A Practical Message ID Assignment Policy for Controller Area Network that Maximizes Extensibility

Florian Pölzlbauer
Virtual Vehicle Research Center, Austria
florian.poelzlbauer@v2c2.at

Robert I. Davis
University of York, UK and INRIA, France
rob.davis@york.ac.uk

Iain Bate
University of York, UK
iain.bate@york.ac.uk

ABSTRACT

Controller Area Network (CAN) is widely used in automotive applications. Policies for message ID and thus priority assignment have a significant impact on schedulability. In addition, they also determine extensibility; the scope to add messages required by future upgrades without compromising schedulability. In this paper we address message ID assignment, such that the system is extensible. First, we provide an assessment metric that provides an in-depth view of the extensibility of a given ID-assignment, tailored for use in automotive applications. Second, we develop a practical ID-assignment policy which maximizes extensibility. This policy provides an upgrade pathway: it is used to provide the initial ID-assignment, and also used for ID-assignments during subsequent upgrades. The policy optimizes extensibility by maintaining Deadline minus Jitter Monotonic Priority Ordering, which ensures that it does not compromise either schedulability or robustness to errors on the bus. Evaluation using a simple automotive benchmark shows the effectiveness of the policy over multiple upgrades.

1. INTRODUCTION

Controller Area Network (CAN) [3, 8] is a broadcast communications bus which is widely used in automotive systems. Each CAN message is uniquely identified by its ID, which also determines its priority during the bus arbitration phase. A key aspect of configuring a CAN-based system is to assign an appropriate ID to each message. This must be done in such a way that the system is schedulable (i.e. all messages meet their deadlines). Many automotive systems are developed in an *incremental* manner, with additional software and hardware components added to the system at a later date (e.g. for vehicle mid-life upgrades). Often it is not possible due to the costs involved in re-verifying components, and the logistics involved in carrying different parts for different cars, to change existing message IDs on an upgrade. This means that ID assignment must be done in such a way that the system is *extensible*. On an upgrade, it must be possible to add additional messages to the system and set their IDs, without changing the IDs of the existing messages. Further, the system must remain extensible to support further enhancements along the upgrade path.

In 2015, Davis et al. [9] presented a method for assigning IDs to new messages while maintaining the fixed IDs of existing messages, thus ensuring backwards compatibility. This addresses the question of how to perform a system upgrade. However, the second aspect of the problem was not tackled: how to assign IDs in such a way that future upgrades can be performed without compromising schedulability? In other words, how to provide an *extensible ID-assignment* policy? In this paper, we aim to answer that question.

From the literature on real-time systems we know that priority ordering has a significant impact on schedulability in fixed priority systems [5]. CAN effectively uses fixed priority non-preemptive scheduling of messages. Audsley's Optimal Priority Assignment (OPA) [1, 2] algorithm is known to be optimal [8] in this case, in the sense that it provides a schedulable priority ordering whenever one exists. In specific cases, i.e. with all messages of the same length and soft real-time diagnostic messages at low priorities, Deadline minus Jitter Monotonic Priority Ordering (DJMPO) [27] has also been shown to be optimal [9].

When it comes to extensibility, there is an important difference between *priority ordering* and *ID-assignment*. The former simply provides a logical or relative priority ordering between the messages, whereas the latter specifies the actual IDs used. Thus for a given priority ordering, there are many possible ID assignments. In contrast, a particular ID-assignment implies a single specific priority ordering.

The priority ordering chosen for a set of messages has a significant effect on its schedulability. While all of the ID-assignments with the same priority ordering have the same schedulability and robustness to errors on the bus [7], they can have very different properties in terms of extensibility. For example if the messages are assigned consecutive IDs, then there will be no flexibility to interleave new messages between them and hence schedulability in the event of an upgrade (i.e. extensibility) may be compromised.

This leads to the following research questions: How to assess the extensibility of a CAN configuration? How to assign IDs to CAN messages, such that extensibility is maximized at each stage along an upgrade path, without prior knowledge of the upgrades? Answers to the second question depend on the starting point and hence there are two flavours of the problem: (i) starting from an empty configuration, in other words applying an extensible ID-assignment policy to determine the first and all subsequent configurations; (ii) starting from a configuration with some arbitrary fixed ID-assignment. In this paper, we address (i), assuming that the extensible ID-assignment policy we propose can be used for the entire life-cycle of a system. We note that, due

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

to the re-use of legacy components, the opportunity to start from scratch is rare in the automotive industry. Addressing (ii) via an extensible ID-assignment policy that works from an arbitrary starting point therefore forms the focus of our ongoing work.

To address the research questions set out above, we apply the following methodology: After giving a brief insight into the CAN protocol and schedulability tests, we give a definition of *extensibility* relevant to the use of CAN in the automotive industry. We then provide an assessment *metric* that determines the extensibility of a given CAN configuration. We use this metric to examine the performance of a number of simple message ID-assignment schemes applied to a well-known automotive case study. This enables us to evaluate the effectiveness of the metric, and also observe what the key factors are in improving the extensibility of a CAN configuration. Based on this insight, we develop an *extensible ID-assignment policy*, and evaluate its performance on a set of consecutive system upgrades to the case study. Finally, we prove that the extensible ID-assignment policy is optimal under certain conditions.

This paper makes two main contributions: First, we introduce a metric and assessment method for determining the degree of extensibility of a CAN configuration. This metric gives detailed insight into *where*, i.e. at which deadlines and periods, and by *how much*, i.e. by how many additional messages or payload bytes, a CAN configuration can be extended and yet remain schedulable. Second, we introduce a practical ID-assignment policy which maximizes extensibility of CAN configurations, tailored for use in an automotive context. This ID-assignment policy is applied to derive the initial configuration, and then again at each upgrade. Thus the policy provides an extensible upgrade path.

1.1 Related Work

Research on priority assignment for CAN has mainly focussed on optimal priority assignment policies, with Audsley's OPA algorithm [1, 2] proved optimal for systems using *priority queues* [8], and DJMPO [27] proved optimal with some common constraints on the sets of messages [9]. Further work has explored the issues that can arise if the priority-based arbitration mechanism is circumvented, for example by the use of *FIFO queues*. In this case, effective priority assignment can be achieved by grouping messages that share the same FIFO queue into priority bands and then applying Audsley's algorithm or DJMPO at the level of bands rather than individual messages [10, 11].

In practice, even for well behaved systems with priority queues, using an optimal priority assignment policy is not in itself enough. The priority ordering generated could potentially leave the system only just schedulable, and thus vulnerable to deadline misses in the event that there is an increase in interference, for example due to errors on the bus. Work on *robust priority ordering* [6, 7] addresses this problem by generating a priority ordering that is not only optimal, but also tolerates the maximum amount of additional interference (i.e. is robust as well). Recent work in this area provides a robust priority assignment for new messages added to a system where the existing message IDs are fixed [9], addressing flaws in previous work [20].

One might expect that a system that is robust and can tolerate the maximum additional interference would also be extensible; however, this is not necessarily the case. In order for a CAN configuration to be extensible, two aspects need to hold: (i) The system must be able to tolerate additional interference, since new messages will certainly generate such interference. Effectively there must be sufficient *time resource* to accommodate the new messages. (ii) There must be enough free IDs with appropriate values so that the new messages can be assigned IDs while preserving an effective, ideally optimal, priority ordering. This can be seen as a *space resource*. Robust priority assignment tackles only the *time resource* aspect of the extensibility problem.

In [24] a time-triggered system is optimized towards extensibility. The schedule is designed such that some time-slots are initially unused. These empty time-slots can be utilized later to host additional messages. This approach tackles the *space resource* aspect, but only for time-triggered systems.

In [17] a CAN configuration is made extensible using Simulated Annealing (SA). The problem is tackled at two levels. At the message level, extensibility is addressed in terms of packing signals into messages, while at the system level extensibility is considered in terms of priority ordering. At the end of the optimization process, the priority ordering is transformed into an ID-assignment by evenly spacing the message IDs over the available range.

Work on system design also tackles issues of robustness. In [25, 26] task allocation is solved in such a way that end-to-end deadlines are met and the system can tolerate the maximum increase in task worst-case execution times. However, in terms of the CAN configuration, only the priority ordering is tackled. In [13] task allocation is solved in such a way that the system can tolerate the addition of tasks for future upgrades. Again, in terms of the CAN configuration, only priority ordering is considered; the assignment of specific message IDs is not addressed.

2. CONTROLLER AREA NETWORK (CAN)

In this section, we recap on the CAN protocol, the system model used in the rest of the paper, and the basic schedulability analysis for CAN.

CAN is an asynchronous multi-master serial data bus that uses Carrier Sense Multiple Access / Collision Resolution (CSMA/CR) to determine access to the bus. The CAN protocol [3] requires that ECUs (Electronic Control Units) connected to the network wait for a bus idle period before attempting to transmit. If two or more ECUs attempt to transmit messages at the same time, then the message with the lowest numeric ID will win arbitration and be sent. The other ECUs will cease transmitting and wait until the bus becomes idle again before attempting to re-transmit their messages. In effect CAN messages are sent according to fixed priority non-preemptive scheduling, with the message IDs acting as priorities. IDs can be 11 or 29 bits long, thus allowing 2032 or 532 676 608 unique IDs[1]. Duplicate IDs are not permitted, as this would lead to errors as multiple ECUs tried to transmit messages at the same time.

2.1 System Model

We now briefly describe the system model and notation used in the paper. The system is assumed to comprise a number of ECUs connected to each other via a CAN bus. Each ECU is assumed to ensure that, at any given time when arbitration starts, the highest priority message queued at that ECU is entered into arbitration on the bus. The system is assumed to contain a set of hard real-time messages, each statically assigned to a single ECU. Each message m has

[1]Note that the 7 most significant bits of the ID are not permitted to be all 1s. Hence the number of valid unique IDs is $2^{11} - 2^4 = 2032$ for 11 bit IDs, and $(2^{11} - 2^4) \cdot 2^{18} = 532\,676\,608$ for 29 bit IDs.

a unique ID which determines its priority. Each message m has a *maximum transmission time* of C_m, a minimum inter-arrival time or *period* of T_m, and a hard *deadline* D_m. The deadline of each message is constrained, i.e. $D_m \leq T_m$. Each message m is assumed to be placed in a queue and available for transmission in a bounded but variable amount of time between 0 and J_m after its initiating event. J_m is referred to as the *release jitter* of the message. The *worst-case response time* R_m is defined as the maximum possible delay from the initiating event for an instance of message m, until it completes transmission. A message is said to be *schedulable* if its worst-case response time does not exceed its deadline ($R_m \leq D_m$). A system is said to be schedulable if all of the messages in the system are schedulable.

2.2 CAN Schedulability Analysis

For a given priority ordering, schedulability analysis [8] can be used to determine the worst-case response time R_m for each message m in the system. The message response time is composed of three terms: the release jitter J_m, the queueing delay w_m and the maximum transmission time C_m.

$$R_m = J_m + w_m + C_m \tag{1}$$

$$R_m \leq D_m \tag{2}$$

The maximum transmission time is determined by the message payload s_m (1 to 8 bytes), the ID-format (11 or 29 bit) and the bit-time τ_{bit} (i.e. the time it takes to transmit a single bit, which is derived from the baud rate).

$$C_m^{\text{11 bit ID}} = (55 + 10s_m)\,\tau_{\text{bit}} \tag{3}$$

$$C_m^{\text{29 bit ID}} = (80 + 10s_m)\,\tau_{\text{bit}} \tag{4}$$

The queueing delay w_m is determined by two factors: the blocking factor B due to non-preemptive message transmission, and the interference due to higher priority messages (denoted by the set $hp(m)$).

$$w_m^{n+1} = B + \sum_{\forall k \in hp(m)} \left\lceil \frac{w_m^n + J_k + \tau_{\text{bit}}}{T_k} \right\rceil C_k \tag{5}$$

A simple upper bound on the blocking factor B is given by the transmission time of the longest message on the network.

$$B = \max_{\forall m} \{C_m\} \tag{6}$$

$$B = B^{\max} \tag{7}$$

Many automotive systems feature low priority soft real-time diagnostic messages. These messages typically have the maximum payload of 8 data bytes; a safe upper-bound $B^{\max}$ can be set accordingly. Note that (5) combined with (6) in general provides a *sufficient* schedulability test; however, in the case where there are low priority soft real-time diagnostic messages, then it is *exact*[2].

3. SAE BENCHMARK

Throughout this paper we use a *running example* to demonstrate different aspects of extensibility and our methodology. For this purpose we utilise the well-known and openly available SAE benchmark [21]. Although somewhat old, this benchmark is indicative of the use of CAN in automotive systems, with groups of messages with the same periods and deadlines, and periods that are chosen from a relatively small set of harmonic values. This is common in automotive systems [14], and is is rooted in the underlying micro-controller and RTOS behaviour where tasks are triggered by a set of timers with pre-defined periods. Although the SAE benchmark is small by modern standards (typical automotive systems may contain of the order of 100 messages or more on each network), it serves our purposes as an example with a realistic set of harmonic periods and deadlines. Table 1 provides details of the benchmark. The case study has a bus utilization of approx. 44% at a baud rate of 250kBit/sec. Note, for simplicity, we assume that the release jitter is zero for all messages.

Details of a larger case study based on one of the CAN buses of an experimental vehcile with 69 messages and 6 ECUs [23] can be found in the Appendix of the technical report on which this paper is based [18].

Message	Format	Size [byte]	T [ms]	D [ms]
m01	Standard	1	50	5
m02	Standard	2	5	5
m03	Standard	1	5	5
m04	Standard	2	5	5
m05	Standard	1	5	5
m06	Standard	2	5	5
m07	Standard	6	10	10
m08	Standard	1	10	10
m09	Standard	2	10	10
m10	Standard	3	10	10
m11	Standard	1	50	50
m12	Standard	4	100	100
m13	Standard	1	100	100
m14	Standard	1	100	100
m15	Standard	3	1000	1000
m16	Standard	1	1000	1000
m17	Standard	1	1000	1000

Table 1: SAE benchmark

3.1 Example ID-Assignments schemes

There are many different ways in which IDs can be assigned to the messages in the SAE benchmark. Some example schemes which we use to understand *how* and *why* different ID-assignments impact extensibility are listed below. The ID assignments produced when these schemes are applied to the SAE benchmark are shown in Table 2.

`DM at lowest IDs / DM at middle IDs / DM at highest IDs` – The messages are in Deadline Monotonic (DM) priority order (effectively DJMPO given that the release jitter is zero), with IDs assigned consecutive values in the lowest / middle / highest part of the range.

`DM evenly spaced IDs` – The messages are in DM priority order, with IDs evenly spaced throughout the available range [17].

`By ECU` – We split the range of IDs into 32 ID-bands[3], each containing 64 IDs. Each ECU is assigned an ID-band. The messages which are sent by the ECU are allocated IDs within the associated band. (This approach, grouping IDs by sending ECU, is sometimes used in automotive systems, with each ECU supplier allocated a range of consecutive IDs to use). Within an ID-band, messages are in DM order.

`Random` – IDs are assigned at random. We use this approach in order to simulate ID-assignment according to an approach which is not linked to any consideration of time constraints.

[2]Assuming maximum bit-stuffing.

[3]Due to the electrical properties of CAN, there is a physical limit on the number of ECUs that can be connected to a single CAN bus. This depends on topology, baudrate, and transceiver details, and is approx. 30-40 ECUs.

Message	DM at lowest IDs	DM at highest IDs	DM at middle IDs	DM evenly spaced	by ECU	random
m01	0	2015	1015	100	1792	870
m02	1	2016	1016	220	1408	832
m03	2	2017	1017	340	256	814
m04	3	2018	1018	460	1024	1743
m05	4	2019	1019	580	576	1215
m06	5	2020	1020	700	832	1543
m07	6	2021	1021	820	833	1827
m08	7	2022	1022	940	1793	1591
m09	8	2023	1023	1060	257	1146
m10	9	2024	1024	1180	1025	378
m11	10	2025	1025	1300	1409	7
m12	11	2026	1026	1420	1794	1199
m13	12	2027	1027	1540	1410	1460
m14	13	2028	1028	1660	577	1220
m15	14	2029	1029	1780	1795	1697
m16	15	2030	1030	1900	578	1626
m17	16	2031	1031	2020	834	1828

Table 2: ID-assignments for SAE benchmark

Note the semantic difference between *priority ordering* and *ID-assignment*. Priority ordering relates only to the relative priority between messages, while ID-assignment specifies the actual ID values. For example, all assignments in Table 2 that are marked `DM` have the same priority-ordering (i.e. DJMPO), but very different ID-assignments.

4. ASSESSMENT OF EXTENSIBILITY

Our goal, in this section, is to provide a metric / assessment method which can determine, in a detailed manner, the degree of extensibility of a CAN configuration. Ideally, the *extensibility metric* should be capable of providing engineers with a detailed insight into *why* a particular CAN configuration is or is not extensible.

From an engineering perspective, the extensibility of a CAN configuration revolves around one key question: How much additional payload data can be transmitted over the network? This can either be implemented by (i) packing the additional payload data into existing messages [16] which are not fully utilized, or by (ii) adding new messages to the system [9]. Option (i) is more efficient since no additional overhead is introduced; however, it is limited by many factors. These include: the maximum allowed payload of 8 bytes, the source ECU, and the message periods and deadlines. Option (ii) offers more flexibility, but introduces additional overhead, and is limited by the availability of free IDs. In this paper, we focus on option (ii).

4.1 Existing Metrics

In the literature various metrics have been used to explore the sensitivity of a system to changes in its parameters. These include:

- *Breakdown utilization* [15]: In the context of CAN this corresponds to lowering the baud rate until the system is only just schedulable [10],
- *Robustness*: Adding additional interference until the system is only just schedulable [6, 7]
- *Sensitivity analysis* [19], increasing C [25, 26] or decreasing D [17] until the system is just schedulable.

The question is: Are these metrics sensitive to differences in ID-assignments? Table 3 shows how these metrics perform, when applied to the example ID-assignment schemes from Table 2. The results show that these metrics are sensitive to differences in relative priority ordering; however, for the same priority ordering (DM in this case), there is absolutely no sensitivity to the actual ID assignment.

ID assignment	min baudrate [bits/sec]	interference [bit]	Δ_C	Δ_D
DM at lowest IDs	123k	715	2.139	0.428
DM at highest IDs	123k	715	2.139	0.428
DM at middle IDs	123k	715	2.139	0.428
DM evenly spaced IDs	123k	715	2.139	0.428
By ECU	227k	115	1.112	0.908
Random	241k	50	1.046	0.960

Table 3: Existing metrics applied to ID-assignments

4.2 Novel Assessment of Extensibility

Given that existing metrics are only sensitive to relative priority assignment, we need to devise an extensibility metric which is (i) sensitive to differences in ID-assignments, (ii) provides detailed insight into *why* an ID-assignment is extensible, and (iii) presents this information in a way that is intuitive for engineers.

In our view, the most intuitive way to measure the extensibility of a CAN configuration is to simply estimate or analyse how many messages can be added before the system becomes unschedulable. However, this approach needs to be clarified, since it is based on a set of assumptions. Before carrying out such analysis, we need to decide on the following parameters for the added messages: size, deadline, period, and also the priorities/IDs used. In addition, the question of comparability arises. For example, which of the following two configurations is better in terms of extensibility: A configuration that can be extended by adding 7 messages with a period of 10 ms, or one that can be extended by adding 40 messages with a period of 100 ms? Each may be more or less favourable, depending on specific needs.

We propose assessing the extensibility of CAN configurations using a method which provides the detailed information needed to support engineering decisions. The assessment proceeds as follows. We add a set of N messages to the system. Each of these messages has a specified size, deadline, and period. The IDs of these messages are assigned using the function `priorityAssignment()` which implements Algorithm 2 from [9]. This is an optimal[4] method of assigning IDs to new messages in the presence of an existing set of messages with fixed IDs. By using a binary search, we find the maximum number $N_{\max}$ of such messages which may be added (all at the same time). Once this number has been found, we try via the function `addLastSmallMessage()` to add one more message, allowing a smaller payload than the previous messages. This way we can make our assessment more precise. Algorithm 1 provides the pseudo-code.

The assessment is performed for a set of pre-defined periods and implicit deadlines. This way we get a detailed insight into *where* a CAN configuration is extensible. (Note, although we use a set of pre-defined periods, it is the set of deadlines derived from them that matter most here). The results indicate the maximum number of messages that can be added at each of the pre-defined periods and deadlines individually, thus indicating *by how much* the configuration can be extended. In order to make the results comparable, and easier to visualise, we normalize them by the periods.

[4]Optimal in terms of schedulability, but not extensibility.

Algorithm 1: Extensibility Assessment

Input: messages /* messages with fixed IDs */
Input: *pay* /* size of message payload */
Input: $\{T_1, ..., T_t\}$ /* set of pre-defined periods */
for *each* T **do**
  $D = T$;
  $n_{upper} = 2032$;
  $n_{lower} = 0$;
  repeat
    $n = (n_{upper} + n_{lower})/2$;
    newMessages = generateMessages(n, pay, T, D);
    priorityAssignment(messages, newMessages);
    if *schedulable* == *true* **then**
      $n_{lower} = n$;
    else
      $n_{upper} = n$;
    end
  until *schedulable* == *true* ***and*** $(n_{upper} - n_{lower}) \leq 1$;
  pay_{last} = addLastSmallMessage(T, D);
  $ext = (n \cdot pay + pay_{last})/T$;
end
Output: extensibility for each period

For those engineers who are not primary interested in the number of messages, we also output the number of payload bytes. Hence, our extensibility metric is:

$$ext. = \frac{\text{added messages}}{\text{period}} \quad \forall \text{ pre-defined deadlines}$$

$$ext. = \frac{\text{added payload bytes}}{\text{period}} \quad \forall \text{ pre-defined deadlines}$$

Our approach can be tuned via three parameters: the deadline D, the period T, and the payload size s of the added messages. Since the assessment method is intended to be effective in practice in typical automotive systems, the periods and deadlines are set according to values observed in real-world automotive systems, following the periods given in [14] ($T = \{1, 2, 5, 10, 20, 50, 100, 200, 500, 1000\}$ ms). The deadlines are set equal to periods ($D = T$), as this is the case in many applications. Finally, the payload size is set to the maximum allowed value (i.e. $s = 8$ bytes). Note we could also include a fixed value for the release jitter, but have deliberately kept the example simple by assuming $J = 0$.

Note that our approach to assessing extensibility consists of several components: a metric (per period and deadline), a set of scenarios (i.e. the set of pre-defined periods), an optimal method for adding messages, and a way of combining the individual results into an overall results plot. For the sake of simplicity and readability, we refer to all of this as the *extensibility metric*.

4.3 Effectiveness of the Extensibility-Metric

Considering the effectiveness of the extensibility metric, the most important questions which arise are: How well does the metric perform? Is it sensitive to differences in ID-assignments? Can it provide insight into why an ID-assignment is extensible?

In order to answer these questions, we apply the metric to the SAE benchmark using the example schemes from Section 3.1 to assign the message IDs. Figure 1 depicts the results[5]. Note that the plot has to be read according to an OR syntax, i.e.: we can add x payload bytes at T_a ms, OR y payload bytes at T_b ms, OR z payload bytes at T_c ms, etc. We cannot add all these messages at the same time.

[5]All the plots in the paper are best viewed online in colour.

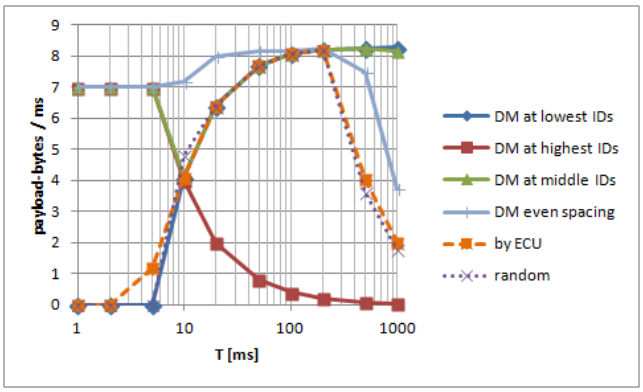

Figure 1: Extensibility of SAE Benchmark for different ID-Assignments

We now comment on the performance of the different schemes.

`DM at lowest IDs`: we cannot add any messages with short deadlines (1, 2, 5 ms); however, we can add a significant number of medium and long deadlines messages (10 to 1000 ms). This is because all low IDs (i.e. high priorities) have already been used, and so the new messages can only be placed at lower priorities. Thus, only new messages with long deadlines are schedulable.

`DM at highest IDs`: we can add some messages with short deadlines (1, 2, 5 ms). For medium and long deadlines (10 to 1000 ms), the number of messages that can be added is constant. Due to the normalization by period, the extensibility metric decreases accordingly. This is because all high IDs (i.e. low priorities) have already been used, and so the new messages can only be placed at high priorities. This causes interference on the existing messages from the SAE benchmark, thus limiting the number of new messages which can be added. The bottleneck is that existing short-deadline messages have been placed at high IDs, i.e. low priorities.

`DM at middle IDs`: the results are a combination of those for the previous two schemes. A significant number of short and long deadlines messages can be added. This is due to the fact that there are many free IDs both at high and low priorities. Only the medium priority IDs are already used, which leads to decreased extensibility for medium deadlines (10, 20, 50 ms). Overall, this ID-assignment scheme dominates the previous two for the SAE benchmark.

`DM evenly spaced IDs`: here we see the positive effect of increasing the ID-spacing between messages (i.e. leaving some IDs unused, so that new messages can be added there later on). Overall, we can add a large number of messages for all deadlines. However, for long deadlines (500, 1000 ms) the number that can be added saturates at 466 messages.

`Random`: here we see the effects of poor schedulability caused by using an ID-assignment scheme that does not take into account message deadlines. There is low extensibility for messages with short and long deadlines, and reduced extensibility for messages with medium deadlines.

`By ECU`: we observe that with this scheme extensibility is similar to that of `Random`. This can be explained as follows. Although ID-assignment inside each band is DM, the assignment of ECUs to bands is random, thus the overall ID-assignment again does not reflect the timing constraints.

It is clear from these observations that the proposed metric is capable of assessing the extensibility of a given CAN configuration. It highlights both *where* (i.e. for what mes-

sage periods and deadlines) and *by how much* a CAN configuration can be extended. This provides useful insight for engineers developing CAN-based systems. Further, it is sensitive to both the relative priority ordering and the actual ID assignment. By contrast state-of-the-art metrics such as breakdown utilisation, robustness, and sensitivity analysis are only sensitive to differences in relative priority ordering. Hence, we claim that the proposed metric is better suited to assessing the extensibility of CAN configurations.

5. EXTENSIBLE ID-ASSIGNMENT

In the previous section we provided a metric for assessing the extensibility of a CAN configuration. By applying it to the SAE benchmark, we gained insight into which of the example ID-assignment schemes are good or bad in terms of extensibility. However, in order to devise an ID-assignment scheme which maximizes extensibility, we first need to fully understand what the *limiting factors* on extensibility are.

5.1 Limiting Factors on Extensibility

Clearly, an empty CAN configuration, which contains no messages at all, is the most extensible. Table 4 gives the maximum number of messages that can be added to an empty CAN configuration for different baud rates. Here, we can add a few short deadline messages, some medium deadline messages, and many long deadline messages. We also see that for higher baud rates, we could theoretically add more long-deadline messages than there are IDs available (indicated by an asterisk). This is clearly a limiting factor for extensibility when using the standard ID format (i.e. 11 bit IDs).

D=T [ms]	125k	250k	500k	1M
1	0	1	3	6
2	1	3	6	14
5	4	8	18	36
10	8	18	36	73
20	18	36	73	147
50	45	92	184	369
100	92	184	369	740
200	184	369	740	1481
500	462	925	1851	3702*
1000	925	1851	3702*	7404*
$\sum$	1739	3487	6982	13972
overfit	0.86	1.72	3.44	6.86

Table 4: Maximum number of 8-Byte messages that can be added to an empty configuration (11 bit IDs)

The key limiting factor is however schedulability, which is critically dependent on maintaining an effective, ideally an optimal, priority ordering. From the example schemes studied in section 4.3, it is clear that extensibility is maximised when there is sufficient space between the existing message IDs to insert all of the new messages with a particular deadline while maintaining an effective priority order (e.g. DJMPO). Thus `DM at lowest IDs` has high extensibility for messages with long deadlines (200, 500, 1000 ms), while conversely `DM at highest IDs` has high extensibility for messages with short deadlines (1, 2, 5 ms). Finally `DM even spacing` has good extensibility for messages with medium deadlines (10, 20, 50 ms).

In summary, we have identified three limiting factors: Firstly, for messages with short deadlines, the main limiting factor is schedulability, and thus maintaining an effective priority ordering is essential. Secondly, for messages with long deadlines the main limiting factor is the number of available IDs. Finally, extensibility may be limited by the number of available IDs between messages with different deadlines; once this space runs out, effective priority ordering can no longer be maintained and schedulability suffers.

5.2 Policy for Extensible ID-Assignment

Knowing the maximum number of messages with a particular deadline that can be scheduled, and how ID-spacing impacts extensibility, we can derive a set of rules for designing an ID-assignment policy that maximizes extensibility:

- The relative priority ordering of the messages must follow a policy which reflects deadline requirements.
- Messages with the same deadline can be grouped together, and thus be inside an *ID-band*. The relative priority ordering of messages with the same deadline is of little or no consequence.
- The width of the ID-bands can be derived from the maximum number of messages with the corresponding deadline (and period) which can possibly exist in a schedulable system. Thus the width of the ID-bands increases with increasing deadline.

Based on these rules, we derive a simple, yet powerful ID-assignment algorithm. We call it DWB, for Deadline-Width-Band (i.e. DJMPO, increasing Width, ID-Bands). It features a set of ID-bands into which messages are assigned according to their deadlines. The pseudo code for the DWB algorithm is given in Algorithm 2.

The DWB algorithm generates ID-assignments which are optimized for future extensibility. It also provides an upgrade pathway, since it can be applied to generate an initial ID-assignment from scratch (i.e. when none of the messages has been assigned an ID yet) and it can also be applied to make repeated upgrades. Every time the systems is upgraded, the algorithm assign IDs to the new messages, preserving an appropriate priority order (i.e. DJMPO) without changing the IDs of the old messages. Hence the algorithm is applicable over the entire lifespan of a CAN-based system.

Algorithm 2: DWB ID-Assignment

```
   Input: oldMessages          /* messages with fixed IDs */
   Input: newMessages          /* messages without ID yet */
1  /** Phase 1: ID-Bands **/;
2  setup ID-bands according to DM and increasing width;
3  /** Phase 2: ID-Assignment **/;
4  for each message in newMessages do
5  |  choose ID-band according to message's deadline;
6  |  assign smallest free ID inside ID-band to message
7  end
   Output: ID-Assignment
```

5.3 Setting the Width of the ID-Bands

The DWB algorithm is based on a set of ID-bands, each dedicated to a specified deadline. The width of these ID-bands impacts extensibility. Hence a key requirement is to find the optimal widths for the ID-band widths. Based on the insight we gained concerning the limiting factors (see Table 4), we set the ID-band widths as follows: For a baud rate of 125Kbits/sec we can set the widths according to the maximum number of messages for each deadline without utilizing all IDs. For higher baud rates, this is no longer possible, since the total number of messages exceeds the number of available IDs. This may not be a major issue, since it is very unlikely that a CAN system could host all these messages at the same time. However, since we cannot predict which messages will be added in the future, it is beneficial

to have all ID-band widths set as close as possible to these numbers. Thus, we have to find a suitable trade-off. We consider three different approaches for this problem:

`scaled` – One way to address this trade-off is to scale the width of the ID-bands according to the total number of messages and the available IDs (i.e. 2032). This way, all ID-bands are narrowed by the same factor.

$$\text{width}_{\text{ID-band}} = \text{messages per D} \cdot \frac{2032}{\sum \text{all messages}}$$

`greedy` – We start to set the ID-band widths according to the maximum number of messages (starting from short deadlines). We do this for each deadline. Once we reach the 2032 limit, we end the current ID-band there. In a sense, this approach simply cuts off the long-deadline band(s). Messages which have a deadline whose ID-band is cut off are simply placed at the highest available IDs (i.e. lowest priorities).

`adjusted` – It is unlikely that a network will ever host the maximum number (i.e. 1000+) of long-deadline messages (500, 1000 ms); however, it may well host most of the short- to medium-deadline messages. This is mainly rooted in the dynamics of automotive systems where most data needs to be sent more frequently than 1000 ms [14]. The `adjusted` approach uses a simple, and somewhat arbitrary, method to give short- and medium-deadline ID-bands their maximum width, while compensating by narrowing the long-deadline ID-bands. First we determine the ID-band with deadline X and maximum width W that has the longest deadline such that if we also assigned all ID-bands with longer deadlines a width of W then there would be sufficient IDs available. Next we assign all ID-bands up to and including X their maximum width, and allocate an initial width of W to the remaining bands with longer deadlines. We then compute the total number of IDs left over. These left over IDs are allocated to the bands with longer deadlines than X in proportion to the value of $log(Y - X)$ where Y is the deadline of the band.

Based on these policies, we set the ID-band widths as shown in Table 5. All policies follow the *increasing ID-band width* concept.

D=T [ms]	max.	scaled	greedy	adjusted
1	1	1	1	1
2	3	2	3	3
5	8	5	8	8
10	18	10	18	18
20	36	21	36	36
50	92	53	92	92
100	184	107	184	184
200	369	215	369	369
500	925	539	925	581
1000	1851	1078	396	740
$\sum$	3487	2032	2032	2032

Table 5: Width of ID-Bands for 250Kbit/sec

We note that an alternative approach to narrowing the long-deadline ID-bands could be to use statistical information about existing systems. For example, it may be considered extremely unlikely that more than 500 messages could be added with any particular deadline, hence that number could serve as a valid upper limit on the width of any band.

5.4 Performance of Extensible ID-Assignment

In order to evaluate the performance of the DWB ID-assignment policy (and its three options for ID-band widths), we applied it to the SAE benchmark. The IDs are listed in Table 6. Note message `m06`, although it has a 5 ms deadline, is placed inside the 10ms-band with the `scaled` option. This is because the 5ms-band can only accommodate 5 messages, but the system has 6 of these messages. This highlights a shortcoming of the `scaled` option.

Message	DWB (scaled)	DWB (greedy)	DWB (adjusted)
m01	3	4	4
m02	4	5	5
m03	5	6	6
m04	6	7	7
m05	7	8	8
m06	8*	9	9
m07	9	12	12
m08	10	13	13
m09	11	14	14
m10	12	15	15
m11	39	66	66
m12	93	158	158
m13	94	159	159
m14	95	160	160
m15	954	1636	1292
m16	955	1637	1293
m17	956	1638	1294

Table 6: ID-Assignment for the SAE benchmark

By assessing the extensibility and comparing against the previously best-performing policies (i.e. `DM at middle IDs` and `DM evenly spaced IDs`) we can see by how much our proposed policy and its options concerning the ID band widths improve extensibility. Figure 2 shows these results. Here, we can see that the DWB ID-assignment policy significantly improves extensibility for a wide range of periods and deadlines. For the `scaled` option we notice poor performance for messages with a 5 ms deadline. This is caused by the fact that the 5ms-band is set too narrow, as mentioned above. Both the `greedy` and the `adjusted` option perform equally well, and they outperform all other policies. They have high extensibility across all periods and deadlines.

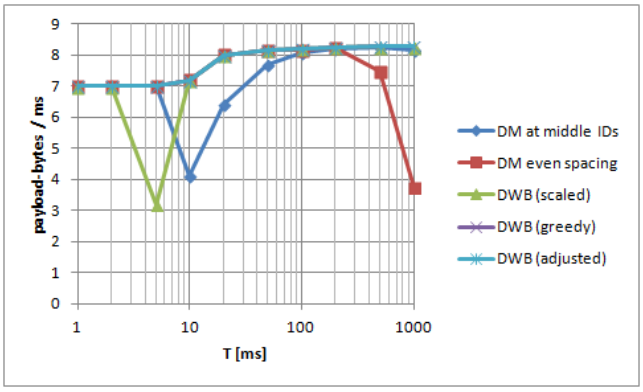

Figure 2: Extensibility of DWB ID-Assignment

5.5 Handling 29 bit IDs

The CAN protocol can handle both 11 and 29 bit IDs. While 11 bits provide only 2032 unique IDs, 29 bits provide a vast 532 676 608 unique IDs. The only downside is that the message transmission time is increased by up to 25 bits (18 extra ID bits plus additional stuff bits) for the longer IDs, which leads to a less efficient payload-to-overhead ratio. Table 7 shows the maximum number of messages which can be scheduled when using 29 bit IDs.

D=T [ms]	125k	250k	500k	1M
1	0	1	2	5
2	1	2	5	11
5	3	7	15	30
10	7	15	30	61
20	15	30	61	124
50	38	77	155	311
100	77	155	311	624
200	155	311	624	1249
500	390	780	1561	3124
1000	780	1561	3124	6251
$\sum$	1466	2939	5888	11790

Table 7: Maximum number of 8-byte messages with 29 bit IDs

Due to the longer IDs, there are significantly more unique IDs than schedulable messages, hence all ID-band widths can be set to the maximum number of messages which are schedulable with that deadline. Based on Table 7 we conclude that only 14 of the 29 bits are needed to uniquely identify the messages. The remaining 15 bits could be used to encode additional information, such as sender ECU, or for message filtering purposes.

6. MULTIPLE SYSTEM UPGRADES

In the previous section we presented the DWB ID assignment policy and showed that it maximizes extensibility. The DWB ID-assignment policy offers an *upgrade path*. We can apply it to build the initial configuration, and then repeatedly to perform system upgrades, adding a new set of messages each time. Note that on each upgrade, the IDs of existing, previously assigned messages remain unchanged. This ensures backwards compatibility, and removes a number of verification and logistical burdens.

In this section we demonstrate how the extensible ID-assignment policy performs for such system upgrades. We apply two consecutive upgrades to the SAE benchmark. Each upgrade imposes an additional 16% utilization on the CAN bus, thus moving the system from 44% to 60% and then to 76% utilization. Table 8 gives details of the messages that were added on each upgrade. These messages were randomly generated. The message periods were randomly chosen from the set of specified periods, and the deadlines set equal to them. The payload of each message was randomly chosen from 1 to 8 bytes.

Message	Format	Size [byte]	T [ms]	D [ms]
m18	Standard	1	5	5
m19	Standard	2	10	10
m20	Standard	4	20	20
m21	Standard	8	100	100
m22	Standard	7	200	200
m23	Standard	5	20	20
m24	Standard	2	10	10
m25	Standard	1	2	2
m26	Standard	2	50	50
m27	Standard	3	100	100
m28	Standard	1	50	50
m29	Standard	1	20	20
m30	Standard	4	500	500
m31	Standard	2	200	200

Table 8: System-Upgrade Scenarios

The evaluation was performed in three consecutive steps:

- `initial` – The initial ID-assignment was built using the DWB algorithm. This provides maximum extensibility for the subsequent upgrades. (Note we used the *adjusted* option, see Table 5 for the ID-band widths, and Table 6 for the message IDs).
- `initial + upgrade #1` – Upgrade #1 is applied to the initial system, with IDs assigned to the new messages. Here we compared two policies: the DWB algorithm and the Robust Priority Assignment Algorithm (Algorithm 3 from [9]).
- `initial + upgrade #1 + upgrade #2` – Upgrade #2 is applied to the already upgraded system, with IDs assigned to the new messages. Again, we applied the two policies discussed above.

Table 9 shows the IDs that were assigned to the upgrade messages. Figure 3 shows the extensibility metric for the resulting CAN configurations.

Message	Robust	DWB	Message	Robust	DWB
m18	11	10	m25	3	1
m19	1290	16	m29	2022	32
m24	1291	17	m26	2023	67
m20	2028	30	m28	2024	68
m23	2029	31	m27	2025	162
m21	2030	161	m31	2026	343
m22	2031	342	m30	2027	711

Table 9: ID-Assignment for system upgrades

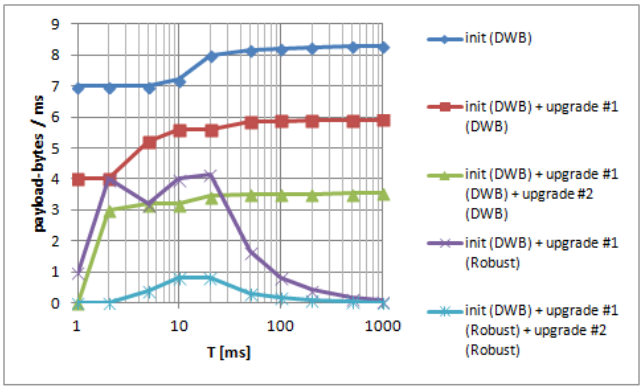

Figure 3: Extensibility of Configuration when performing System-Upgrades

We observe from Figure 3 that the upgrade path using the extensible ID-assignment policy (i.e. the DWB Algorithm) performs very well. Although extensibility drops, since we are adding messages, it stays at relatively high levels, and is almost even across all deadlines.

In contrast, the upgrade pathway using the Robust Priority Assignment Algorithm performs poorly, with extensibility dropping significantly. This is caused by the fact that the medium-deadline messages are placed at relatively low priorities during upgrade #1. This limits and decreases extensibility, and hence makes upgrade #2 challenging. Again, the messages are placed at low priorities leaving very little in terms of future extensibility.

Since we are comparing against Robust Priority Assignment, it is also interesting to see which message is the limiting factor in terms of schedulability. For that purpose we apply robustness analysis: We analyse how much additional interference (in terms of bit times) the system can handle before it becomes unschedulable [7]. The results are shown in Table 10. For both upgrade pathways the same message is the limiting factor, although IDs have been assigned in a

completely different manner. However, this is not surprising, since these messages have the shortest deadlines in their respective configurations: m06 has a 5 ms deadline, upgrade #1 then introduces m18 which also has a 5 ms deadline, and finally upgrade #2 adds m25 which has a 2 ms deadline.

System	Interference	Message
init	715	m06
init + upgrade #1 (extensible)	630	m18
init + upgrade #1 (robust)	630	m18
init + #1 + #2 (extensible)	300	m25
init + #1 + #2 (robust)	300	m25

Table 10: System Upgrades: Robustness Metric

Based on these comparisons we see that applying the extensible ID-assignment policy (DWB Algorithm) proposed in this paper, results in CAN configurations that are both *extensible and robust*. This enables multiple system upgrades to be performed consecutively without the need to re-assign any existing (previously assigned) message IDs. However, if we apply Robust Priority Assignment then the resulting CAN configuration is *robust, but hardly extensible*. It becomes much less likely that we can upgrade the system several times in a row.

While the evaluation results can only provide data on robustness for the case study considered, the extensible ID-assignment policy is in fact a *robust* policy, providing ID assignments that tolerate the maximum additional interference. This stems from the fact that the DWB Algorithm preserves, at each step, Deadline minus Jitter Monotonic Priority Ordering, which has been proven to also be robust [9] in the cases where it is optimal.

7. OPTIMALITY OF ID-ASSIGNMENT

In this section we consider to what extent the DWB algorithm is optimal in terms of extensibility. Note as with the rest of the work in this paper, the scope of the problem we are interested in comprises building an initial configuration and then extending it via a sequence of upgrades. At each step no specific information is assumed to be available about the set of messages in the subsequent upgrade steps, and the message IDs need to be assigned and fixed at each step.

We define *optimal extensibility* as follows: Let $UP = (I^0, U^1, U^2, U^3, ...)$ be an arbitrary upgrade path consisting of a sequence of message sets corresponding to an initial message set I^0 and those added by subsequent upgrades $(U^1, U^2, U^3, ...)$.

Definition: An ID-assignment algorithm X provides *optimal extensibility* if it is able to assign message IDs and obtain a schedulable system at every step in some upgrade path UP, for every upgrade path UP where there exists an ID-assignment algorithm that makes UP schedulable at every step.

In determining optimality it is useful to consider the behaviour of a hypothetical *clairvoyant* ID-assignment algorithm. A clairvoyant algorithm would have full information about the upgrade path and hence be able to work back from the final set of messages. If that final set of messages is feasible, i.e. can be scheduled using optimal priority assignment [1, 2] then the ID-assignment trivially follows, since the previous stages of the upgrade path involve subsets of the final message set and are hence schedulable using the same IDs.

The above discussion provides a simple way to determine optimality with respect to extensibility, we can make comparisons against a policy that is permitted to re-assign message IDs that were previously fixed. Since schedulability depends only on the relative priority ordering of the messages rather than the actual IDs assigned, such a flexible approach effectively equates to the behaviour of a clairvoyant algorithm for the problem where message IDs are fixed after initial configuration, and again after each upgrade.

We further define *weak optimality* with respect to extensibility as follows.

Definition: An ID-assignment algorithm provides *weakly optimal extensibility* if optimality holds conditional on the following constraints: (i) The added messages have constrained deadlines that equate to the deadlines specified for the ID-bands used. (ii) The number of messages with each deadline does not exceed the width of the corresponding ID-band.

THEOREM 1. *The extensible ID-assignment policy implemented by the DWB algorithm provides* weakly optimal extensibility *compared to using DJMPO with freely assignable message IDs.*

PROOF. Proof follows from the fact that providing constraints (i) and (ii) hold, then the priority ordering of the messages produced by the DWB algorithm is at every stage maintained in accordance with DJMPO, hence schedulability is the same as it is when message IDs are freely assigned using DJMPO. □

If all messages are of the maximum length, then it is *not* possible to have more messages of a particular deadline than the maximum width of the corresponding ID-band without the system becoming unschedulable, even when IDs can be freely assigned. This is the case because the maximum width of each ID-band is computed in such a way that it can accommodate the largest number of maximum length messages with that deadline that it is possible to schedule, regardless of the message periods[6], and assuming there are no other messages in the system. Hence (ii) is only a constraint for ID-bands that have been reduced in size.

We note that if the system designer knows that the system will evolve by adding many messages that have payloads less than the maximum (8 data bytes) then they may choose to compute the size of the bands accordingly. In this paper, we assume that as the system evolves, where possible, good use will be made of any spare capacity in existing messages and thus accounting only for maximum length messages is a reasonable compromise. In practice it may be appropriate to allow for shorter messages in the shortest deadline ID-bands, since individual ECUs need to send disparate messages. The width of these ID-bands could instead be set on the basis of the maximum number of short (i.e. 1 data byte) messages that can be scheduled with that deadline and period.

8. DISCUSSION & CONCLUSIONS

In this paper we introduced an ID-assignment policy for CAN messages which maximizes extensibility for future upgrades. The policy is applied to derive an initial CAN configuration, and also used to perform system upgrades. A key aspect of the policy is *ID-bands with increasing width*; bands of IDs are reserved for messages with different deadlines, and these bands are wider, with more IDs available, for longer deadlines. We proved that the policy is weakly-optimal in the sense that, subject to some constraints, it provides the same extensibility as a clairvoyant policy which knows what the upgrades, i.e. sets of added messages, will be. It gives the same extensibility as freely assigning message IDs using

[6]Provided they are no smaller than the deadline.

Deadline minus Jitter Priority Order (DJMPO), since it ensures that relative priority ordering is maintained throughout. Further, since the policy maintains DJMPO it makes no compromises in terms of robustness; the ability to tolerate additional interference e.g. from errors on the bus.

In addition we introduced a metric and assessment method aimed at determining the extensibility of a CAN configuration. This metric provides insight into *where* (i.e. at which periods and deadlines) and *by how much* a CAN configuration can be extended. Our evaluation based on a simple automotive benchmark shows that the ID-assignment policy is effective at maximizing extensibility, and that it performs well for consecutive system upgrades.

The extensible ID-assignment policy presented in this paper needs to be applied starting from an empty configuration. Due to the use of legacy components, beginning a clean sheet design is rare in the context of automotive systems. An open problem therefore remains: How to extend (upgrade) an existing CAN configuration with arbitrary fixed message IDs to maintain maximum extensibility for further upgrades. This problem is the subject of our ongoing work. We note that the extensibility-assessment metric presented in this paper is applicable in that case.

In a technical report [18], we provide additional discussion about how the approach proposed in this paper may be extended, in particular considering systems with *release jitter* [12], messages with *offset* release times [22], and applicability of the approach to CAN-FD which employs a *flexible data-rate* [4].

9. ACKNOWLEDGEMENTS

The authors would like to acknowledge the financial support of the "COMET K2 - Competence Centres for Excellent Technologies Programme" of the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT), the Austrian Federal Ministry of Economy, Family and Youth (BMWFJ), the Austrian Research Promotion Agency (FFG), the Province of Styria and the Styrian Business Promotion Agency (SFG), the ARTEMIS Joint Undertaking project EMC^2 (grant agreement nb 621429), the EPSRC project MCC (EP/K011626/1) and the INRIA International Chair program. EPSRC Research Data Management: No new primary data was created during this study.

10. REFERENCES

[1] N.C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical Report YCS 164, Department of Computer Science, University of York, 1991.

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

[3] Bosch. CAN specification 2.0, 1991.

[4] Bosch. CAN with flexible data-rate specification 1.0, 2012.

[5] R. I. Davis, L. Cucu-Grosjean, M. Bertogna, and A. Burns. A review of priority assignment in real-time systems. *Journal of Systems Architecture*, 65(C):64–82, 2016.

[6] R.I. Davis and A. Burns. Robust priority assignment for fixed priority real-time systems. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 3–14, 2007.

[7] R.I. Davis and A. Burns. Robust priority assignment for messages on Controller Area Network (CAN). *Real-Time Systems*, 41(2):152–180, 2009.

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

[9] R.I. Davis, A. Burns, V. Pollex, and F. Slomka. On priority assignment for controller area network when some message identifiers are fixed. In *International Conference on Real-Time Networks and Systems (RTNS)*, pages 279–288, 2015.

[10] R.I. 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(1):73–116, 2013.

[11] R.I. Davis and N. Navet. Controller Area Network (CAN) schedulability analysis for messages with arbitrary deadlines in FIFO and work-conserving queues. In *IEEE International Workshop on Factory Communication Systems (WFCS)*, pages 33–42, 2012.

[12] R.I. Davis and N. Navet. Traffic shaping to reduce jitter in Controller Area Network (CAN). In *Work-in-Progress Session of Euromicro Conference on Real-Time Systems (ECRTS)*, 2012.

[13] P. Emberson and I. Bate. Stressing search with scenarios for flexible solutions to real-time task allocation problems. *IEEE Transaction on Software Engineering*, 36(5):704–718, 2010.

[14] S. Kramer, D. Ziegenbein, and A. Hamann. Real world automotive benchmarks for free. In *International Workshop on Analysis and Methodologies for Embedded and Real-Time Systems (WATERS)*, 2015.

[15] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *IEEE Real Time Systems Symposium (RTSS)*, pages 166–171, 1989.

[16] F. Pölzlbauer, I. Bate, and E. Brenner. Optimized frame packing for embedded systems. *IEEE Embedded Systems Letters*, 4(3):65–68, 2012.

[17] F. Pölzlbauer, I. Bate, and E. Brenner. On extensible networks for embedded systems. In *IEEE International Conference and Workshops on the Engineering of Computer Based Systems (ECBS)*, pages 69–77, 2013.

[18] F. Pölzlbauer, R.I. Davis, and I. Bate. A practical message ID assignment policy for Controller Area Network that maximizes extensibility. Technical Report YCS-2016-504, Department of Computer Science, University of York, 2016.

[19] S. Punnekkat, R.I. Davis, and A. Burns. Sensitivity analysis of real-time task sets. In *Advances in Computing Science – ASIAN'97*, volume 1345 of *Lecture Notes in Computer Science*, pages 72–82. 1997.

[20] K. W. Schmidt. Robust priority assignments for extending existing controller area network applications. *IEEE Transactions on Industrial Informatics*, 10(1):578–585, 2014.

[21] K. Tindell and A. Burns. Guaranteeing message latencies on controller area network (CAN). In *International CAN Conference*, 1994.

[22] P. M. Yomsi, D. Bertrand, N. Navet, and R. I. Davis. Controller Area Network (CAN): Response time analysis with offsets. In *IEEE International Workshop on Factory Communication Systems (WFCS)*, pages 43–52, 2012.

[23] H. Zeng, M. Di Natale, P. Giusto, and A. Sangiovanni-Vincentelli. Stochastic analysis of can-based real-time automotive systems. *IEEE Transactions on Industrial Informatics*, 5(4):388–401, 2009.

[24] W. Zheng, J. Chong, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli. Extensible and scalable time triggered scheduling. In *International Conference on Application of Concurrency to System Design (ACSD)*, pages 132–141, 2005.

[25] Q. Zhu, Y. Yang, E. Scholte, M. Di Natale, and A. Sangiovanni-Vincentelli. Optimizing extensibility in hard real-time distributed systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 275–284, 2008.

[26] Qi Zhu, Yang Yang, M. Natale, E. Scholte, and A. Sangiovanni-Vincentelli. Optimizing the software architecture for extensibility in hard real-time distributed systems. *IEEE Transactions on Industrial Informatics*, 6(4):621–636, 2010.

[27] A. Zuhily and A. Burns. Optimality of (D-J)-monotonic priority assignment. *Information Processing Letters*, 103(6), 2007.

APPENDIX

A. MULTIPLE SYSTEM UPGRADES: A LARGER CASE STUDY

In order to evaluate how the proposed approach scales, we have performed a larger case study. The system is based on one described in [23]. It features a system that contains 69 messages, which are sent by 6 ECUs. The CAN bus is operated at 500 kbit/s, which results in a CAN utilization of 60%.

ECU	Messages	Utilization [%]
1	18	9.28
2	15	15.12
3	18	19.29
4	7	5.03
5	6	3.51
6	5	7.94
$\sum$	69	60.25

Table 11: Case Study

Based on this system, we have performed the following evaluation: First we build an *initial configuration*. For this we pick the messages which are sent by ECUs 1, 5, and 6. This leads to an initial utilization of 20%. Then we perform *upgrade #1* by adding the messages which are sent by ECUs 2 and 4. This moves the utilization to 40%. Finally we perform *upgrade #2* by adding the messages which are sent by ECU 3. This moves the utilization to 60%. The messages are specified in table 12, 13 and 14 respectively.

Message	Format	Size [byte]	T [ms]	D [ms]	ECU
m05	Standard	4	10	10	1
m07	Standard	8	100	100	1
m08	Standard	2	100	100	1
m09	Standard	3	100	100	1
m11	Standard	2	100	100	1
m12	Standard	4	20	20	1
m14	Standard	4	100	100	1
m16	Standard	6	10	10	1
m29	Standard	3	25	25	1
m35	Standard	7	25	25	1
m50	Standard	8	100	100	1
m53	Standard	4	100	100	1
m61	Standard	8	100	100	1
m62	Standard	1	100	100	1
m64	Standard	8	100	100	1
m65	Standard	1	100	100	1
m66	Standard	1	100	100	1
m68	Standard	1	100	100	1
m10	Standard	8	25	25	5
m15	Standard	8	100	100	5
m27	Standard	8	25	25	5
m39	Standard	8	50	50	5
m44	Standard	8	100	100	5
m56	Standard	8	100	100	5
m06	Standard	8	10	10	6
m21	Standard	8	10	10	6
m22	Standard	8	25	25	6
m36	Standard	8	25	25	6
m42	Standard	8	50	50	6

Table 12: Case Study – Initial System

The evaluation is performed as follows:

- `initial` – The initial ID-assignment was built using the DWB algorithm. This provides maximum extensibility for the subsequent upgrades. (Note we use the *adjusted* option.)
- `initial + upgrade #1` – Upgrade #1 is applied to the initial system, with IDs assigned to the new messages. Here we compared two policies: the DWB algorithm and the Robust Priority Assignment Algorithm (Algorithm 3 from [9]).
- `initial + upgrade #1 + upgrade #2` – Upgrade #2 is applied to the already upgraded system, with IDs assigned to the new messages. Again, we applied the two policies discussed above.

Message	Format	Size [byte]	T [ms]	D [ms]	ECU
m01	Standard	8	10	10	2
m02	Standard	8	10	10	2
m13	Standard	4	100	100	2
m17	Standard	7	100	100	2
m18	Standard	8	100	100	2
m19	Standard	7	50	50	2
m26	Standard	8	20	20	2
m28	Standard	8	20	20	2
m30	Standard	5	10	10	2
m31	Standard	8	20	20	2
m43	Standard	8	50	50	2
m45	Standard	2	25	25	2
m46	Standard	4	50	50	2
m47	Standard	4	50	50	2
m67	Standard	8	50	50	2
m32	Standard	8	10	10	4
m37	Standard	8	50	50	4
m38	Standard	8	50	50	4
m41	Standard	8	50	50	4
m48	Standard	8	100	100	4
m60	Standard	3	100	100	4
m69	Standard	8	100	100	4

Table 13: Case Study – Upgrade #1

Message	Format	Size [byte]	T [ms]	D [ms]	ECU
m03	Standard	4	5	5	3
m04	Standard	7	10	10	3
m20	Standard	8	10	10	3
m23	Standard	6	25	25	3
m24	Standard	6	25	25	3
m25	Standard	7	25	25	3
m33	Standard	8	10	10	3
m34	Standard	8	10	10	3
m40	Standard	5	50	50	3
m49	Standard	8	100	100	3
m51	Standard	1	100	100	3
m52	Standard	8	100	100	3
m54	Standard	1	100	100	3
m55	Standard	1	100	100	3
m57	Standard	7	100	100	3
m58	Standard	1	100	100	3
m59	Standard	1	100	100	3
m63	Standard	4	100	100	3

Table 14: Case Study – Upgrade #2

The IDs which have been assigned during each step are given in table 15, 16 and 17 respectively. Note that the DWB policy puts the 25 ms messages into the 20 ms band. This is done on purpose. We could have introduced a new 25 ms band, but decided not to. Instead we use the bands already derived from [14]. This makes for a fair comparison with the SAE benchmark.

Figure 4 shows the extensibility of the individual configurations along the upgrade pathway. When applying the

Message	ID (DWB)	Message	ID (DWB)	Message	ID (DWB)
m05	27	m50	325	m27	67
m07	320	m53	326	m39	136
m08	321	m61	327	m44	334
m09	322	m62	328	m56	335
m11	323	m64	329	m06	29
m12	63	m65	330	m21	30
m14	324	m66	331	m22	68
m16	28	m68	332	m36	69
m29	64	m10	66	m42	137
m35	65	m15	333		

Table 15: ID-Assignment for Initial System

Message	ID (DWB)	ID (RPA)	Message	ID (DWB)	ID (RPA)
m01	31	23	m45	73	2017
m02	32	24	m46	140	2020
m13	336	2026	m47	141	2021
m17	337	2027	m67	142	2022
m18	338	2028	m32	34	26
m19	138	2018	m37	143	2023
m26	70	2014	m38	144	2024
m28	71	2015	m41	145	2025
m30	33	25	m48	339	2029
m31	72	2016	m60	340	2030
m43	139	2019	m69	341	2031

Table 16: ID-Assignment for Upgrade #1

Message	ID (DWB)	ID (RPA)	Message	ID (DWB)	ID (RPA)
m03	9	22	m49	342	2005
m04	35	316	m51	343	2006
m20	36	317	m52	344	2007
m23	74	2001	m54	345	2008
m24	75	2002	m55	346	2009
m25	76	2003	m57	347	2010
m33	37	318	m58	348	2011
m34	38	319	m59	349	2012
m40	146	2004	m63	350	2013

Table 17: ID-Assignment for Upgrade #2

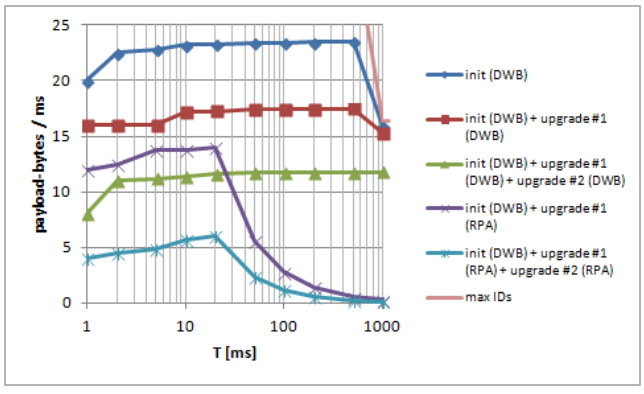

Figure 4: Extensibility of Configuration when performing System-Upgrades

DWB algorithm, the extensibility remains at high levels, and is relatively even across all deadlines. Note that the reduced extensibility at 1000 ms is caused by the fact that we run out of free IDs. This is indicated by the `max IDs` line, which is a hard upper-bound for extensibility. When applying *Robust Priority Assignment (RPA)* for the upgrade steps, we see that the extensibility is significantly reduced. In addition, extensibility is no longer even cross all deadlines, but drops significantly for longer deadlines.

Table 18 shows the robustness (the maximum additional interference tolerated in terms of bit times) for the individual configurations along the upgrade pathways.

System	Interference	Message
init	4385	m21
init + upgrade #1 (extensible)	3875	m32
init + upgrade #1 (robust)	3875	m21
init + #1 + #2 (extensible)	2270	m03
init + #1 + #2 (robust)	2040	m31

Table 18: System Upgrade: Robustness Metric

In summary, the results are very similar to those seen for the SAE benchmark, and show that the DWB approach scales well with respect to the size of the message set.

B. DISCUSSION

In this section, we discuss extension of the approach proposed in this paper to systems with release jitter, offset release times, and using CAN-FD. Further research is needed to investigate the effectiveness of the proposed approach in each of these cases.

B.1 Release Jitter

In our analysis of message ID-assignment, we have deliberately assumed that the release jitter of each message is zero. This simplifies both the explanations and the experiments. The concept can however be easily extended to messages with release jitter. What is important in terms of priority assignment is the *transmission deadline* E_i of each message, where $E_i = D_i - J_i$. Messages should therefore be allocated to ID-bands on the basis of their transmission deadlines. When release jitter is small compared to message deadlines, as is the case for most messages in real systems, then transmission deadlines form bands of similar values, reflecting their common deadlines and periods, thus all of the messages with similar transmission deadlines can be assigned to the same band. The width of the ID bands can be computed assuming messages with zero jitter as this allows for additional messages to be added that have zero (or minimal) jitter. We note that this potential under-estimation of the release jitter has the effect of making the ID-bands slightly wider than they may need to be, which is a conservative over-approximation. Messages gatewayed from another network can however have large jitter compared with their deadlines. These messages need to be allocated an ID band based on their transmission deadline, which may be quite different from their overall deadline. This may be problematic, since the transmission deadline of such a message may not necessarily fit well with the deadlines associated with the available ID-bands. We note; however, that methods exist to eliminate jitter due to gateways [12], which may go some way to alleviate this problem.

B.2 Offsets

Many automotive systems based on CAN use *offset* release times for messages sent by the same ECU [22], thus

avoiding the peak loads on the bus inherent in synchronous release. For systems with offsets, DJMPO is no longer optimal; however, the concepts developed in this paper can still be applied as a heuristic.

B.3 CAN-FD

This paper focuses on systems complying with the original CAN standard [3]. In some automotive systems, this is being superseded by CAN-FD which has a *flexible data-rate* [4]. With CAN-FD, the data part of each message can be up to 64 bytes and may be sent at a higher baud rate than the arbitration field and other parts of the message. With CAN-FD, it is no longer reasonable to assume that all messages may have the same maximum length. When there are large variations in the maximum transmission time of different messages, then it follows that DJMPO is no longer an optimal priority ordering. Nevertheless, the concepts developed in this paper can still be applied, with messages allocated to ID-bands reflecting their deadlines (or transmission deadlines). The width of those bands may be determined based on the transmission time of *short* messages, thus ensuring that the bands are sufficiently wide.

C. ADJUSTING WIDTHS OF ID-BANDS

For baud rates higher than 125 kbit/s the number of schedulable messages exceeds the number of available IDs. Thus, the widths of the ID-bands need to be adapted. We have proposed three methods to do so: linear `scaled`, `greedy`, and `adjusted`. Tables 19, 20, and 21 show the resulting ID-band widths.

D=T [ms]	max.	scaled	greedy	adjusted
1	1	1	1	1
2	3	2	3	3
5	8	5	8	8
10	18	10	18	18
20	36	21	36	36
50	92	53	92	92
100	184	107	184	184
200	369	215	369	369
500	925	539	925	581*
1000	1851	1078	396	740*
$\sum$	3487	2032	2032	2032

Table 19: ID-Band Widths for 250 kbit/s

D=T [ms]	max.	scaled	greedy	adjusted
1	3	1	3	3
2	6	2	6	6
5	18	5	18	18
10	36	11	36	36
20	73	21	73	73
50	184	54	184	184
100	369	107	369	276*
200	740	215	740	367*
500	1851	539	603	489*
1000	3702	1077	–	580*
$\sum$	6982	2032	2032	2032

Table 20: ID-Band Widths for 500 kbit/s

D=T [ms]	max.	scaled	greedy	adjusted
1	6	1	6	6
2	14	2	14	14
5	36	5	36	36
10	73	11	73	73
20	147	21	147	147
50	369	54	369	225*
100	740	107	740	284*
200	1481	215	647	344*
500	3702	539	–	422*
1000	7404	1077	–	481*
$\sum$	13972	2032	2032	2032

Table 21: ID-Band Widths for 1 Mbit/s

The `'-'` indicates that an ID-band is non-existing. We see: the higher the baudrate, the more ID-bands need to be cut-off when using the `greedy` approach.

The `adjusted` approach circumvents this problem by narrowing long deadline ID-bands. These are marks with `'*'`. We think that this approach is more balanced and more fair.