section IV describes the event service for transforming ET into TT communication services. The establishment of the correct temporal message order is the focus of Section V. The front-end for establishing legacy CAN APIs is the topic of Section VI. In combination, these three services (event service, protocol emulation, front-end) cover the three requirements concerning a VCN identified above. Finally, a validation setup with a prototype implementation in Section VII provides experimental evidence on the ability of the VCNs to support the reuse of CAN-based legacy applications.

II. Related Work of Integration of ET and TT Communication

The VCN presented in this paper provides an ET communication services for the dissemination of CAN messages via an overlay network on top of an underlying TT network. This section gives an overview of other solutions for the integration of ET and TT control, all of which perform the integration of the two control paradigms at the Media Access Control (MAC) layer. As depicted in Figure 1, these protocols distinguish two types of reoccurring intervals: ET and TT intervals. The arrows represent the message transmission start instants with the sender node superscripted to each arrow. A TT interval permits a single node to send a message after the a priori specified start instant of the interval. The start and end instants of such a periodic TT message transmission, as well as the respective sending node computer are fixed at

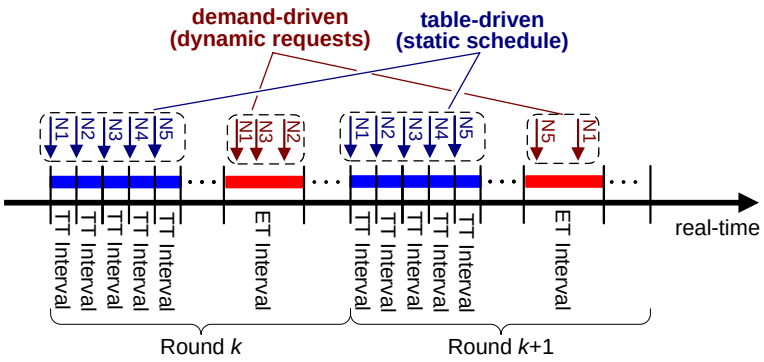

Fig. 1. Communication Intervals for ET and TT Messages

design time. For this class of messages, contention is resolved statically. All TT message transmissions follow an a priori defined schedule, which repeats itself after each communication round. In ET intervals, message exchanges depend on transmission requests from the application (i.e., external control) and the start instants of message transmissions can vary. Furthermore, ET intervals can be assigned to multiple (or all) node computers of the system. For this reason, the MAC layer needs to support the dynamic resolving of contention when more than one node computer intends to transmit a message. During ET intervals a sub-protocol (e.g., CSMA/CA, CSMA/CD) takes over that is not required during TT intervals in which contention is prevented by design.

While TT messages can always be scheduled to fit into the respective intervals at design time, on-demand ET message transmissions require support by the MAC protocol for protecting TT intervals. The mechanism for the delimitation of ET and TT intervals allows to further classify protocols for ET/TT integration at the MAC layer: contention avoidance protocols and contention tolerant protocols.

A. ET/TT Contention Avoidance

Contention avoidance protocols reserve at the end of each ET interval a time interval, in which no message transmissions may be started. The length of this time interval is equal to the maximum message transmission duration of an ET message. Consequently, it is ensured that an ET message transmission can always be completed before the next TT interval starts.

An example for a communication protocol realizing this solution is FlexRay [8]. In each communication round, FlexRay supports a single ET interval denoted as the *dynamic segment* and multiple TT intervals forming a continuous time interval named the *static segment*. The static segment realizes a strict Time Division Multiple Access (TDMA) scheme, while the dynamic segment employs an event-driven mini-slotting sub-protocol. The fixed duration of the dynamic segment is subdivided into mini-slots that identify potential start times of message transmissions. Each node computer counts the number of idle mini-slots and is assigned one or more unique counter values. If during a dynamic segment the bus has been idle for a number of mini-slots equal to one of these counter values, the respective node can send a message. During the transmission of a message, the incrementing of the idle mini-slot counter is paused. Consequently, a smaller counter value gives a node computer a higher priority compared to node computers with larger

counter values. Due to the demand driven access pattern, the reserved bandwidth of a dynamic segment can be shared between node computers. In FlexRay, the time interval reserved for contention avoidance at the end of an ET interval is part of the dynamic slot idle phase.

Time-Triggered CAN (TTCAN) [14] and Flexible Time-Triggered CAN (FTT-CAN) [13] in controlled mode are further examples of protocols integrating ET and TT communication at the MAC Layer with contention avoidance. TTCAN and FTT-CAN are CAN-based master/slave protocols for building a TT communication service on top of CAN, while also permitting ET CAN communication. TT intervals (denoted as exclusive windows in TTCAN) support periodic TT messages, which are scheduled statically by an off-line tool in order to prevent collisions. ET intervals (denoted arbitrating windows in TTCAN) resolve contention dynamically with the CSMA/CA arbitration mechanism of CAN. In TTCAN and the controlled mode of FTT-CAN, a node computer may only send an ET message, if the remaining time interval before the next TT message has a sufficient length for preventing any interference of TT and ET messages.

B. ET/TT Contention Tolerant Protocols

ET/TT contention tolerant protocols do neither restrict transmission start instants within an ET interval nor preempt ongoing ET message transmissions. All ET message transmissions are permitted to finish, thus leading to potential perturbations of the boundaries of TT intervals.

An example for this protocol type is FTT-CAN in *uncontrolled mode* [13]. In analogy to the controlled mode of FTT-CAN, both ET communication and TT communication are supported. Every node participating in TT communication is equipped with a local table that contains information about the TT messages transmitted and received during each communication round. Node computers can start with the transmission of ET messages at arbitrary instants. Through assigning higher priorities to TT messages, it is ensured that TT messages always win in the arbitration process. Nevertheless, the non-preemptive nature of CAN results in transmission jitter of TT messages, in case an ET message is being transmitted when a TT message transmission is scheduled. In the worst-case, a TT interval with one or more TT messages is delayed by the maximum transmission duration of an ET message.

C. Relationship to Virtual Networks

A major reason for choosing an ET overlay network for the realization of the virtual CAN network is the ability to integrate this solution into different TT communication protocols (e.g., TTP [6], FlexRay [8], SAFEbus [7]). Virtual CAN networks build on top of the periodic exchange of state messages, which is supported by all TT communication protocols, while refraining from the exploitation of the MAC layer ET communication service that is specific to each of the protocols.

Unlike the VCN presented in this paper, other ET/TT integration solutions either inherit the limitations of CAN or do not explicitly focus on the reuse of CAN-based legacy applications. By building on top of CAN, FTT-CAN naturally supports CAN-based legacy applications, but also inherits all limits with

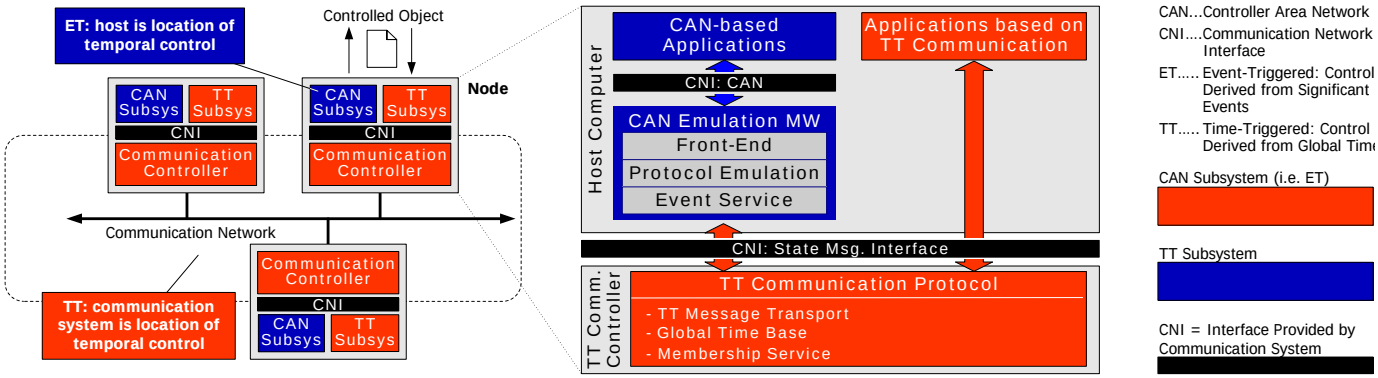

Fig. 2. Integrated Node Computer with CAN Emulation Middleware

respect to bandwidth and dependability of the CAN protocol. For example, CSMA/CA requires bits to stabilize on the channel, i.e., the bit length must be at least as large the propagation delay. The resulting maximum bandwidth of 1 Mbps for a network of 40 m is insufficient, when integrating multiple CAN buses in combination with TT message exchanges on a single network. In uncontrolled mode, there is the additional communication jitter of TT messages induced by standard CAN messages, which results in an adverse effect on the quality of control in jitter-sensitive applications (e.g., control loops).

The ET communication service of FlexRay, on the one hand, does not establish the temporal message order of a physical CAN network through an emulation of the CSMA/CA media access control protocol. Secondly, no mapping to a CAN API, such as the widely used HIS/VectorCAN driver API, has been performed. Both issues are requirements for the reuse of CAN-based legacy applications without redevelopment efforts. Nevertheless, the TT communication service of FlexRay is a suitable as a baseline for the establishment of a VCN as an overlay network.

III. Controller Area Network Emulation

Physically, the distributed system for the integration of CAN and TT communication consists of a set of *integrated nodes* and a single *TT network* that interconnects these nodes. Each integrated node consists of a host computer and a communication controller. The communication controller executes a TT communication protocol to provide a TT message transport service, a global time base, and a membership service. Any communication protocol that provides these services can be used as a basis [15] for the integration of CAN and TT communication. Examples of suitable communication protocols are the Time-Triggered Protocol (TTP) [6] and FlexRay [8], when extended with a membership service. The three services, which abstract from the implementation technology of the underlying TT communication protocol, are explained in the following:

- **TT message transport service.** This service performs periodic TT exchanges of state message. At each node the communication controller (e.g., TTP controller C2 [17], FlexRay Controller MFR4200 [18]) provides a memory element with outgoing state messages that are written by the application and read by the communication controller prior to broadcasting them on the TT network. In addition, the memory element contains incoming state messages that are read by the application and updated by the

communication controller with state messages read from the TT network (i.e., information broadcast by other nodes). This memory element, which is denoted Communication Network Interface (CNI) in TTP and Controller Host Interface (CHI) in FlexRay, is provided by most TT communication protocols with syntactic differences of state messages (e.g., header format) and protocol-specific constraints (e.g., only one message sent by a node per communication round in TTP [6], same size for all state messages in FlexRay [8]).

- **Global time.** In a distributed computer system, nodes capture the progression of time with physical clocks containing a counter and an oscillation mechanism. An observer can record the current granule of the clock to establish the *timestamp* of an event. Since any two physical clocks will employ slightly different oscillators, the time-references generated by two clocks will drift apart. Clock synchronization is concerned with bringing the time of clocks in a distributed system into close relation with respect to each other. IEEE 1588 [19] is an example for a protocol to synchronize nodes of a distributed system to a high degree of accuracy and precision. By performing clock synchronization in an ensemble of local clocks, each node can construct a local implementation of a global notion of time.

  For the emulation of CAN, we require the ability to assign timestamps w.r.t. to a global time base to events, such as a transmission request of the application. These timestamps are the basis for ensuring the correct temporal order of CAN messages, which will be discussed in Section V. For example, in TTP a global time base with a precision down to $1\,\mu$s (depending on clock drifts) is provided by a macrotick counter at the controller-host interface [6]. Similarly, FlexRay [8] offers a *cycle counter* denoting the number of the current communication round (denoted as cycle in FlexRay) and a *macrotick counter* within the cycle.
- **Membership service.** The *membership service* provides consistent information about the operational state (correct or faulty) of nodes [20]. In a TT communication system the periodic message send times are membership points of the sender [21]. Every receiver knows a priori when a message of a sender is supposed to arrive, and interprets the arrival of the message as a life sign of the sender.

  TTP, which has been used in the prototype implementation of the CAN emulation, natively provides a membership service. For other protocols, such a FlexRay, a membership service can be layered on top of the protocol services. For example, in [16] a membership service is presented that is based on a generic model of a TT system that is not restricted to a specific communication protocol.

On top of the communication controller providing these three services, the host computer runs the application software modules, which are either based on TT communication or require a CAN communication service. For this purpose, the host computer of an integrated node (see Figure 2) contains two subsystems: an ET subsystem for CAN-based applications and a TT subsystem for applications based on the periodic exchange of state messages. In the latter execution environment, the application software directly exploits the services of the communication controller. The CAN execution environment employs a *CAN emulation middleware* comprising three layers (event service, protocol emulation, and front-end)

for performing a step-wise transformation of the TT services into a CAN communication service.

The *event service* establishes *Event Channels (ECs)* for the on-demand transmission of messages. Message transmission requests can occur at arbitrary instants, but the dissemination of the messages on the underlying TT network is always performed at the predefined global points in time of the TDMA slots. The middleware service for *protocol emulation* exploits the ECs provided by the event service in order to realize a Virtual CAN Network (VCN). The protocol emulation establishes the temporal message order of a physical CAN network by performing at run-time a simulation of the CSMA/CA media access control strategy of a physical CAN network within each node of the TT system. The third middleware service establishes a CAN-based higher protocol and provides an *API* by which application software can access a VCN. The *front-end* provides the operations for the transmission and reception of messages and the reconfiguration of the CAN communication system (e.g., setting of message filters). The front-end also implements interrupt mechanisms via callbacks that enable the application to react to significant events, such the reception of a message or the occurrence of a fault.

The following sections provide a detailed discussion on the event service, the protocol emulation, and the front-end.

IV. Event Service

The purpose of the *event service* is the mapping of the state message interface provided by the TT communication controller onto an event message interface that is the basis for all higher layers (i.e., the protocol emulation, the front-end, and the application). The event message interface comprises queues with support for ET on-demand transmission requests.

A. State Message Interface

The state message interface of the underlying TT communication system is also known as a temporal firewall [22] due to the absence of control flow from the application to the communication system. The interface between the communication system and the application software are state messages that are read (in case of a message received by the node) or written (in case of a message sent by the node) by the communication system at a priori specified instants w.r.t. to a global time base. A temporal firewall is based on the principle of *updates in place*, i.e., idempotent state messages are overwritten when a more recent version of the state messages becomes available. Updates in place require each message to be self-contained, i.e., the data contained in a message is an absolute value that can be interpreted independently from previous versions of the message. This self-contained nature of messages is denoted as *state semantics*.

The application software is aware of the predefined instants, at which the communication system accesses the state messages. The application software uses this a priori knowledge to fulfill the sender and receiver obligations of a temporal firewall. The sender obligation consists in updating those state message that are sent by the respective node at a sufficiently low update period that ensures temporal accuracy

of the state message at the receivers. Based on the a priori knowledge about the temporal accuracy of the real-time images in the temporal firewall, the consumer must sample the information in the temporal firewall with a sampling rate that ensures that the accessed real-time image is temporally accurate at its time of use.

B. Event Message Interface

In contrast to the TT applications, CAN-based applications do not know upfront about the instants of messages exchanges at the communication system. A CAN communication system reacts dynamically to transmission requests, transmitting messages (normal CAN frames and request CAN frames [2]) only in response to a transmission request from the application. The transmission requests from CAN applications are not synchronized to the underlying TT communication protocol. Furthermore, in general only an average message transmission frequency is known for ET applications. For example, in CAN-based automotive systems a bus utilization in the range of 50% is demanded for non safety-critical applications [23]. The bus utilization is a statement about the average frequency of message transmissions on the CAN bus, permitting the exceeding of the number of messages per second during limited intervals of time. The underlying TT communication system, however, allows to send exactly n data bytes during each communication round, where n (e.g., up to 254 in FlexRay [8], up to 240 in TTP [6]) depends on the duration of the CAN subslot and the bitrate of the TT physical network. Hence, it can occur that within a communication round multiple transmission requests occur, while none occur in other communication rounds. In order to buffer messages that cannot be disseminated immediately, the interface of the event service to the higher layers are message queues. With messages queues, the event service also supports limited intervals of time during which the transmission of more messages is requested by the application than can be disseminated via the communication system.

C. Mapping of State Message Interface to Event Message Interface

The media access protocol of the TT network is TDMA, which statically divides the channel capacity into a number of slots and assigns to each node a unique slot that periodically reoccurs at a priori specified global points in time. Support for CAN communication results from the temporal subdivision of the communication resources provided by this TDMA scheme. Each node's slot is subdivided into two subslots, namely a slot for TT communication and a slot for the ET dissemination of CAN messages (see Figure 3).

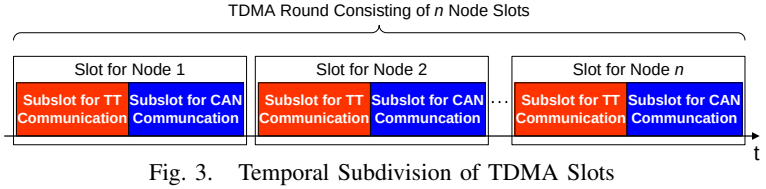

Fig. 3. Temporal Subdivision of TDMA Slots

The event service disseminates the messages in the outgoing queues via the bandwidth provided via the CAN subslots. Although message transmission requests can occur at arbitrary instants, the dissemination

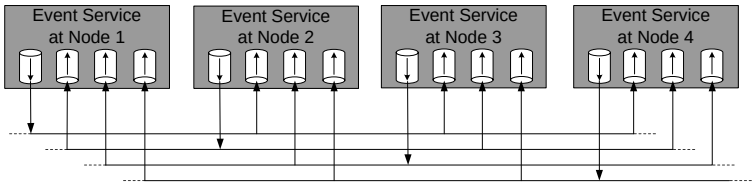

Fig. 4. Event Channels on top of the Time-Triggered Physical Network

of the messages on the underlying TT network is always performed at the predefined global points in time of the CAN subslots. The event services performs a sampling of the ET message transmission requests and defers their transmission until the next CAN subslot of the node occurs in the TDMA scheme. During each communication round, the event service transfers CAN messages to be sent from the outgoing message queues of the event message interface into the temporal firewall interface, namely into the state variables that are broadcast by TT communication controller. In addition, the event service retrieves received messages at the temporal firewall interface and forwards them into the incoming messages queues. Message fragmentation, i.e., the filling of CAN subslots with packets from as many messages as possible, ensures that the CAN subslots are optimally utilized in case of varying message sizes. In addition, message fragmentation decouples the duration of CAN subslots from the maximum message sizes. CAN subslots can be shorter than a CAN message (e.g., CAN subslot with a single byte), thus permitting short round lengths despite the support for CAN communication. Short round lengths are important for control applications that require high update frequencies of state variables.

The event service constructs for each node an *Event Channel (EC)*, via which the node can broadcast its event messages to all other nodes of the distributed system (i.e., point-to-multipoint topology). In order to support a general communication topology, in which each node can transmit CAN messages, a system with n nodes employs n ECs with an equal number of CAN subslots (see Figure 4).

An overview of the functionality of the event service for establishing one outgoing EC and multiple incoming ECs at a node is depicted in Figure 5. The part that is responsible for the transmission of event messages (starting at Line 8) is activated prior to the node's CAN subslot in the TDMA scheme of the TT communication protocol. The activation occurs when the current time t_{now} is equal to the start instant t_{slot} of the node's CAN subslot subtracted by the worst-case execution time d_{mw} of the CAN emulation middleware. This activation time ensures that the update of the outgoing state message is finished before the TT communication controller reads the state message at instant t_{slot}. For the fragmentation of event messages, the event service maintains a partially sent event message m_s. Upon each invocation, the event service extracts a packet from this partially sent event message and places the packet into the outgoing state message. Furthermore, the event service fetches additional event messages from the protocol emulation in case space for packets of more than one event message is available in the state message.

The reception part in Figure 5 starts at Line 16. The event service maintains a vector containing for each node i a partially assembled event message $m_r[i]$. Upon the (TT) arrival of a state message from the network, the event service extracts the packets contained in the state message. The event service concatenates each packet with the partially assembled event message from the respective node. After the

```
1   // Variable description
2   // m_r[]      : vector with partially received event msg. from node i in element i
3   // m_s        : partially sent event msg.
4   // t_slot     : start instant of this node's next CAN subslot
5   // d_mw       : worst-case execution time of CAN emulation middleware
6   // g[]        : membership vector

7   // Transmission of event msgs.: update of outgoing state msg. s prior to node's own slot
8   when ( t_now = t_slot – d_mw )
9     s = empty msg.
10    while (space available in s)
11      if ( m_s = empty ) m_s ← protocol emulation   // retrieve next event msg. to be sent
12      extract packet p from m_s                     // extract packet
13      s | p                                         // concatenate state msg. s with packet
14    s → TT network                                  // no more space available in s

15  // Reception of event msgs.: process incoming state msg. s from TT network
16  when ( (node i via TT network → s_i = {p_1,p_2,...}) ∧ (g[i] = 1) )
17    for every p_j ∈ s_i do
18      m_r[i] | p_j                                  // concatenate partially received msg. with packet
19      if m_r[i] completely assembled
20        m_r[i] → protocol emulation                 // forward assembled event msg. to next layer
21        m_r[i] = empty msg.                         // start assembly of next msg.
```

Fig. 5. Mapping between State and Event Messages through the Event Service

assembly of an event message is completed, the message is forwarded to the protocol emulation.

For error handling, the event service exploits the membership vector $\overrightarrow{g}$ ($g_i = 1$ if node i is correct, 0 otherwise) provided by the TT communication protocol. The state message corresponding to a particular sender node in the CNI is only processed, if the sender node is classified as correct according to the membership vector.

Note, that the realization of an EC as described above, i.e., by layering event messages on top of a service for the TT exchange of state messages supports different underlying communication protocols. As described in Section III, most TT communication protocols provide at the controller-host interface state messages that are read or written by the communication system at a priori defined instants (i.e., temporal firewall interface). For example, in FlexRay the state messages exchanged in the so-called static segment [8] are accessed by the application software via this temporal firewall interface. In TTP, all messages exchanged in a communication round are provided to the application software via the temporal firewall interface.

V. Protocol Emulation

Different types of message orderings (e.g., total order, FIFO order, causal order [24]) have been defined for distributed systems, because the interpretation of a message can depend on precedent messages. The protocol emulation focuses on the temporal order of the message receive instants, where the temporal order denotes the order of the instants on the timeline [25].

The CAN protocol emulation is a middleware service that aims at the reuse of legacy applications with a minimum of redevelopment and retesting efforts. The CAN protocol emulation ensures that a VCN exhibits the same temporal order of the receive instants as a physical CAN network provided that the

virtual network gets as input the same set of messages, in particular with identical transmission request instants. For this purpose, a *protocol emulation algorithm* is executed in every node to perform at run-time a simulation of a physical CAN network. This algorithm uses as inputs both messages for which a transmission has been requested by the local application (i.e., at the same node) and messages received via the event service from other nodes. The protocol emulation algorithm exploits the global timebase of the underlying TT protocol in order to capture in a timestamp the request instant of each message. The run-time simulation of a physical CAN network takes into account these timestamps of the request instants, as well as the message priorities, and the message lengths. Based on these inputs, the protocol emulation algorithm computes for each message the send instant when the message would have been sent on a physical CAN network. Messages are passed to the application in the order of ascending message send instants. Due to the non-preemptive nature of CAN, this strategy also ensures ascending message receive instants and thus the correct temporal message order. The protocol emulation algorithm is fully decentralized and runs in all nodes participating in the CAN emulation. It forms the middleware service for protocol emulation, which exchanges CAN messages with its adjacent service layers, namely the front-end and the event service.

The protocol emulation depends on the underlying event service in order to establish the temporal message order of a physical CAN network. For a precise emulation, the protocol emulation requires the event service to provide to each node the same bandwidth that would be available to the node in a physical CAN network (e.g., 500 kbps when emulating a high-speed automotive CAN bus). If no knowledge concerning the distribution of the bandwidth consumption is available, then the static allocation of communication resources via CAN subslots results in the need to provide at the event service a total bandwidth that is n times (where n is the number of nodes) as large as the bandwidth of the emulated physical CAN network. However, in many cases a priori knowledge about the maximum bandwidth consumption of an automotive node is availble, e.g., expressed as a fraction of the overall bandwidth, which allows to considerably reduce the resource requirements of the VCN.

A. Temporal Message Order of Event Channels

If the application software requests the transmission of a message m at instant t_{request}, the communication system will result in a delay until the complete message (i.e., the last bit of the CAN frame) has arrived at the receivers at instant t_{receive}. This communication system induced delay consists of the access delay d_{access} and the transmission delay $d_{\text{transmission}}$. $d_{\text{transmission}}$ is determined by the bit rate of the network and the size of the message. The access delay d_{access} results from the media access protocol and the set of competing messages. On a physical CAN network, the access delay of a message comprises the remaining transmission delay of the currently transmitted message (since CAN is non-preemptive) and the transmission delays of all higher priority messages competing for bus access. The arbitration mechanism of the CSMA/CA media access protocol exploits the dominant and recessive states of the bus for resolving contention. After an idle bus in the recessive state for at least 7 bit times, which is the duration of the

end-of-frame delimiter, one or more nodes can start the transmission of a message. All CAN messages start with the transmission of the 11 or 29 bit identifier, the bits of which are subject to arbitration. During the transmission of each bit of the identifier, nodes compare the state of the bus with the sent bit. If a sent recessive bit is overwritten by a dominant bit, the respective node stops the message transmission, leaving the bus to the higher priority message. In this case, the access delay of the lower-priority message is increased by the transmission delay of the higher-priority one.

Without protocol emulation, ECs can exhibit a different temporal order of the receive instants compared to a physical CAN network due to the access delays caused by the underlying TT communication system. Firstly, messages from different nodes do not dynamically compete for the shared medium. All conflicts are resolved with a static schedule. Consequently, the message priorities (specified via the message identifiers) of two messages sent at separate nodes have no influence on the message send instants. The nodes do not share bandwidth and thus there is no dynamic decision on which message to transmit first. The send instant of each message is determined locally by the other transmission requests at this node. While this non-interference between nodes is beneficial from a fault isolation and composability perspective (i.e., a constructive integration of a distributed system), the missing contention between nodes also implies that legacy applications can perceive a different message order compared to a physical CAN network.

Secondly, although an EC accepts transmission requests at arbitrary instants, the dissemination of messages occurs only in the statically assigned CAN subslot of the respective node. After the transmission request of a message at a particular node, the message remains in a message queue until the respective node's CAN subslot occurs in the TDMA scheme. Since the CAN communication activities are not synchronized to this TDMA scheme, the access delays can include the duration of a complete TDMA round until the slot of the respective node reoccurs. The mapping from the continuous time of the message request instants to the discrete time of the start instants of CAN subslots corresponds to a sampling of CAN messages once every TDMA round. The access delay of a message depends on the instant of the transmission request relative to the start instant of the node's slot in the TT communication schedule.

B. Phases of a CAN Message Transmission on a Virtual CAN Network

The transmission of a CAN message is requested at the front-end, which provides the API to the application software. Based on the information provided by the application (i.e., identifier, remote transmission request flag, 0-8 data bytes), the front-end builds a timestamped CAN message (step 1 in Figure 6). A timestamp t_{request} stores the transmission request instant w.r.t. the global time base established by the underlying TT communication protocol, i.e., the instant at which the CAN emulation at the sending node was handed over the message for being sent.

In a second step, the timestamped CAN message is broadcast to all other nodes of the distributed system via ECs. The timestamped CAN message informs the protocol emulation at other nodes about the message transmission request. In addition, the front-end forwards the timestamped CAN message to the local protocol emulation, i.e., the protocol emulation located at the node at which the transmission request

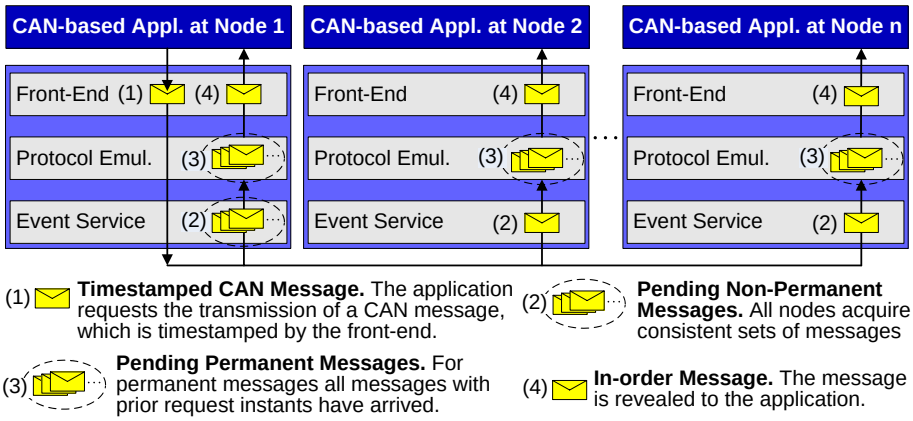

Fig. 6. Phases of a CAN Message Transmission

has been issued. In conjunction with the consistency properties of the underlying TT communication service (which is the foundation for the ECs and the exchange of CAN messages), the CAN emulation at each node acquires a consistent set of messages that is used as input for message reordering through the simulation of the CSMA/CA protocol. The timestamped CAN messages arriving at a node's protocol emulation service are not immediately revealed to the application, but remain pending in intermediate data structures until the correct message order can be established.

The timestamped CAN messages that arrives via ECs at the protocol emulation are initially *pending non-permanent messages* (step 2 in Figure 6). A message m with a transmission request instant t_{request} is non-permanent, if future messages, i.e., messages that have not yet arrived at the protocol emulation, can exhibit an earlier request instant than m. The notion of message permanence is based on the definition in [1]. Since messages with an earlier request instant can precede m in the temporal message order, message m must become permanent before it can be used in the simulation.

After passing the *permanence test* described in Section V-C, a pending message becomes permanent (step 3 in Figure 6). For a *permanent message*, it is ensured that all future messages will possess later request instants. The simulation determines the temporal message order based on the pending permanent message as its input. After a pending message has been sent in the simulation of the physical CAN bus (see Section V-D), the message is denoted as *in-order*. As part of step 4 in Figure 6, the message is forwarded to the front-end to reveal it to the application software.

C. Permanence Test

A message m_1 is *permanent* at instant t_p, if it is known that no message m_2 with an earlier or equal request instant $(m_2.t_{\text{request}} \leq m_1.t_{\text{request}})$ can be received at a later instant t $(t > t_p)$ via an EC.

For determining permanence, one can exploit the fact that transmission request timestamps of messages received via an EC are monotonically increasing. The monotony of the transmission request timestamps results from the fact, that for every EC there is only a single sender (i.e., the protocol emulation layer at the respective node), which exclusively sends messages via the EC.

In order to determine the permanence of messages, the protocol emulation maintains a vector $\overrightarrow{t_{\text{latest}}}$, which contains a timestamp for every sender. The element i of this vector contains the message request instant of the most recent message received from sender i. Since the request instants of messages from a particular sender are monotonically increasing, the element associated with the sender in $\overrightarrow{t_{\text{latest}}}$ represents a temporal bound for subsequent messages, i.e., all future timestamped CAN messages must contain a later request instant. Furthermore, the permanence test exploits the global consistent membership vector $\overrightarrow{g}$ ($g_i = 1$ if node i is correct, 0 otherwise) provided by the TT communication protocol in order to exclude faulty nodes. In case a node is not in the membership (i.e., $g_i = 0$), the node's element in $\overrightarrow{t_{\text{latest}}}$ is not used in the permanence test. Otherwise, a node with a crash failure would prevent the messages of other nodes from becoming permanent.

A sufficient condition for the permanence of a message is that its request instant is earlier than all temporal bounds for message request instants of correct nodes (from a total of n nodes) in the vector $\overrightarrow{t_{\text{latest}}}$:

Permanence Test $\quad \overset{n}{\underset{i=1}{\forall}} \left(m.t_{\text{request}} < t_{\text{latest},i} \vee g_i = 0 \right) \rightarrow m$ permanent

D. Message Ordering

The temporal ordering of messages occurs through a simulation of a physical CAN network at run-time, where simulated message transmissions represent the simulation steps. The current simulation time is specified by the instant $t_{\text{idlestart}}$. $t_{\text{idlestart}}$ is a special instant that separates the messages which have been sent on the simulated CAN bus from those that have not. $t_{\text{idlestart}}$ marks the beginning of idleness on the simulated CAN bus. The message transmissions before $t_{\text{idlestart}}$ are already fixed, i.e., no later transmission requests can result in a modification of the sequence of message transmissions. Consequently, $t_{\text{idlestart}}$ also separates the ordered messages from the non-ordered ones.

In case the simulation time lies before the minimum request instant of a future timestamped CAN message ($t_{\text{idlestart}} < \min_i(t_{\text{latest},i})$ for all nodes i) and one or more pending permanent messages are available, a simulation step can be taken. Out of the set of pending permanent messages, the protocol emulation chooses a message for the next simulation step based on the request instants and the message priorities. After the simulation step, the selected message becomes in-order and is transferred from the protocol emulation to the CAN front-end. Simulation steps are executed until no more pending permanent messages are available or a future timestamped CAN message can exhibit an earlier request instant than the current simulation time.

E. Algorithm

The protocol emulation service executes the permanence test and the reordering of messages via the algorithm in Figure 7. This algorithm operates on two data structures, namely a heap of non-permanent pending messages and a heap of permanent pending messages. The elements of this heap are timestamped CAN messages. The primary key used for ordering in this heap is the sum of the message's request instant

```
1   // Variable description
2   //   t_latest[]          :  vector containing most recent request instants
3   //   g[]                 :  membership vector
4   //   H_nonperm           :  heap of non-permanent pending messages
5   //   H_perm              :  heap of permanent pending messages
6   //   t_idlestart         :  start of idle interval on simulated CAN bus
7   //   t_request           :  transmission request instant of a message

8   when (front-end → m = ⟨t_request, id, data, this node, 0⟩)
9     t_latest[this node] = m.t_request
10    insert m into H_nonperm
11    m → event service

12  when (event service → m = ⟨t_request, id, data, sender node, 0⟩)
13    t_latest[sender node] = m.t_request
14    insert m into H_nonperm

15  // Permanence test
16  when ( H_nonperm ≠ ∅ ∧ ∃n for which t_latest[n] or g[n] has changed)
17    // get non EC-permanent message with smallest timestamp
18    get message m = ⟨t_request, priority, data, 0⟩ from top of H_nonperm
19    if ( ∀n (t_latest[n] ≥ m.t_request) ∨ (g[n] = 0) )
20      // m is permanent
21      remove m from H_nonperm
22      m.t_request := max(m.t_request, t_idlestart)
23      insert m into H_perm

24  // Message Ordering
25  when ( H_perm ≠ ∅ ∧ ∄n t_latest[n] < t_idlestart)
26    get message m = ⟨t_request, priority, data, d_access⟩ from top of H_perm
27    d_m = transmission duration of m
28    remove m from H_perm
29    m → front-end
30    t_idlestart = max(m.t_request, t_idlestart) + d_m
31    for every message n = ⟨t_request, priority, data, d_access⟩ ∈ H_perm  do
32      n.d_access = max(t_idlestart − n.t_request, 0)
33    reorder H_perm
```

Fig. 7. Overview of Protocol Emulation Algorithm

and the message's access delay. The secondary key is the message priority (identifier). The reason for selecting heaps as the data structures for pending messages is the need for the repeated retrieval of messages with the smallest request instant plus access delay and highest priority.

A message is inserted into the heap of non-permanent pending messages either when a message arrives from the network (i.e., the event service passes the messages to the protocol emulation in line 12) or after the application software has issued a transmission request at the front-end (line 8). In both cases, the front-end at the sending node has already set the transmission request timestamp of the timestamped CAN message.

The permanence test in line 15 is triggered by a change of the request instant vector $\overrightarrow{t_{\text{latest}}}$ or the membership vector $\overrightarrow{g}$. The protocol emulation reads the message from the top of the heap of non-permanent messages. If the permanence test gives a positive result, the message is removed from the heap of non-permanent and inserted into the heap of permanent pending messages. The retrieval of a message

from the top of the heap with the subsequent evaluation of the permanence condition proceeds until the heap becomes empty or the permanence test fails. As soon as the permanence test fails for a message, the permanence checking is finished, because the ordering of the heap ensures that the permanence test also fails for all other messages in the heap.

A simulation step in the simulation of the physical CAN network is triggered by the availability of permanent pending messages (see line 23) and a simulation time that is earlier than all elements of the request instant vector $\overrightarrow{t_{\text{latest}}}$. The protocol emulation removes the message m from the top of the heap of permanent pending messages and forwards the m to the front-end (invocation of a message push function of the front-end). Subsequently, the simulation time and the access delays of the pending messages are updated using the transmission duration $d_{\text{transmission}}$ of message m, which depends on the length of m, the identifier type, the bandwidth of the emulated CAN network, and the bit stuffing overhead.

Since the primary keys of messages in H_{perm} have changed (i.e., different access delays), it is necessary to perform a reordering of the heap. In general, multiple messages will now be ordered by the secondary key (message identifier). For these messages, the selection for transmission in the protocol emulation will occur based on the message priority, which corresponds to the media access control strategy of CAN.

VI. Front-End

In present-day automotive ECUs, CAN drivers establish generic APIs towards the underlying CAN communication system. These APIs abstract from any particular CAN controller chip and provide to the application software a generic interface that abstracts from low-level details (e.g., register set of a particular CAN controller, message buffer configurations like Full-CAN and Basic-CAN). CAN-based APIs facilitate the separation of the hardware of an ECU from its embedded software, which has been identified as a key requirement for the reuse of automotive software [26].

The front-end of the CAN emulation middleware realizes the API of the HIS/VectorCAN driver from the OEM Initiative Software [27]. The reason for choosing HIS/VectorCAN driver over other higher protocols and APIs (e.g., CANopen [28], DeviceNet [29]) is the wide use and support in the automotive domain by vehicle manufacturers Audi, BMW, DaimlerChrysler, Porsche, and Volkswagen[1]. Nevertheless, due to the modular structure of the CAN emulation with its three layers, only a single layer – the Front-End – needs to be replaced in order to establish a different CAN API. Figure 8 gives an overview of the standard software core of an automotive ECU [26]. While the event service and the protocol emulation take over the responsibility to emulate the CAN hardware, the establishment of the CAN driver is within the responsibility of the front-end.

In compliance with the specification of the HIS/VectorCAN driver, the front-end of the CAN emulation middleware offers to the application one or more *message handles*, each being assigned to a specific CAN message, which is similar to a full CAN message buffer, or to a range of messages which is similar to a basic CAN message buffer. A message handle is either a transmit object or a receive object

[1] www.automotive-his.de

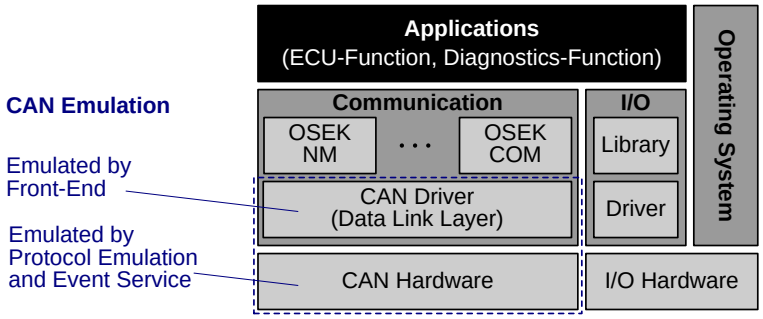

Fig. 8. Model of an Automotive ECU

and incorporates a CAN identifier, a data length code, a pointer to the CAN frame data, and pointers to callbacks. Based on filter masks, each message handle is associated with a subset of the identifiers representing the CAN namespace. Thereby, the application can selectively decide which messages are accepted, i.e., stored in message buffers and triggering callbacks.

Callbacks are sporadic computational activities, which are registered at the front-end. The occurrence of a triggering event causes the activation of the callback through the front-end. The front-end supports the following types of callbacks:

- **Error handling callbacks.** In these callbacks, the application can react to the bus off state, to fatal errors, overrun errors, and syntactically incorrect CAN messages (e.g., invalid data length code).
- **Transmission callbacks.** The transmission callbacks include a pretransmit function, which can be used for setting the data bytes of an outgoing CAN message, as well as a confirmation function that serves as an acknowledgment of a successful message transmission.
- **Reception callbacks.** These callbacks comprise reception notification functions, e.g., for user-defined identifier ranges and specific message handles.

A. Transmission Path

The CAN-based application initiates the transmission process by invoking a *transmit operation* at the API provided by the front-end. The application identifies the CAN message that shall be sent via a message handle. The invocation of the transmit operation represents a transmission request from the application, which triggers the activities comprising the transmission path depicted in Figure 9.

After being invoked by the transmit operation, the front-end first checks whether the transmission path is enabled. In the next step, the front-end searches for the entry in the transmit data structure that matches the handle specified by the application. If a *pre-transmit callback* is registered for the handle, it is now called in order to give the CAN-based application the ability to execute application-specific code prior to the message transmission. In particular, the CAN-based application is passed a reference to the message handle, thus allowing the application to modify the contents of the CAN message (identifier, data length code, data bytes). The front-end, then, passes the message to the protocol emulation in order to broadcast the message via the VCN and invokes the *transmission-start callback*.

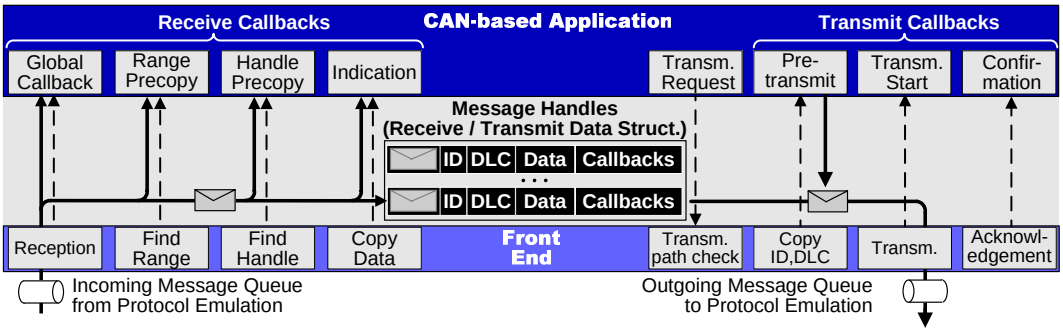

Fig. 9. Message Transmission and Reception Path

The front-end exploits the membership information provided by the underlying TT communication protocol in order to determine whether a sent message has been successfully received by all correct nodes. The membership information serves as a substitute for the acknowledgment bit of a physical CAN network. A successful transmission from a node is indicated through a set membership bit of the node a TDMA round after a message has been sent. In this case, the front-end invokes the *confirmation callback*.

B. Reception Path

In contrast to the transmission process, the reception process is triggered by a message received from a VCN. The front-end checks upon each invocation whether incoming messages are present in the queue forming the interface towards the protocol emulation (see Figure 9). If a message is present, the front-end invokes the *global callback*, which is a common callback function for all messages. The front-end, then, determines if the message identifier matches one of four identifier ranges that are defined by statically set bit masks. Each of the four ranges possesses a *range-precopy callback* function, which is called in case of an identifier match.

In the next step, the front-end determines whether the receive data structure contains a message handle matching the identifier of the received message. Thereby, the front-end implements message filtering, because the message is discarded when there is no match with a handle in the receive data structure. After the execution of the corresponding *handle-precopy callback*, the front-end writes the contents of the message (identifier, data length code and data bytes) into the message handle's entry in the receive data structure. Finally, an *indication callback* in the CAN-based application is called in order to allow the CAN-based application to process the new message in the receive data structure.

VII. Results and Discussion

A prototype implementation of a VCN on top of a physical network running the Time-Triggered Protocol (TTP) has served as the validation platform [6]. The underlying TTP network has provided two redundant channels with a bitrate of 25 Mbps. The three layers of the CAN emulation have been subject to test applications and real-world automotive communication matrices provided by a vehicle manufacturer. The event service has proven effective to handle aperiodic and sporadic message transmissions with comparable latencies as a physical CAN network and superior bandwidth. With enabled protocol emulation, the VCN has also exhibited the exact same temporal message order as a physical CAN network. Finally, the conformance of the API with the HIS/VectorCAN driver specification has been validated with with the diagnostic protocol stack from a vehicle manufacturer.

In order to compare the behavior of the VCN with a physical CAN network, a MATLAB/Simulink simulation of a physical CAN network has served as a reference point. The validation of the event service and protocol emulation has occurred through test applications executed in both the implementation of the VCN and a MATLAB/Simulink framework. The test applications have performed transmission requests at predefined instants and included sequence numbers and timestamps denoting the transmission request instants in the broadcasted CAN messages.

Furthermore, this sections discusses the effects of different TT communication schedules (i.e., different duration of communication rounds) on the delay of messages exchanges via a VCN.

A. Measurement Framework – Virtual CAN Network

The measurement framework employs an implementation of a VCN on a cluster with four TTP MPC855 single board computers [30] as the nodes. Each node is equipped with an MPC855 PowerPC from Freescale clocked with 80 MHz and contains the C2 (AS8202) TTP communication controller. The TTP MPC855 single board computer uses the embedded real-time Linux variant Real-Time Application Interface (RTAI) [31] as the operating system, combining a real-time hardware abstraction layer with a real-time application interface for making Linux suitable for hard real-time applications [32]. The CAN emulation middleware has been realized via a time-triggered RTAI-task, which has been periodically invoked in each communication round by a control signal from the communication controller. For this task a worst-case execution time of 80 μs has been observed in the measurements.

A distributed measurement application uses off-line computed message transmission request tables as inputs for the VCN. The measurement application at every sender node accesses the VCN and requests message transmissions at the instants specified in the request table (see Figure 10). The table also determines the length and identifier of each transmitted message. The data area of the CAN message contains a message index, which uniquely identifies a particular message transmission request along with the node from which the message originated.

Whenever a message m is received, the respective node assigns a timestamp $m.t_{\text{receive}}$ to the received message and computes the transmission latency ($m.d_{\text{latency}} = m.t_{\text{receive}} - m.t_{\text{request}}$) by exploiting the global

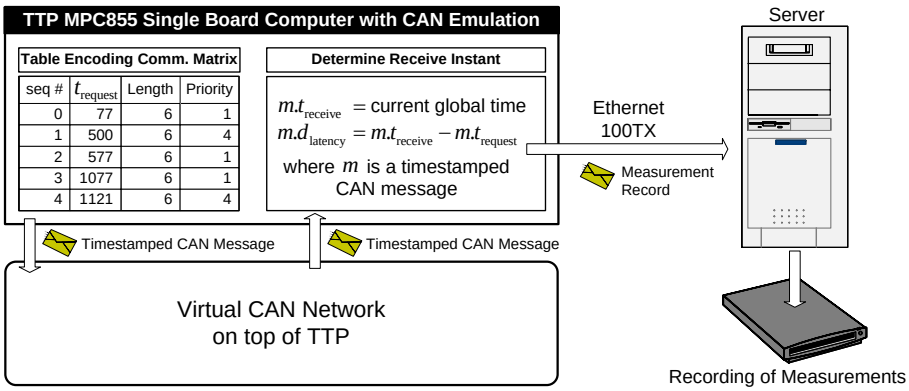

Fig. 10. Measurement Framework

timebase provided by TTP. $m.t_{\text{request}}$ denotes the point in time at which the message needs to be sent according to the off-line computed message transmission request tables. The transmission latency $m.d_{\text{latency}}$ incorporates queuing delays of the VCN at the sender, latencies of the underlying network and execution times of the CAN middleware. A measurement record is constructed with the index of the sending node, the sequence number of the message, the receive instant $m.t_{\text{receive}}$, and the transmission latency $m.d_{\text{latency}}$.

$$\langle \text{sender node},\ \text{msg. sequence number},\ m.t_{\text{receive}},\ m.d_{\text{latency}} \rangle \tag{1}$$

This measurement record is stored in an Ethernet message and transferred to a workstation. The workstation collects the measurement records from the TTP MPC855 single board computers for a later analysis.

B. Simulation Framework – Physical CAN Network

The simulation framework based on the MATLAB/Simulink toolbox TRUETIME provides information about the behavior of a physical CAN network, when provided with a particular message pattern as input. TRUETIME [33] is a MATLAB/Simulink-based simulation toolbox for real-time control systems. TRUETIME supports the simulation of the temporal behavior of tasks in a host computer, as well the simulation of the timing behavior of communication networks. For this purpose, it offers two Simulink blocks: a computer block and a network block. In the framework, a TRUETIME network block has been parameterized with the CSMA/CA medium access control protocol of CAN, while the nodes are modeled by TRUETIME computer blocks. In every node, a task is executed that transmits messages according to the message transmission request table as described in the measurement framework. At the points in time specified in this table, the task passes CAN messages to the TRUETIME network. Each CAN message is assigned the priority and length as specified in the table. One of the nodes also hosts a reception task and writes the message sequence numbers and the transmission latencies into a file for a later analysis.

C. Results – Comparison of Physical and Virtual CAN Networks

For the validation of the CAN emulation, a real-world automotive communication matrix has been used as input for both the implementation of VCNs and a simulation of a physical CAN network. Based

on the communication matrix, the message transmission request tables have been computed in order to parameterize the distributed measurement application of the VCN and the sender tasks of the TRUETIME computer blocks. The communication matrix originates from a powertrain network and consists of 102 periodic messages. The message periods range from 3.3 ms to 1 s, the number of data bytes is between 2 and 8 bytes. Messages comply with the standard-CAN format [2] and contain 47 control bits, thereby resulting in a total message size between 47 and 111 bits. The overall network bandwidth required for the exchange of these messages is 300 kbps.

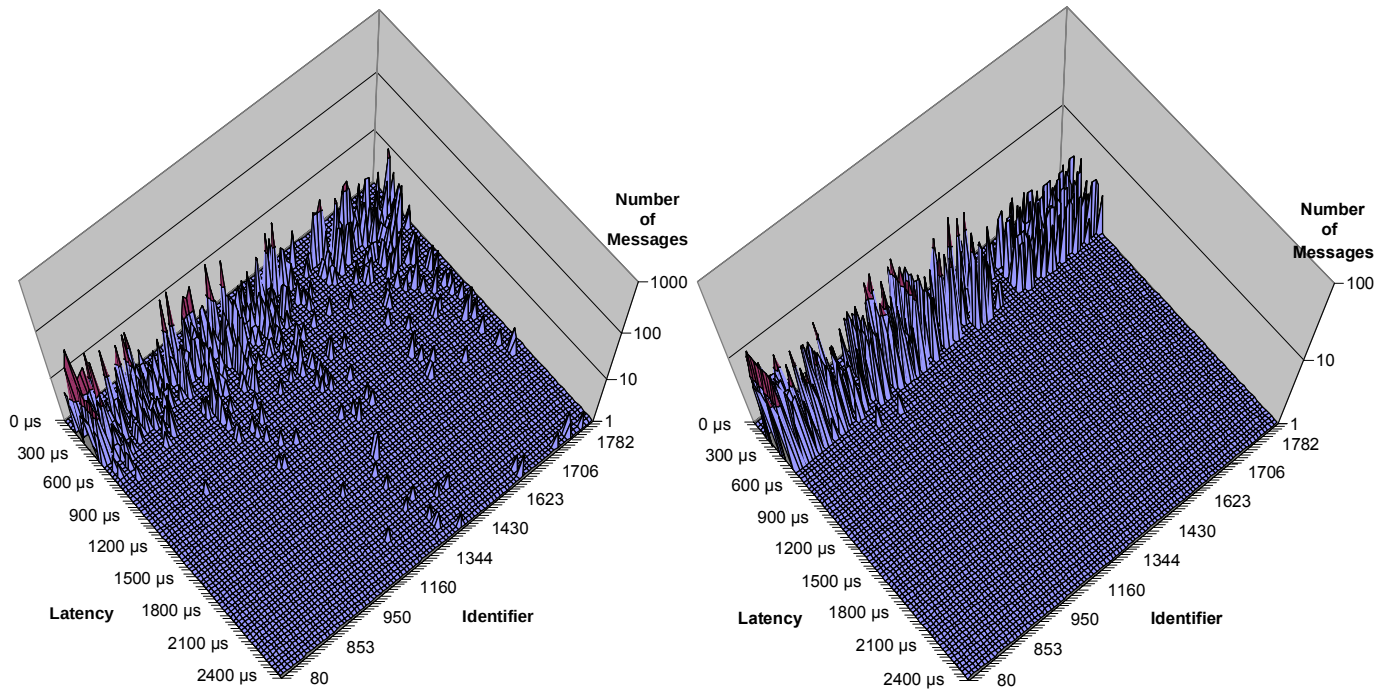

Fig. 11. Simulation Results (left) and Measurement Results (right) with Logarithmic Scaling of z-axis

At the underlying TT communication schedule, the communication schedule provides CAN slots for four nodes with a communication round length of 320 μs. The CAN slot of each node has a size of 64 bytes. In the simulation, the bandwidth of the network block has been set to 500 kbps.

The simulation results for the automotive message sets are depicted in Figure VII-C. The x-axis represents the message transmission latencies. The message identifiers are distinguished along the y-axis. These identifiers range from 80 to 2004 and denote the message priority. Larger CAN identifiers correspond to lower message priorities. The distance along the z-axis represents the number of messages with a given message priority and transmission latency. Figure VII-C shows that high priority messages make up for a large amount of the overall bandwidth. The third of messages with the highest priority makes up for 49% of exchanged messages. The simulation results also demonstrate the high average performance of CAN. 97% of all message transmissions possess transmission latencies below 1 ms. However, transmission latencies vary considerably. The logarithmic scaling of the z-axis in Figure VII-C emphasizes the rare cases, in which transmission latencies significantly differ from the average values.

The right hand side of Figure VII-C depicts the measured transmission latencies for the automotive

message sets in the implementation of the event service (i.e., before protocol emulation). The observed worst-case latency is 608 μs, i.e., significantly lower compared to the 3165 μs of the physical CAN network. The best-case latency is 147 μs. The average message transmission latency of the measurements is 323.51 μs.

The correct behavior of the protocol emulation has been demonstrated via the sequence numbers contained in the measurement records. Both the measurement framework and the simulation frameworks assign sequence numbers to received messages in order to capture the temporal message order. In test runs without protocol emulation, the message sequences have been different for the physical CAN network and the VCNs. With enabled protocol emulation, the sequence numbers in the measurement records of the VCN have shown the same message order as in the simulation of the physical CAN network.

However, there is a fundamental trade-off between improving the worst-case latencies in comparison to a physical CAN network and the establishment of the correct temporal message order. With enabled protocol emulation, the worst-case latency of the VCN is also 3.1 ms, since the CSMA/CA protocol is emulated. Although a CAN messages is received earlier via the event service, the forwarding to the front-end is delayed until the message wins in the simulated arbitration process. For this purpose, a designer needs to decide whether he prefers a superior temporal performance or enables protocol emulation in order to ensure the correct behavior of legacy software without redevelopment efforts.

D. Results – Integration of CAN-based Diagnostic Software Stack

The implementation of the CAN emulation has been used to integrate the diagnostic software stack of an automotive company into nodes of a TT system, including a transport layer and a diagnostic event handler as part of KWP2000 [34]. The diagnostic software stack builds on top of the API of the HIS/VectorCAN driver, thus exploiting the services of the front-end in the CAN emulation.

KWP2000 is used in maintenance mode in order to retrieve diagnostic information, such as breakdown log entries. Breakdown log entries are generated by the OBD system in a node computer, once an error is detected (e.g., by the Built-In Self Test (BIST) or by application-specific plausibility checks). At the service station the mechanic uses a diagnostic testing device (e.g., VAS tester) to retrieve breakdown log entries and determine the Diagnostics Trouble Codes (DTC).

In conjunction with a CAN/TTP gateway implemented on a TTP node [35], it is possible to diagnose the TTP network with a standard diagnostic computer (VAS Tester) as presently used in repair stations. For example, using the VAS tester the diagnostic information provided by C2 controller can be displayed (e.g., membership vector of the TTP nodes).

E. Delay of Event-Triggered Message Exchanges Induced by Mapping to the Time-Triggered Network

In an integrated system for TT and CAN-based applications, the duration of a communication round on the TT network is an application-specific parameter that contributes significantly to the end-to-end delay of messages exchanged via a VCN. As depicted in Figure 12, the application requests the transmission of a

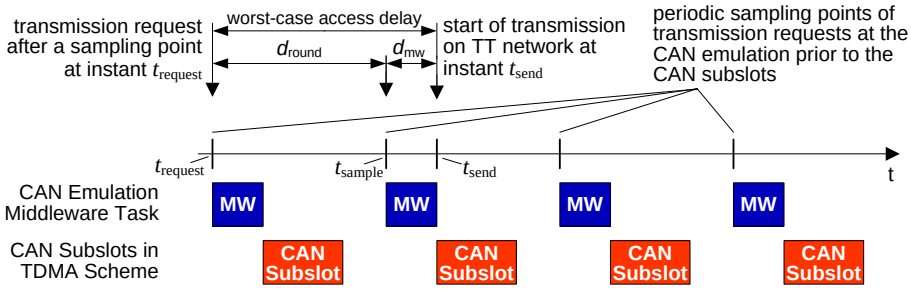

Fig. 12. Worst-Case Access Delay Induced by Mapping to the Time-Triggered Network

message at the request instant t_{request}. This action causes an update of the interface data structures provided by the CAN emulation middleware, which contain now a new message that needs to be transmitted via the VCN. After the request instant the access delay (see also Section V) follows before the transmission of the CAN message is started on the TT network at instant t_{send}. Since CAN-based applications are not synchronized to the TDMA scheme of the TT network, the access delay depends on the request instant relative to the start instant of the next CAN subslot. The CAN emulation middleware periodically samples the transmission requests prior to each CAN subslot, thus incurring a worst-case delay of a complete communication round in case a message transmission is requested immediately after a sampling point of the CAN emulation middleware. In this case, which is depicted in Figure 12, the sampling of the message transmission request (at instant t_{sample}) is delayed until the CAN emulation is activated again for the next CAN subslot. Therefore, the worst-case access delay comprises the sum of the communication round duration d_{round} and the worst-case execution time d_{mw} of the CAN emulation middleware. The access delay jitter is equal to the duration of a communication round, while the average access delay is $d_{\text{mw}} + d_{\text{round}}/2$, when assuming a uniform distribution of the request instants within a communication round.

Based on this analysis, the effects of different TT communication schedules can be quantified. In many practical systems, a communication round length of 320 μs as used in the prototype implementation can be difficult to attain due to larger numbers of nodes and application software requiring longer execution time slots (e.g., several ms). For example, assuming a communication round length of 5 ms, both the worst-case access delay and the access delay jitter would be extended by 4.68 ms in comparison to the 320 μs schedule. The average access delay would be increased by 2.34 ms due to an average delay of 2.5 ms before the next sampling point (instead of 160 μs in case of the 320 μs schedule).

VIII. Conclusion

With the introduction of TT communication protocols in safety-related and safety-critical computer systems of upcoming car series, the reuse of CAN-based legacy application in TT computer systems becomes of high economic relevance. The CAN emulation provides a basis for designers who intend to reuse CAN-based applications despite the migration to a TT platform. In addition to the ability to retain investments in existing software, such a migration permits the replacement of physical CAN networks

through event channels. As overlay networks on the TT communication service, the resulting *virtual CAN networks* enable significant cost and reliability benefits through the reduction of connectors and wiring. In order to minimize the redevelopment efforts for CAN-based legacy applications, it is important to provide to applications the same interfaces to the underlying communication system as in a physical CAN system. In particular, the correct behavior of many legacy applications depends on the temporal properties (bandwidth, latencies, message order) of the emulated CAN communication service. For this reason, virtual CAN networks as described in this paper not only support ET on-demand communication activities on top of a TT communication protocol, but also emulate the CSMA/CA media access control protocol of CAN in order to ensure the same temporal message order as a physical CAN network. By exploiting the higher native bitrate of the underlying TT network, virtual CAN networks can exceed physical CAN network with respect to bandwidth and latencies. The ability of virtual CAN networks to handle the communication needs of automotive applications has been demonstrated in a validation framework employing a real-world automotive communication matrix and a diagnostic protocol stack.

ACKNOWLEDGMENTS

This work has been supported in part by the European IST project DECOS (IST-511764) and the European IST project ARTIST2 (IST-004527).

REFERENCES

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

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

[3] J. Leohold and C. Schmidt, "Communication requirements of future driver assistance systems in automobiles," in *Proceedings of the IEEE International Workshop on Factory Communication Systems*, Sept. 2004, pp. 167–174.

[4] H. Heitzer, "Development of a fault-tolerant steer-by-wire steering system," *Auto Technology*, vol. 4, pp. 56–60, Apr. 2003.

[5] G. Leen and D. Heffernan, "Expanding automotive electronic systems," *Computer*, vol. 35, no. 1, pp. 88–93, Jan. 2002.

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

[7] K. Hoyme and K. Driscoll, "SAFEbus," *IEEE Aerospace and Electronic Systems Magazine*, vol. 8, pp. 34–39, Mar. 1993.

[8] *FlexRay Communications System Protocol Specification Version 2.1*, FlexRay Consortium. BMW AG, DaimlerChrysler AG, General Motors Corporation, Freescale GmbH, Philips GmbH, Robert Bosch GmbH, and Volkswagen AG., May 2005.

[9] "Analysis of the european automotive in-vehicle network architecture markets," Frost & Sullivan, Tech. Rep., Oct. 2004.

[10] Vector Informatik GmbH, "CANalyzer and DENalyzer 5.2 – the tools for comprehensive network analysis," Stuttgart, Tech. Rep., 2005.

[11] J. Swingler and J. McBride, "The degradation of road tested automotive connectors," in *Proceedings of the 45th IEEE Holm Conference on Electrical Contacts*. Pittsburgh, PA, USA: Dept. of Mech. Eng., Southampton Univ., Oct. 1999, pp. 146–152.

[12] OSEK/VDX, "Operating system – version 2.2.3," Tech. Rep., Feb. 2005.

[13] P. Pedreiras and L. Almeida, "Combining event-triggered and time-triggered traffic in FTT-CAN: Analysis of the asynchronous messaging system," in *Proceedings of 3rd IEEE International Workshop on Factory Communication Systems*, Sept. 2000.

[14] T. Führer, B. Müller, W. Dieterle, F. Hartwich, and R. Hugel, "Time-triggered CAN - TTCAN: Time-triggered communication on can," in *Proc. of 6th International CAN Conference (ICC6)*, Torino, Italy, 2000.

[15] J. Rushby, "A comparison of bus architectures for safety-critical embedded systems," Computer Science Laboratory, SRI International, Tech. Rep., Sept. 2001.

[16] P. L. S. Katz and J. Rushby, "Low-overhead time-triggered group membership," in *Proc. of 11th International Workshop on Distributed Algorithms, Lecture Notes in Computer Science*, vol. 1320. Springer-Verlag, 1997, pp. 155–169.

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

[18] Freescale Semiconductor, "MFR4200 datasheet FlexRay communication controllers," Tech. Rep., Aug. 2005.

[19] "IEEE Std. 1588 - 2002 IEEE Standard for a precision clock synchronization protocol for networked measurement and control systems," IEEE Instrumentation and Measurement Society, Tech. Rep., Nov. 2002, iSBN: 0-7381-3369-8.

[20] F. Cristian, "Understanding fault-tolerant distributed systems," *Communications of the ACM*, vol. 34, no. 2, pp. 56–78, 1991.

[21] H. Kopetz, G. Grünsteidl, and J. Reisinger, "Fault-tolerant membership service in a synchronous distributed real-time system," *Dependable Computing for Critical Applications (A. Avizienis and J. C. Laprie, Eds.), pp.411-29, Springer-Verlag, Wien, Austria*, 1991.

[22] H. Kopetz and R. Nossal, "Temporal firewalls in large distributed real-time systems," *Proceedings of the 6th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS '97)*, 1997.

[23] A. Albert and W. Gerth, "Evaluation and comparison of the real-time performance of CAN and TTCAN," in *Proc. of 9th International CAN Conference*, Munich, 2003.

[24] V. Hadzilacos and S. Toueg, "A modular approach to fault-tolerant broadcasts and related problems," Department of Computer Science, Cornell University, Ithaca NY, USA, Tech. Rep. TR94-1425, May 1994.

[25] P. Bellini, R. Mattolini, and P. Nesi, "Temporal logics for real-time system specification," *ACM Computing Surveys (CSUR)*, vol. 32, no. 1, pp. 12–42, 2000.

[26] B. Hardung, T. Kölzow, and A. Krüger, "Reuse of software in distributed embedded automotive systems," in *EMSOFT '04: Proceedings of the fourth ACM international conference on Embedded software*. New York, NY, USA: ACM Press, 2004, pp. 203–210.

[27] Volkswagen AG, "HIS/VectorCAN driver specification 1.0," Berliner Ring 2, 38440 Wolfsburg, Tech. Rep., Aug. 2003.

[28] "CANopen application layer and communication profile v4.0.2," CiA DS 301, Tech. Rep., Mar. 2005.

[29] D. Noonen, S. Siegel, and P. Maloney, "DeviceNet application protocol," in *Proceedings of the 1st International CAN Conference*, Mainz, Germany, 1994.

[30] *TTP Monitoring Node – A TTP Development Board for the Time-Triggered Architecture*, TTTech Computertechnik AG, Schönbrunner Strasse 7, A-1040 Vienna, Austria, Mar. 2002.

[31] D. Beal, E. Bianchi, L. Dozio, S. Hughes, P. Mantegazza, and S. Papacharalambous, "RTAI: Real-Time Application Interface," *Linux Journal*, April 2000.

[32] "RTAI Programming Guide, Version 1.0," Dipartimento di Ingegneria Aerospaziale Politecnico di Milano (DIAPM), Italy, September 2000, available at http://www.rtai.org.

[33] D. Henriksson, A. Cervin, and K. Arzen, "Truetime: Simulation of control loops under shared computer resources," in *Proceedings of the 15th IFAC World Congress on Automatic Control*. Barcelona, Spain: Department of Automatic Control, Lund Institute of Technology, July 2002.

[34] *Keyword Protocol 2000, ISO 14230*, ISO, www.iso.ch, International Organization for Standardization, 1999.

[35] *TTP Development Cluster - The Complete TTP System*, TTTech Computertechnik AG, Schönbrunner Strasse 7, A-1040 Vienna, Austria, 2004.