Mixed Criticality on Controller Area Network

A. Burns
Department of Computer Science,
University of York, UK.
Email: alan.burns@york.ac.uk

R.I. Davis
Department of Computer Science,
University of York, UK.
Email: rob.davis@york.ac.uk

***Abstract*—An increasingly important trend in the design of real-time and embedded systems is the integration of components with different levels of criticality onto a common hardware platform. Where the platform incorporates a communication media it is necessary for that media to be able to safely and efficiently transfer messages of different criticality levels. In this paper we consider the Controller Area Network (CAN), and define mixed criticality protocols that could form the basis of a Trusted Network Component for CAN. Sufficient response-time analysis is derived for these protocols and an optimal priority assignment scheme is provided. Evaluations illustrate the benefits of the schemes.**

I. Introduction

An increasingly important trend in the design of real-time and embedded systems is the integration of components with different levels of criticality onto a common hardware platform. A mixed criticality system (MCS) is one that has two or more distinct levels (e.g. safety critical, mission critical and non-critical). Perhaps up to five levels may be identified (see, as examples, the IEC 61508, DO-178B, DO-254 and ISO 26262 standards).

Most of the complex embedded systems found in the automotive and avionic industries are evolving into mixed criticality systems in order to meet stringent non-functional requirements relating to cost, space, weight, heat generation and power consumption. Indeed the software standards in the European automotive industry (AUTOSAR) and in the avionics domain (ARINC) address mixed criticality issues (in the sense that they recognise that MCSs must be supported on their platforms).

If MCSs are to be hosted on platforms such as AUTOSAR then the communication media must be capable of supporting mixed criticality messages. Indeed if applications are partitioned so that any one processor only has software of one criticality level then the main focus of design must be on what links the different processors (and criticality levels).

Within the context of automotive applications and platforms, CAN is the predominant network protocol. It is therefore necessary to develop means by which CAN can be used safely for transmitting messages of different criticality. This challenge is itself composed of two conflicting requirements:

1) how to partition the use of the network to enhance safety;
2) how to share the capacity of the network to reduce cost.

In this paper we develop analysis that can be used to facilitate effective sharing of the available bandwidth of CAN.

CAN is a serial communications bus designed to provide simple, efficient and reliable communications for in-vehicle networks [15]. An indication of the scale of adoption of CAN by the automotive industry can be gained from the sales of microcontrollers with on-chip CAN peripherals. Over 1 billion such devices had been deployed in automotive applications by 2006 [11].

A number of papers (e.g. [13], [14]) have already provided detailed descriptions of the CAN protocol and the analysis required to determine if message streams will meet their deadlines. Here, we assume that the reader is familiar with the CAN protocol, its basic analysis, and error handling as described in [14].

The paper is organised as follows. In the next section a system model is defined that allows MCS to be developed on CAN. Section III reviews material on mixed criticality and introduces the message stream model. Section IV then introduces basic CAN analysis. The main protocol derived in this paper is explained, with its analysis in Section V. A more basic form is then derived in Section VI. Evaluations are provided in Section VII followed by conclusions.

II. System Model

If a single CAN bus is to be used to pass messages of differing criticality then some level of partitioning is required. One simple approach would be to assign priorities according to criticality. But this has a detrimental impact on schedulability [3]. Also there would still be the need to provide protection from a low criticality message inappropriately being assigned a high priority (due to either an error or a security breach).

A review of the methods by which a CAN controller can be reliably constructed was given by Broster and Burns in 2003 [7]. They showed how the so called 'babbling idiot' problem could be addressed – this is when a node sends out more messages than was assumed in the static analysis of the hosted applications. Two basic approaches were considered:

1) the use of a Trusted Network Component (TNC) as an interface to the network, or
2) the use of a Network Guardian (NG).

The trusted network component (TNC) is designed and built according to the constraints imposed by the highest level of criticality of any application running (or liable to run) on the platform. It can be trusted to only allow onto the network messages that conform to the constraints of the static analysis.

A network guardian (NG) is used when the network interface hardware cannot be trusted. The NG monitors the messages on the network and identifies inappropriate levels of traffic. It will then disconnect the offending node and prevent any further transmissions from that source. Of course the NG itself must be trusted, but it is a much simpler component than a TNC; it does not itself transmit any messages.

Although a NG can prevent excessive misuse of the network, it only identifies a problem after some level of inappropriate message flow has occurred and been identified. This means that the CAN schedulability analysis has to be updated to take into account some (bounded) level of 'babbling' [7].

In this work we build upon the use of TNCs. In the automotive environment it is unlikely that the extra costs of NGs would be acceptable. Moreover, the high volumes of production have lead to highly reliable basic controllers to which a trusted software layer can be added. Nevertheless, if a guardian-based approach is advocated the analysis developed in this paper could be augmented by that for NGs [7] in a relatively straightforward way.

The approach to criticality developed in the rest of this paper defines the operation of the complete system to be in one of an ordered set of *criticality modes*. The system starts in the lowest mode, in which all the standard functions of the application are supported. The system will normally stay in this mode. However, if required (see below for details) the mode may change to a higher level, reaching where necessary the highest level, in which only the highest criticality functions of the system need to be guaranteed. It will stay at this level until the system is re-initialised. Movements down the criticality levels are not part of the proposed protocol.

The TNC is aware of the current mode and will only allow messages to be transmitted that are compatible with that mode. The need for a change of mode will be identified by a TNC when:

- Some functional component on the TNC's node indicates that a mode change is required.
- Messages arrive for transmission too frequently for the current mode.
- Errors occur too frequently for the current mode.

In response to any of these conditions the TNC will transmit a high priority message to all other nodes informing them of a criticality change. Each TNC will have a distinct message id for this purpose; but all these messages will have priorities higher than any application message. In some circumstances, a message that is only allowed in a particular mode will itself be the indicator to the rest of the system of the need to change mode. Such a message is referred to as a *triggering message*.

These properties of the TNC are integrated with the explicit support of a mixed criticality protocol which we call MixedCAN, see Section V. Having introduced MixedCAN a more basic form of the protocol (BMC) is defined in Section VI.

A. Fault Model

For high criticality applications it is usually required that they are capable of surviving a predefined levels of faults. For CAN this means that messages should be delivered by their deadlines even when some re-transmissions are necessary. Of course no deadline can be satisfied if there are an unbounded number of faults on the network. The *fault model* defines the level of faults that must be tolerated [17]. A certification authority may then require evidence that during deployment the probability of faults arriving outside the fault model is below a specified level [8], [19].

There are various ways of defining a fault model. A simple one for CAN is to specify the number of faults that must be tolerated during the queuing and communication of a message. Let F represent this value which is then used in the schedulability analysis to compute the worst-case response time of each message. If D_{max} is the longest message deadline then the probability of failure is bounded by the probability of more than F faults occurring in an interval of duration less than D_{max}. If the fault arrival pattern can be modeled as a Poisson distribution then this probability can be computed [8]. For other patterns it is more problematic. However, in this paper we are only concerned with the impact the fault model has on schedulability. For this we only need the F value (or, more specifically, as will be explained shortly, an F value per criticality level). More complex fault models, for example defining F as the number of faults in unit time, are easily incorporated into schedulability analysis.

III. Mixed Criticality Model

The first paper on the verification of a Mixed Criticality Systems used an extension of standard fixed priority (FP) real-time scheduling theory, and was published by Vestal in 2007 [21]. It employed a somewhat restrictive work-flow model, focused on single processor task scheduling and made use of Response Time Analysis [2]. It showed that neither rate monotonic nor deadline monotonic priority assignment was optimal for MCS; however Audsley's optimal priority assignment algorithm [1] was found to be applicable.

This paper was followed by two publications in 2008 by Baruah and Vestal [5], and Huber et al [16]. The first of these papers generalises Vestal's model by using a sporadic task model and by assessing fixed job-priority scheduling and dynamic priority scheduling. It contains the important result that EDF (earliest deadline first) does not dominate FP when criticality levels are introduced, and that there are feasible systems that cannot be scheduled by EDF. The latter paper addresses multi-processor issues and virtualisation (though did not use that term). It focused on AUTOSAR and resource management (encapsulation and monitoring) with time-triggered applications and a trusted network layer. Since these early papers a number of publications have extended the applicability of these results, but they have tended to concentrate only on task scheduling. Here we focus on messages being scheduled for non-preemptive transmission on CAN.

Each message stream, τ_i, is defined by its period (minimum inter-arrival interval), blocking, release jitter, relative deadline, maximum transmission time and criticality level: (T_i, B_i, J_i, D_i, C_i, L_i). These parameters are however not independent. In the standard mixed criticality approach [21] only the tasks' computation times are a function of criticality. But in general all these parameter's could be dependent on criticality [9]. For message streams it is the stream's period that has this dependency. The higher the criticality level, the more conservative the assumptions made about the rate of arrival of high criticality messages. For example, event-handling tasks may give rise to inter-node messages; the more events that must be handled, the more messages will be generated. Assumptions about event arrivals directly impacts on message frequencies.

With applications such as those in the automotive domain it will also be the case that some message streams will only occur in the higher criticality modes. For example, in an active safety system, the identification of a potential collision could result in messages being sent to apply brakes, tighten seat belts, close windows etc. Other standard but critical messages will also need to be delivered, but other none critical ones could be dropped.

In a mixed criticality system more information is needed in order to undertake schedulability analysis. Message streams can depend on other messages with higher or lower levels of criticality. In general, a message is now defined by: ($\vec{T}$, B_i, $\vec{J}_i$, D, $\vec{C}$, L), where $\vec{X}$ is a vector of values – one per criticality level, with the constraints:

$$L1 > L2 \Rightarrow C(L1) \geq C(L2)$$
$$L1 > L2 \Rightarrow T(L1) \leq T(L2)$$

for any two criticality levels $L1$ and $L2$.

In general, although message streams may have different release jitter values, when criticality is considered, there will be no ordering property over the values in $\vec{J}_i$. For analysis purposes the worst-case jitter must be used, and hence a single J_i parameter is employed; it being the maximum of the values in $\vec{J}_i$.

Although we allow each message's period to be dependent on criticality, so that the model can cope with different rates of computation and hence communication in the criticality modes, we have the same relative deadline for each message regardless of criticality. There is therefore only a single D term in the description of a message stream.

The final element of the mixed criticality model is the fault model. It is reasonable to assume that the number of faults that must be tolerated will be higher for higher levels of criticality. Hence

$$L1 > L2 \Rightarrow F(L1) \geq F(L2)$$

where $F(Lx)$ is the number of faults that must be tolerated at criticality level Lx (see Section II-A).

A. Restricted Model

For ease of presentation, in this paper we will restrict our attention to *dual-criticality* systems: systems in which there are only two criticality levels: HI (high) and LO (low), with $HI > LO$. We also consider constrained deadlines. So for low criticality messages: $D \leq T(LO)$ and for high criticality messages: $D \leq T(HI) \leq T(LO)$.

$\mathcal{M}$	Crit	T(HI)	T(LO)	D	C
τ_1	HI	∞	-	5	2
τ_2	HI	12	24	12	2
τ_3	LO	-	11	11	2
τ_4	LO	-	6	6	1
τ_5	HI	18	36	18	3

TABLE I
EXAMPLE MESSAGE SET

In this study the message transmission times (the C values) for each message are not criticality dependent, apart from the specific case of a high criticality message that does not occur in the LO criticality mode. In this case $C(LO) = 0$; otherwise $C(HI) = C(LO)$ for all messages. In general we will use the single term C for all criticality levels unless we need to explicitly identify the $C(LO) = 0$ property. We assume that in both criticality levels all messages are subject to the maximum amount of bit stuffing.

B. Example

We use an abstract example (parameters not representative of CAN messages) to illustrate our analysis (see Table I). Message τ_1 only exists in the HI mode, indeed it is the message that causes the mode to move from low criticality to high criticality. Moreover, τ_1 is transmitted only once in the HI mode; hence its period is given as ∞. Two further messages are HI criticality; both have their periods reduced by half in the higher criticality mode of operation. There are in addition two LO criticality messages that are only transmitted in the LO criticality mode; hence their periods in the other mode are undefined. Jitter is assumed to be zero and faults are not considered in this example.

If the distinctive criticalities are ignored (in the sense that all messages must meet their more stringent requirements at all times) then the message set is unschedulable (see the following analysis).

IV. BASIC SCHEDULING ANALYSIS FOR CAN

In this paper, we make use of the simple sufficient but not necessary schedulability test given by Davis et al. [13]. We note that this analysis is exact for many commercial CAN systems that have 8 data byte (soft) real-time messages present at lower priorities. The interested reader is referred to [13] for details of other schedulability tests for CAN and the conditions under which the various tests provide sufficient or exact analysis.

The worst-case response time of a message can be viewed as being made up of three elements:

1) The queuing jitter J_i, corresponding to the longest time between the initiating event and the message being queued, ready for transmission.
2) The queuing delay R_i^s, corresponding to the longest time that the message can remain in the CAN controller slot or device driver queue, before commencing successful

transmission, i.e. it is the maximum time before the start of transmission.

3) The transmission time C_i, corresponding to the longest time that the message can take to be transmitted.

The queuing delay, R_i^s, can be determined by solving the following response-time equation [13]:

$$R_i^s \;=\; \max\left(B_i, C_i\right) \;+\; \sum_{j \in \mathbf{hp}(i)} \left\lceil \frac{R_i^s + J_j + \delta}{T_j} \right\rceil C_j. \quad (1)$$

Here the δ term is used to ensure that the message of interest has actually started to be transmitted; for a real example it corresponds to the time it takes to transmit a single bit; $\mathbf{hp}(i)$ is the set of messages with a higher priority than message τ_i and $\max\left(B_i, C_i\right)$ corresponds to the longest possible time that τ_i can be blocked by either a lower priority message or via push through blocking due to a previous invocation from the same message stream [13]. For ease of presentation we shall use the symbol $\hat{B}_i = \max\left(B_i, C_i\right)$ in subsequent equations (and where appropriate $\hat{B}_i(LO)$ and $\hat{B}_i(HI)$). Equation (1), and all subsequent response time equations, are solved by forming a recurrence relation. The complete worst-case response-time for the message is given by:

$$R_i \;=\; R_i^s \;+\; J_i \;+\; C_i. \quad (2)$$

In the example from Table I it is easy to confirm that no message is schedulable at the lowest priority level if issues of criticality are ignored. For example, τ_5 at the lowest level has a blocking value of 3 and an initial interference of 2+2+2+1. Equation (1) iterates to give a solution of 16. Equation (3) then gives $R_5 = 19$ which is beyond the message's deadline.

To incorporate retransmission due to faults requires an extra component to be added to equation (2) [8]. Let Err_{max} be the maximum length error, then to tolerate F faults:

$$\begin{aligned} R_i^s \;&=\; \hat{B}_i \;+\; \sum_{j \in \mathbf{hp}(i)} \left\lceil \frac{R_i^s + J_j + \delta}{T_j} \right\rceil C_j \\ &+\; F * \{Err_{max} + \max_{k \in \mathbf{hep}(i)} C_k\}, \end{aligned} \quad (3)$$

where $\mathbf{hep}(i)$ is the set of messages with a higher or equal priority than message τ_i.

As F will, in general, be a function of criticality, we define $\hat{F}_i(HI) = F(HI) * \{Err_{max} + \max_{k \in \mathbf{hep}(i)} C_k\}$; i.e $F(HI)$ faults, each causing extra transmission. Similarly, $\hat{F}_i(LO) = F(LO) * \{Err_{max} + \max_{k \in \mathbf{hep}(i)} C_k\}$.

V. A Mixed Criticality Protocol for CAN

In this section we develop a protocol for safely and effectively using CAN to communicate messages of different criticality. This protocol, call MixedCAN, is implemented in the Trusted Network Component (TNC). First we define the necessary properties of the TNC (to monitor and control access to CAN), then analysis is developed for the protocol. In Section VI a simpler version of MixedCAN is defined.

Recall that the system has two criticality modes, HI and LO (abbreviated to HI-crit and LO-crit). Each node has a local view of the current criticality of the system. The system (and all local views) are initialised to LO. Run-time behaviour may cause a node to change its local view to HI; this is then communicated to all other nodes. Eventually all nodes will switch their local view to HI. The system will then stay in that state.

A. Monitoring capabilities of the TNC

One key property of the TNC is that it will monitor message send requests and determine if they are occurring too frequently for the mode of operation. Here we formalise this behaviour of the TNC. We do this by extending the *sporadic invariant* defined by Broster and Burns [7].

In the dual-criticality model, to monitor the arrival pattern of message stream τ, three parameters are required: J, $T(HI)$ and $T(LO)$ (as we are only considering one message stream at this time we omit the subscript). Consider the TNC for the node that generates the messages of τ; and let $E_1, E_2, \ldots E_k$ be the times at which the TNC receives requests to transmit messages from the τ stream. For each of these messages there is a guide time, $G_1, G_2, \ldots G_k$ that bounds the arrival times; i.e. for correct behaviour:

$$\forall k : E_k \geq G_k,$$

i.e. a message must not arrive too early.

There are two situations that determine the values for the sporadic invariant (i.e. the constraint on the arrival of messages):

1) If there has been a reasonable gap since the last message then the next two consecutive messages can be as close as 'period - jitter', T-J, (but no closer).
2) If messages are coming at the maximum rate then two consecutive messages can be no closer than 'period', T.

So if the TNC receives a correctly timed request at time E_{k-1} (i.e. $E_{k-1} \geq G_{k-1}$) then the next request must arrive no sooner than time G_k[7], where

$$G_k = \max\{G_{k-1} + T, E_{k-1} + T - J\}.$$

In the LO criticality mode, this implies:

$$G_k(LO) = \max\{G_{k-1}(LO), E_{k-1} - J\} + T(LO). \quad (4)$$

At run-time if $E_k \geq G_k(LO)$ then this message is valid in the LO-crit mode and the system can continue to execute in that mode. If however $E_k < G_k(LO)$ then the message is not valid in LO-crit mode and the system criticality level must be changed to HI. It will remain at this level for the rest of the system's execution.

In addition to computing $G_k(LO)$ a similar bound for HI criticality behaviour can be defined:

$$G_k(HI) = \max\{G_{k-1}(HI), E_{k-1} - J\} + T(HI) \quad (5)$$

Note as $T(HI) \leq T(LO)$ then $G_k(HI) \leq G_k(LO)$.

If the HI criticality invariant is broken (i.e. $E_k < G_k(HI)$) then the system designer must choose between two valid (but mutually exclusive strategies):

1) Allow the message to be sent as it is a high-criticality message, and although deadlines may be missed this is probably the most robust behaviour available, or
2) Assume a major problem with the node and not allow any further messages to be transmitted (of any criticality); in effect the node is assumed to be babbling and hence must not be allowed to interfere with other messages on the network.

If there is adequate redundancy in the system (i.e. all crucial functionality is replicated) then strategy 2 is probably best. If not, then the TNC should allow the message to be transmitted. This is the scheme used below, and hence the $G_k(HI)$ values are not computed at run-time and the TNC does not need to know the $T(HI)$ parameters.

A refinement of the protocol, which is incorporated into the functionality defined below, has to differentiate between low-criticality messages arriving early (which are just ignored, with perhaps an exception being raised) and high-criticality messages arriving early which result in a criticality change.

B. Functional Behaviour of MixedCAN

The trusted software within the TNC consists of two routines. One (`output`) is called by the application code within a node to pass a message (`M`) to the network. The second routine (`input`) is called by the CAN controller hardware, as an interrupt handler, to move a message from the network into the buffer space of the application. To prevent concurrent calls to `output`, and interleaved calls to `output` and `input`, we require both routines to be executed at the same interrupt level (and hence all calls are serialised). We note however that there are other means of achieving this serialisation (e.g. by the use of a dedicated thread).

To police the passing traffic, the TNC must be initialised with information about the messages it will handle. We assume that this is (safely) represented by the following data structures that are defined over all the valid message streams.

`T[M]` $\equiv$ period of message `M` in `LO` mode.
`J[M]` $\equiv$ maximum jitter of message `M`.
`S[M]` $\equiv$ maximum size of message `M`.

To define the functional behaviour of the `output` and `input` routines some basic procedures and functions are required:

`Send(M)` $\equiv$ Commits `M` to the controller for output on CAN.
`Flush(M)` $\equiv$ Removes any message with id `M` from CAN output buffers.
`Receive(M)` $\equiv$ Copies incoming message into memory for application to read.
`Clock` $\equiv$ returns the current time.
`Valid(M)` $\equiv$ returns true if `M` is valid (described below).
`Crit(M)` $\equiv$ returns the criticality of message `M` (`HI` or `LO`).
`Trigger(M)` $\equiv$ returns true if `M` should cause a criticality change from `LO` to `HI`; if `M` is a triggering message it must also be of `HI` criticality and have a priority higher than any `LO` criticality message.
`Errors_High` $\equiv$ returns true if current error count is above level for `LO` mode.
`flushALL` $\equiv$ repeatedly calls `Flush` to remove all `LO`-crit messages from output buffers.

Note `Send` and `Flush` are instructions to the CAN controller, the application software on the node must be structured or configured so that only the `output` and `input` routines can call these procedures directly.

Some state variables are also required:

`G[M]` $\equiv$ next arrival time bound for message `M` in `LO` mode, initialised to some appropriate negative value ($-\infty$).
`Crit_Level` $\equiv$ local notion of current criticality level, initialised to `LO`.
`Go_HI` $\equiv$ predefined message used by TNC to communicate to the rest of the system that a mode change from `LO` to `HI` is required; there will be a distinct `Go_HI` message for each node. The priorities of `Go_HI` messages will be higher than any LO criticality message. Note `Trigger(Go_HI)` is true for all such messages. A final point for the protocol: all triggering messages must be delivered to all nodes.

With these definitions it is now possible to provide pseudo code for the key routines[1]. We note that error counts are checked as messages are input. First `output`:

```
output(M) is  -- called by application code
  t := clock
  if not Valid(M) then return <invalid> end if
  if Crit_Level = LO then
    if Trigger(M) then
      send(M)
      Crit_Level := HI
      flushALL
    else if t < G[M] then -- too early for LO mode
      if Crit(M) = HI then
        send(Go_HI)
        send(M)
        Crit_Level := HI
        flushALL
      else
        return <invalid, too early>
      end if
    else
      G[M] := max(G[M], t - J[M]) + T[M]
      send(M)
    end if
  else  -- in HI mode
    if Crit(M) = HI then send(M) end if
  end if
  return <success>
```

First each message is checked to make sure it is valid for that node. This means that its size is not greater than `S[M]` and that `M` is in the set of allowable output messages.

When the system is in HI mode, only HI-crit messages are passed on to the controller. A move from LO mode to HI will occur if a triggering message is received from the node (including the `Go_HI` message), or when a request is made too early for the LO mode by a high-crit message. Whilst in the LO mode whenever a (valid) message arrives the earliest

[1]Presentation takes precedence over efficiency in this pseudo code; an implementation could clearly improve on this code.

time for the next message is calculated (and held in G[M]).

During the system's execution at most one change from LO to HI can occur. If it does occur then flushALL is used to instruct the controller to remove from its output buffers all LO-crit messages not yet started transmission.

The code for a simple input routine is:

```
input(M) is -- called by interrupt handler
  if Crit_Level = LO then
    if Errors_High then
      Crit_Level := HI
      flushALL
      send(Go_HI)
    else if Trigger(M) then
      Crit_Level := HI
      flushALL
      Flush(Go_HI)
    end if
  end if
  if (Crit_Level = LO) or
     (Crit_Level = HI and Crit(M) = HI) then
    Receive(M)
  end if
```

The first possibility Crit_Level = LO and Errors_High implies that this node could be the first to identify the conditions for a mode change. It therefore sends the Go_HI message and flushes its own buffers. If however the received message is a triggering message then the mode change has already been initiated by another node, and hence all local messages, including the mode change Go_HI message if it has not yet been sent, are flushed.

When combining this protocol with the typical behaviour of commercial CAN controllers then it is clear that between receiving a Go_HI or triggering message and flushing the output buffers of LO-crit messages a single unflushed high priority message could be transmitted. This could be a LO-crit message that has started to be communicated before it could be flushed, or it could be a second Go_HI message. Either way the analysis of the protocol must take this possibility into account.

In addition, whenever the 'error high' check recognises that a criticality change is required, a further low criticality, but high priority message might be in transmission before the Go_HI message can be queued. Again the analysis must take this into account.

A polling input routine. The above protocol assumes that an incoming message generates an interrupt if it has a client on the node in question. This is a common behaviour for a CAN controller, but it is possible to configure the controller to just place the incoming message in a buffer. Here there is no interrupt and the client software polls to check for incoming messages. For the TNC this behaviour would introduce a latency between a system mode change and the flushing of low criticality messages. The latency would be bounded, but its impact would need to be factored into the analysis.

$\mathcal{M}$	Crit	T(HI)	T(LO)	D	C	Pri	$R^s(LO)$	$R(LO)$
τ_1	HI	∞	-	5	2	1	-	-
τ_2	HI	12	24	12	2	4	7	9
τ_3	LO	-	11	11	2	3	4	6
τ_4	LO	-	6	6	1	2	3	4
τ_5	HI	18	36	18	3	5	9	12

TABLE II
EXAMPLE MESSAGE SET - QUEUING TIMES

C. Analysis of MixedCAN

The analysis derived in this section is an adaptation of that presented in [4] and has a number of similarities to that applicable to systems subject to mode changes [18]. Fortunately the model here is simpler than the one needed for general mode changes.

The form that the analysis takes has two phases:

1) Verifying the schedulability of the LO-crit mode,
2) Verifying the schedulability of the criticality change from LO to HI.

We shall show that in most circumstances a system that is schedulable during its criticality change will also be schedulable in the stable HI-crit mode. However, the reverse is not true, a system that is schedulable in the LO and HI modes may not be schedulable during the transition [20]. For the LO criticality mode, equations (3) and (2) can be adapted:

$$R_i^s(LO) = \hat{B}_i(LO) \; + \; \hat{F}_i(LO) + \sum_{j \in \mathbf{hp}(i)} \left\lceil \frac{R_i^s(LO) + J_j + \delta}{T_j(LO)} \right\rceil C_j(LO) \qquad (6)$$

$$R_i(LO) \; = \; R_i^s(LO) \; + \; J_i \; + \; C_i \qquad (7)$$

For illustration, consider the example given earlier (in Table I). Message τ_1 is the triggering message which is used to communicate the mode change (there is no Go_HI message). The LO-crit response times can be computed easily using this analysis. Initially we will assume deadline monotonic priority assignment – see Table II for the results of applying equations (6) and (7). In this example release jitter (though not δ) is assumed to be zero, there is no fault tolerance (i.e. $\hat{F}_i(LO) = 0$) and the $\hat{B}_i$ term is equal to 3 for all messages. We note that the LO criticality level is schedulable.

In [4] we showed that an adaptive model when task computation times can change is not amenable to exact analysis. This argument equally applies to the model used here where message periods can change; we therefore restrict ourselves to only sufficient analysis. Specifically we conservatively assume that, given a critical instant at time 0:

- All HI-crit messages are assigned their HI-crit parameters from time 0.
- All LO-crit messages are assigned their LO-crit parameters from time 0, with no messages being transmitted after the criticality change.
- The maximum sized message (from those defined by the application) is used to communicate the mode change.

- The maximum sized LO-crit message (from those defined by the application) is communicated after the $F(LO)+1$ fault occurs but before the `Go_HI` is queued (only occurs if $F(HI) > F(LO)$).
- The maximum sized LO-crit message (from those defined by the application) is communicated after the system has changed mode.

This is represented by the following response-time equation (which is straightforward extension of equation(6)):

$$R_i^s(HI) = C_i^F \ + \ C_i^{Mode} \ + \ \hat{B}_i(LO) + \sum_{\tau_j \in \mathbf{hpH}(i)} \left\lceil \frac{R_i^s(HI) + J_j + \delta}{T_j(HI)} \right\rceil C_j + \sum_{\tau_k \in \mathbf{hpL}(i)} \left\lceil \frac{R_i^s(LO) + J_i}{T_k(LO)} \right\rceil C_k + \hat{F}_i(HI), \quad (8)$$

where $\mathbf{hpH(i)}$ is the set of HI-crit messages with priority higher than that of message τ_i and $\mathbf{hpL(i)}$ denotes the LO-crit messages with priority higher than that of message τ_i. Note that the triggering messages are contained within $\mathbf{hpH(i)}$.

The δ term is not needed in the last term as $R_i^s(LO) + J_i$ could not be a solution of equation (6) if $R_i^s(LO) + J_i + \delta$ would have induced further interference.

The term C_i^F is used to capture the cost of the extra message that could be transmitted before `Go_HI` is queued (if $F(HI) > F(LO)$):

$$C_i^F = \max(C_k), \quad (9)$$

where C_k is the longest message in $\mathbf{hpL}(i)$.

Similarly the term C_i^{Mode} is used to capture the extra cost of the mode change itself. In the previous section it was noted that this consists of the `Go_HI` message and one extra (potentially low criticality) message that may be transmitted after the triggering message or first `Go_HI` message. Note that if there are only triggering messages (i.e no explicit `Go_HI` message) then $C_{Go\_HI} = 0$ is the following. That is:

$$C_i^{Mode} = C_{Go\_HI} \ + \ \max(C_{Go\_HI}, C_k), \quad (10)$$

Equation (8) 'caps' the interference from LO-crit messages as the response-time during the change, $R_i^s(HI)$, must be greater than $R_i^s(LO)$. Moreover the change must occur before $R_i^s(LO)$ otherwise the message would have started transmission before the mode change.

The final response-time during the transition is given by

$$R_i(HI) \ = \ R_i^s(HI) \ + \ J_i \ + \ C_i. \quad (11)$$

Note that if equations (8) and (11) determine that message τ_i is schedulable during the transition then it will also be schedulable in the steady-state HI-crit mode unless there is a much larger blocking term ($B_i(HI)$) in the HI-crit mode (because of a long low priority message that is only transmitted in the HI-crit mode) and this term is larger than the impact of the LO-crit component of equation (8). This is a highly unlikely circumstance, but would need to be checked.

If this analysis is applied to τ_5 in the message set of Table II then $R_5^s(LO) = 9$ and

$$\sum_{\tau_k \in \mathbf{hpL}(5)} \left\lceil \frac{9}{T_k(LO)} \right\rceil C_k = 1 + 1 + 2 = 4.$$

Also

$$C_5^{Mode} = 0 \ + \ \max(0, 2) = 2,$$

Note all C^F terms are zero in this example and there is no explicit triggering message so $C_{Go\_HI} = 0$. So $R_5^s(HI)$ is initially $2 + 3 + 4 + 4 = 13$, but τ_2 interferes a second time, so $R_5^s(HI) = 15$ and $R_5(HI) = 15 + 3 = 18$. Which is (just) schedulable. However, if this analysis is applied to τ_2 then $R_2^s(LO) = 7$ and

$$\sum_{\tau_k \in \mathbf{hpL}(2)} \left\lceil \frac{7}{T_k(LO)} \right\rceil C_k = 4.$$

Also

$$C_2^{Mode} = 0 \ + \ \max(0, 2) = 2.$$

So $R_2^s(HI) = 2+3+2+4 = 11$ and $R_2(HI) = 11+2 = 13$ which is greater than the message's deadline. However, this does not mean the message set is unschedulable for all priority orderings. We next consider optimal priority assignment.

D. Optimal Priority Assignment

An important observation contained in Vestal's paper [21] is that Deadline Monotonic Priority ordering is not optimal for mixed criticality systems, but that Audsley's optimal priority assignment algorithm is. To apply Audsley's algorithm to the MixedCAN model requires it to satisfy a set of pre-requisites [10], [12].

- The schedulability of a message may be a function of the set of higher priority messages, but not their specific priorities.
- The schedulability of a message may depend on the set of lower priority messages, but not on their specific priorities.
- A schedulable message that has its priority raised cannot become unschedulable, and conversely an unschedulable message that has its priority lowered cannot become schedulable.

An examination of the scheduling equations (6) and (8) shows that these properties are indeed fulfilled. We shall use the algorithm to assign priorities to our running example, that we have already shown is not schedulable by Deadline Monotonic priority assignment.

As indicated above, τ_5 is schedulable at the lowest level. For the next priority level (4) τ_3 is actually schedulable (see Tables III and IV for details) and τ_4 is schedulable at priority 3. With τ_2 at level 2 it is now schedulable. It now has no messages of higher priority but LO criticality, so $R_2^s(HI) = 2+3+2 = 7$ and $R_2(HI) = 7 + 2 = 9$ which is within the message's deadline. We also note that the triggering message, that is only transmitted once and is given the highest priority of all is schedulable.

$\mathcal{M}$	Crit	T(HI)	T(LO)	D	C	Pri	$R^s(LO)$	$R(LO)$
τ_2	HI	12	24	12	2	2	3	5
τ_3	LO	-	11	11	2	4	7	9
τ_4	LO	-	6	6	1	3	5	6
τ_5	HI	18	36	18	3	5	9	12

TABLE III
EXAMPLE MESSAGE SET – NEW PRIORITIES, LO-MODE

$\mathcal{M}$	Crit	T(HI)	T(LO)	D	C	Pri	$R^s(HI)$	$R(HI)$
τ_1	HI	∞	-	5	2	1	3	5
τ_2	HI	12	24	12	2	2	7	9
τ_5	HI	18	36	18	3	5	15	18

TABLE IV
EXAMPLE MESSAGE SET – NEW PRIORITIES, HI-MODE

VI. A BASIC MIXED CRITICALITY PROTOCOL FOR CAN

With the full MixedCAN protocol, as defined in the previous section, there is a performance trade-off between sending (extra) specific messages to induce a criticality mode change and preventing LO-crit messages being sent after the change. If for a particular application the analysis does not allow many LO-crit messages to be discounted then it may not be useful to broadcast the mode change to other nodes. Rather a basic form of the protocol can be defined in which the TNC's only role is to prevent LO-crit messages from been sent too early; and the analysis is used to determine that:

1) All LO-crit messages meet their deadlines if all messages have their LO-crit values (i.e. the system is in LO-crit mode). This is determined by the use of equations (6) and (7).
2) All HI-crit messages meet their deadlines when their HI-crit parameters are used (i.e. the system is in HI-crit mode) and LO-crit messages continue to be transmitted with their LO-crit parameters. This is determined by the use of equation (12) derived below and equation (11).

The response-time equation for $R_i^s(HI)$ with this more basic protocol is derived from equation (8) by dropping the mode change messages but prolonging the the time that LO-crit messages can interfere.

$$R_i^s(HI) = \hat{B}_i(LO) + \sum_{\tau_j \in \mathbf{hpH}(i)} \left\lceil \frac{R_i^s(HI) + J_j + \delta}{T_j(HI)} \right\rceil C_j + \sum_{\tau_k \in \mathbf{hpL}(i)} \left\lceil \frac{R_i^s(HI) + J_j + \delta}{T_k(LO)} \right\rceil C_k + \hat{F}_i(HI), \quad (12)$$

We term this new protocol Basic MixedCAN (BMC). It is straightforward to show that Audsley's algorithm for optimal priority assignment is needed and is applicable.

VII. EVALUATION

In this section, we present the results of an empirical evaluation, which examines the effectiveness of the BMC analysis and the MixedCAN protocol and its analysis. In our experiments, we compare the following schemes:

- PartitionCAN – Assigns HI-crit messages higher priorities than LO-crit messages. (Uses Deadline Monotonic Priority Ordering (DMPO) within the subsets containing LO- and HI-crit messages).
- StandardCAN – Assumes the worst-case parameters for all messages, ignoring criticality. Uses DMPO.
- BMC - Determines the schedulability of LO-crit messages in LO-crit mode, and that of HI-crit messages in both HI- and LO-crit modes. Uses Audsley's algorithm for priority assignment.
- MixedCAN – Uses the protocol and analysis developed in this paper. Uses Audsley's algorithm for priority assignment.
- UB-H&L-CAN - A necessary test providing an upper bound on schedulability. Only checks that LO-crit messages are schedulable in LO-crit mode, and that HI-crit messages are schedulable in HI-crit mode. Assumes DMPO.

We note DMPO is optimal for PartitionCAN, StandardCAN and UB-H&L-CAN for the parameters used in the evaluations (i.e. all transmission time are identical as are the blocking times, jitter is zero and the sufficient test (1) is used).

The effectiveness of the schemes is measured in two ways, (i) via the success ratio; the percentage of message sets that are deemed schedulable at each utilisation level, and (ii) via a weighted schedulability measure [6]. Our use of this weighted measure follows the approach we used elsewhere [4].

We note that all four approaches to using CAN for messages of different criticality need a TNC. Here we are simply comparing their ability to schedule message sets, and hence examining under what conditions it is worth deploying MixedCAN with its more complex protocol.

We noted earlier that with MixedCAN a switch to the HI-crit mode can be due to a number of reasons (explicit request from the host node, error counts too high and messages being queued too quickly etc.). Also there could be messages that are only transmitted in the HI-crit mode. To simplify the comparison we focus on differing fault tolerance and message periods in the two modes. We set $F(LO)$ to be zero, and consider values of $F(HI)$ from 3 to 30. We also allow the periods of messages in the HI-crit mode to be as little as one fifth of their values in the LO-crit mode.

A. Message set parameter generation

The message set parameters used in our experiments were generated as follows:

- Message sets were created containing 20 to 120 messages (default n=80).
- LO criticality message periods were generated according to a log-uniform distribution with a factor of 100 difference between the minimum and maximum possible message period. This represents a spread of message periods from 10ms to 1 second, as found in many automotive applications.
- HI criticality message periods were derived from their LO-crit values using a fixed divisor of the LO-crit period, $T_i(HI) = T_i(LO)/CF$ (e.g. $CF = 2.0$).
- Message deadlines were set equal to $T(LO)$ for LO-crit messages and $T(HI)$ for HI-crit messages.
- The transmission time of each message corresponded to an 8 data byte message with maximum bit stuffing (i.e.

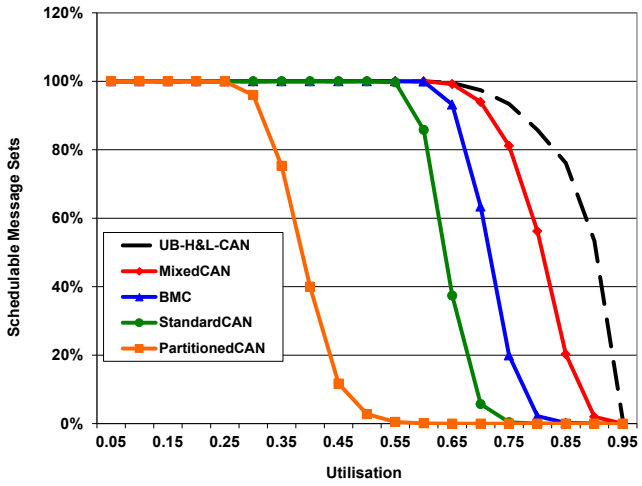

Fig. 1. Success Ratios for Varying Message Periods

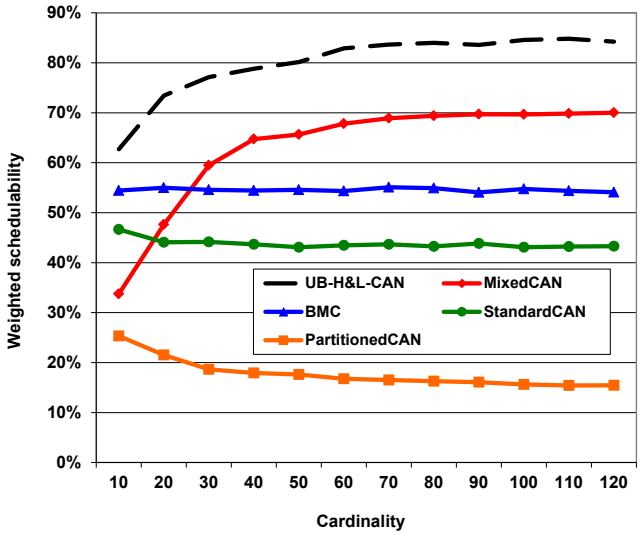

Fig. 2. Weighted Schedulability for Varying Message Periods

135 bits).

- The blocking factor for all messages was set to the largest possible value, i.e. assuming the presence of maximum length soft real-time messages at a lower priority.
- The probability that each generated message was a HI-crit message was given by the parameter CP (e.g. $CP = 0.5$).
- The CAN bus speed was adjusted to give the required total utilisation for each message set, with utilisation measured only in the LO-crit mode.
- The number of faults that must be tolerated by each message was set to zero in LO-crit mode and to a value $F(HI)$ for HI-crit mode (e.g. $F(HI) = 15$).
- Additional $GoHI$ messages were included in the analysis for MixedCAN.

B. Varying message periods and deadlines

In our first set of experiments, we assumed a fault-free CAN bus, with HI-crit messages having reduced periods in the HI-crit mode.

Figure 1 plots the success ratio for each scheme against message set utilisation, for message sets of cardinality $n = 80$, with each message having a 50% probability of being a HI criticality message ($CP = 0.5$), and a period in HI-crit mode that was half that in LO-crit mode ($CF = 2.0$).

The results show that message set schedulability is significantly improved using the MixedCAN scheme over BMC and the StandardCAN approach. All of these approaches significantly out-performed Partitioned-CAN, which has relatively poor performance due to priority inversion.

Figure 2 shows the weighted schedulability measure for message sets (with $CP = 0.5$, $CF = 2.0$) plotted against message set cardinality (size). The interesting aspect of these results is the behaviour of MixedCAN. For small sets of messages (e.g. $n = 10$), the performance of MixedCAN is relatively poor. This is because with only 10 messages, the bus speed has to be reduced to a low value to obtain high bus utilisation, and at these low bus speeds the extra overheads of the MixedCAN protocol impinge heavily on message schedulability. This is not the case for larger, more realistic examples with 50+ messages where MixedCAN performs well.

C. Varying numbers of faults

In our second set of experiments, we considered sets of messages where the periods of HI-crit messages were the same in both modes; however, we now considered different levels of fault tolerance in the two modes. HI-crit messages had to be schedulable in HI-crit mode, tolerating $F(HI)$ faults, and all messages had to be schedulable in LO-crit mode with no faults. Each fault on the bus was assumed to affect the longest possible message (135 bits) and cause an error overhead of an additional 31 bits.

Figure 3 plots the success ratio for each scheme against message set utilisation, for message sets of cardinality $n = 80$, with each message having a 50% probability of being a HI criticality message ($CP = 0.5$), and a fault tolerance of $F(HI) = 15$. Here, we observe that the BMC scheme outperforms MixedCAN. This is because with a large fault tolerance, the priority of HI-crit messages need to be significantly higher than that of LO-crit messages with similar deadlines. Thus there are few high priority but LO-crit messages and so the overheads of the MixedCAN scheme out-weight the gains made in stopping higher priority LO-crit messages from being sent when the system enters HI-crit mode.

Figure 4 shows the weighted schedulability measure for message sets (with n=80, $CP = 0.5$, $CF = 1.0$) plotted against the fault tolerance in HI-crit mode. Here, we observe that MixedCAN shows a very small performance gain over BMC for fault tolerance levels of 6-9 errors, which then reverses for larger numbers of errors for the reasons described above. As the required fault tolerance level increases, all schemes tend towards the performance of Partitioned-CAN. This is because, in these circumstances, HI-crit messages have

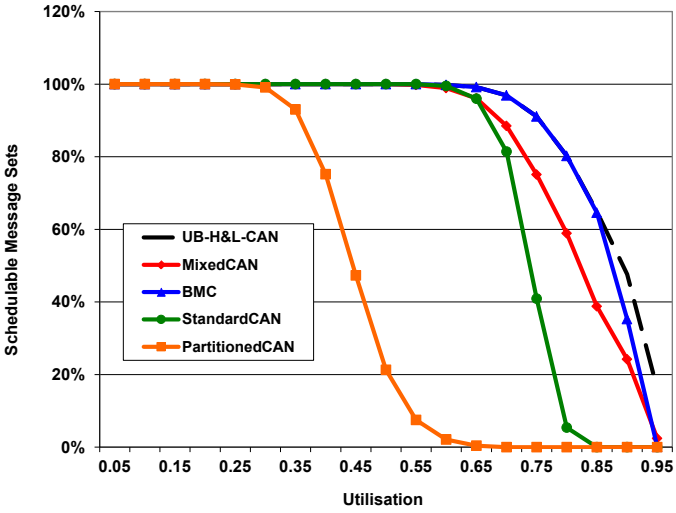

Fig. 3. Success Ratios for Varying Faults

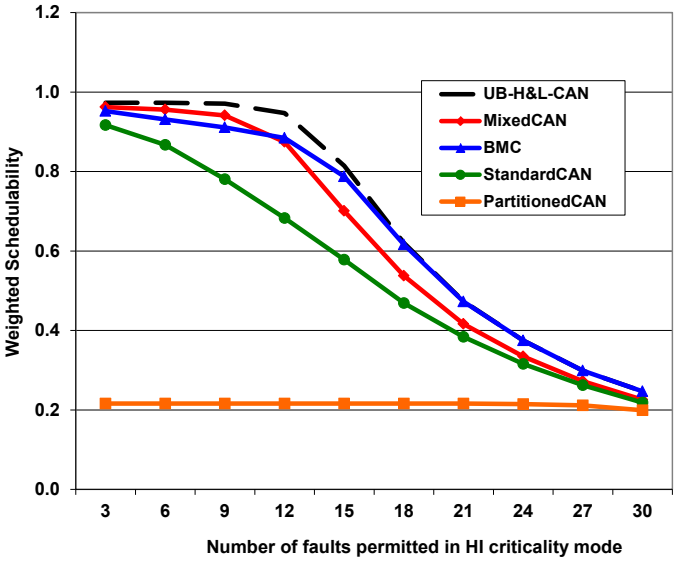

Fig. 4. Weighted Schedulability for Varying Faults

to be given the highest priorities in order to be schedulable at all. We note that the performance of Partitioned-CAN is essentially flat as it is constrained by the schedulability of the *LO*-crit messages.

VIII. Conclusions

Industrial pressure from the automotive sector will require CAN to be used to support mixed criticality systems. In this paper we have investigated what form this support should take. A trusted network controller (TNC) is an imperative. It will prevent lower criticality messages from misusing the network. The use of a TNC is then augmented by a protocol (MixedCAN) that exploits the different characteristics of the different criticality levels to deliver high utilisation of the network. MixedCAN and a more basic form of the protocol (BMC) are evaluated and shown to perform much better than partitioning message priorities or ignoring criticality. The evaluations show when the full protocol is particularly effective, and when the simpler form outperforms it. As part of future work tighter, though inevitably not exact, analysis will be sort for the MixedCAN model.

Acknowledgements The research described in this paper is funded, in part, by the ESPRC grant, MCC (EP/K011626/1).

References

[1] N. Audsley. On priority assignment in fixed priority scheduling. *Information Processing Letters*, 79(1):39–44, 2001.

[2] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings. Applying new scheduling theory to static priority preemptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

[3] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie. Scheduling real-time mixed-criticality jobs. *IEEE Transactions on Computers*, 61(8):1140–1152, 2012.

[4] S. Baruah, A. Burns, and R. I. Davis. Response-time analysis for mixed criticality systems. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 34–43, 2011.

[5] S. Baruah and S. Vestal. Schedulability analysis of sporadic tasks with multiple criticality specifications. In *ECRTS*, pages 147–155, 2008.

[6] A. Bastoni, B. Brandenburg, and J. Anderson. Cache-related preemption and migration delays: Empirical approximation and impact on schedulability. In *Proc. of Sixth International Workshop on Operating Systems Platforms for Embedded Real-Time Applications*, pages 33–44, 2010.

[7] I. Broster and A. Burns. An analysable bus-guardian for event-triggered communication. In *Proc. of the 24th Real-time Systems Symposium*, pages 410–419, Cancun, Mexico, Dec 2003. Computer Society, IEEE.

[8] I. Broster, A. Burns, and G. Rodríguez-Navas. Probabilistic analysis of CAN with faults. In *Proc. of the 23rd Real-time Systems Symposium*, Austin, Texas, Dec 2002. IEEE.

[9] A. Burns and S. Baruah. Timing faults and mixed criticality systems. In Jones and Lloyd, editors, *Dependable and Historic Computing*, volume LNCS 6875, pages 147–166. Springer, 2011.

[10] R. Davis and A. Burns. Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. In *Proc. of IEEE Real-Time Systems Symposium (RTSS)*, pages 398–409, 2009.

[11] R. Davis and A. Burns. Robust priority assignment for messages on controller area network (CAN). *Real-Time Systems*, 41(2):152–180, 2009.

[12] R. Davis and A. Burns. Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. *Real-Time Systems Journal*, 48:1–40, 2010.

[13] R. Davis, A. Burns, R. Bril, and J. Lukkien. Controller area network (CAN) schedulability analysis: Refuted, revisited and revised. *Journal of Real-Time Systems*, 35(3):239–272, 2007.

[14] R. Davis, S. Kollmann, V. Pollex, and F. Slomka. Schedulability analysis for controller area network (CAN) with FIFO queues priority queues and gateways. *Real-Time Systems*, 49:73–116, 2013.

[15] R. B. GmbH. CAN specification version 2.0. Technical report, Postfach 30 02 40, D-70442 Stuttgart, 1991.

[16] B. Huber, C. El Salloum, and R. Obermaisser. A resource management framework for mixed-criticality embedded systems. In *34th IEEE IECON*, pages 2425–2431, 2008.

[17] J. Laprie. Dependable computing and fault tolerance: Concepts and terminology. In *Digest of Papers, The Fifteenth Annual International Symposium on Fault-Tolerant Computing*, pages 2–11, Michigan, USA, 1985.

[18] P. Pedro and A. Burns. Schedulability analysis for mode changes in flexible real-time systems. In *10th Euromicro Workshop on Real-Time Systems*, pages 172–179. IEEE Computer Society, 1998.

[19] J. Ruffino, P. Verissimo, G. Arroz, C. Almeida, and L. Rodrigues. Fault-tolerant broadcast in CAN. In *Proc. of the 28th FTCS, Munich*. IEEE Computer Society Press, 1998.

[20] K. Tindell, A. Burns, and A. J. Wellings. Mode changes in priority preemptive scheduled systems. In *Proc. Real Time Systems Symposium*, pages 100–109, Phoenix, Arizona, 1992.

[21] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS)*, pages 239–243, 2007.