On Priority Assignment for Controller Area Network when some Message Identifiers are Fixed

Robert I. Davis[1,2], Alan Burns[1], Victor Pollex[3], Frank Slomka[3],
[1]Real-Time Systems Research Group, Department of Computer Science, University of York, York, UK.
[2]AOSTE Team, INRIA Paris-Rocquencourt, France.
[3]Institute of Embedded Systems / Real-Time Systems, Ulm University, Germany
{rob.davis, alan.burns}@york.ac.uk, {victor.pollex, frank.slomka}@uni-ulm.de

ABSTRACT

Controller Area Network (CAN) is widely used in automotive applications. With CAN, the network utilisation that may be obtained while ensuring that all messages meet their deadlines is strongly dependent on the policy used for priority (message identifier) assignment. This paper addresses the problem of priority assignment when some message identifiers are fixed. There are two variants of this problem: P1 where the gaps between fixed identifiers are large enough to accommodate the freely assignable messages and P2 when the gaps are too small. For problem P1, we provide algorithms that give optimal and robust priority orderings based on an adaptation of existing techniques. Problem P2 is more difficult to solve. We show via a counter example that the algorithms derived for P1 and others recently published can fail to find a schedulable priority ordering when the gaps are small, even though one exists. We derive an optimal and robust solution to this problem with respect to a simple form of schedulability analysis which assumes the same upper bound on the length of all messages.

1. INTRODUCTION

Controller Area Network (CAN) [3], [18] is a simple, efficient, and robust, broadcast communications bus for in-vehicle networks. Today, typical mainstream family cars contain 25-35 Electronic Control Units (ECUs), many of which communicate using 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 requires that nodes wait for a bus idle period before attempting to transmit. If two or more nodes attempt to transmit messages at the same time, then the node with the message with the lowest numeric identifier will win arbitration and continue to send its message. The other nodes will cease transmitting and wait until the bus becomes idle again before attempting to re-transmit their messages. (Full details of the CAN physical layer protocol are given in [3], with a summary in [6]). In effect CAN messages are sent according to fixed priority non-pre-emptive scheduling, with the message identifiers acting as priorities.

RTNS 2015, November 04-06, 2015, Lille, France
© 2015 ACM. ISBN 978-1-4503-3591-1/15/11…$15.00
DOI: http://dx.doi.org/10.1145/2834848.2834866

In the configuration of CAN, the assignment of priorities (allocation of message identifiers) is of great importance. An optimal assignment of priorities enables maximum use to be made of network bandwidth, while ensuring that the network remains schedulable i.e. all messages meet their time constraints or deadlines. In his keynote talk at ECRTS 2012 [4], Darren Buttle of ETAS remarked on the myth of CAN bus utilisation believed by many in industry: *"You cannot run CAN reliably at more than 35% utilisation[1]"*. This myth comes about because it is general practice to assign message identifiers (i.e. priorities) in an ad-hoc way reflecting the data content of the message, ECU supplier and other legacy issues.

In this paper, we revisit the problem of priority assignment for CAN for the case where some message identifiers (priorities) are fixed, and only a subset may be freely assigned. There are two variants of the problem, **P1** where all of the gaps between the fixed message identifiers are sufficiently large to accommodate all of the freely assignable messages, and **P2** where these gaps are too small. We provide an optimal and robust algorithm for problem **P1**, and show via a counter example that both this algorithm and a similar one recently proposed in [23] are not optimal for problem **P2**. Further, we derive an optimal and robust solution to this problem with respect to a simple form of schedulability analysis which assumes the same upper bound on the length of all messages (approximation **A1**).

In 1994, Tindell et al. [24], [25], [26] showed how research into fixed priority scheduling for single processor systems could be adapted and applied to the scheduling of messages on CAN. The analysis of Tindell et al. provided a method of calculating the maximum queuing delay and hence the worst-case response time of each message on the network. They also recognised that with fixed priority scheduling, an appropriate priority assignment policy is key to obtaining effective real-time performance. Tindell et al. [26] suggested that messages should be assigned priorities in Deadline minus Jitter Monotonic Priority Order (DJMPO) [29]. In 2007, Davis et al. [6] found and corrected significant flaws in the schedulability analysis given by Tindell et al. [24], [25], [26]. These flaws could potentially result in the original analysis providing guarantees for messages that could miss their deadlines during operation. Further, Davis et al. [6] showed that the DJMPO, claimed by Tindell et al. to be optimal for CAN, is not optimal with respect to exact tests; and that Audsley's Optimal Priority Assignment (OPA) algorithm [1], [2] is required in this case.

For the analysis in [6] to be applicable, it is necessary that all of the assumptions of the classical scheduling model are met. In particular, each CAN controller and device driver or communications stack must ensure that whenever message arbitration starts on the bus, the highest priority message queued

[1] Figure may vary but not significantly.

at that node is entered into arbitration. This behaviour is essential if message transmission is to take place as if there were a single global priority queue and for the analysis to be correct. There are however many ways in which the communications stack, device driver, or CAN controller hardware can be implemented that do not match these assumptions. Issues include: Non-abortable transmit buffers [19], [16]; delays in refilling a transmit buffer [15]; the use of FIFO queuing policies [11]; work-conserving but otherwise ill-defined queuing policies [10], for example those using multiple levels of queues, or internal CAN controller message arbitration based on transmit buffer number. In this paper, we assume that the middleware and device drivers have been carefully designed to meet the assumptions of the classical analysis [6], as has been done for example with the Volcano Target Package[2].

In 2007, Davis and Burns [7] introduced the concept of *robust priority ordering* able to tolerate the most additional interference of any schedulable ordering. This concept was extended to CAN [8], where it is important in providing priority assignments that best tolerate errors on the bus. In 2014 Schmidt [23] applied a variant of the Robust Priority Assignment (RPA) algorithm [8] to problem **P2**, claiming in Theorem 1 of [23] that it gives an optimal and robust priority assignment with respect to the sufficient schedulability test used. (In this paper, we refer to this test as **S1**, see (6) and (7) in Section 2).

The remainder of the paper is organised as follows. Section 2 summarises the system model, terminology and notation used. Section 3 recapitulates the exact and the sufficient schedulability tests given in [6]. In Sections 4 and 5 we present solutions to the problem of providing an optimal and robust priority assignment when some message identifiers are fixed. Section 6 concludes with a summary and directions for future work.

2. SYSTEM MODEL AND TERMINOLOGY

In this section we describe the system model and notation used to analyse the worst-case response times of CAN messages. The system model is as described in [6]. Note that here we give only a high level description necessary to understand the message scheduling behaviour of CAN. Readers interested in details of the CAN protocol are directed to section 2.1 of [6].

The system is assumed to comprise a number of nodes (microprocessors) connected to each other via a CAN bus. Each node is assumed to ensure that, at any given time when arbitration starts, the highest priority message queued at that node is entered into arbitration on the bus.

The system is assumed to contain a static set of hard real-time messages, each statically assigned to a single node. Each message m has a unique priority. We distinguish between the *logical priority* of a message, which describes only its place in the priority order in relation to other messages, and the *physical priority* or *ID* of a message, meaning the actual value used for its identifier. We note that a complete set of physical priorities fully defines a set of logical priorities, but the reverse is not the case, since there are many sets of physical priorities that can map to the same logical priority order. The priority assignment policies considered in this paper effectively determine *logical priority* orderings, with some scope remaining to set physical priorities, for example to make use of the least significant bits of the ID for message filtering. (Note when we refer to a message as having lowest physical priority, due to the way in which arbitration works on CAN, this equates to the highest numerical ID). For brevity, we often just use the word priority when referring to the logical priority of a message, but are explicit when referring to physical priorities or IDs.

As priority uniquely identifies each message, we overload m to mean either message m or priority m as appropriate. We use $hp(m)$ to denote the set of messages with priorities higher than m, and $lp(m)$ to denote those with priorities lower than m. Similarly, we use $hep(m)$ to denote the set of messages with priorities higher than or equal to m, and $lep(m)$ to denote those with priorities lower than or equal to m.

Each message m has a maximum transmission time of C_m (see [6] for details of how to compute the maximum transmission time, taking into account the number of data bytes and bit-stuffing). The event that triggers queuing of an instance of message m is assumed to occur with a minimum inter-arrival time of T_m, referred to as the message *period*. Each message m has a hard *deadline* D_m, corresponding to the maximum permitted time from occurrence of the initiating event to the end of successful transmission of the message, at which time the message data becomes available on the receiving nodes that require it. The deadline of each message is constrained, i.e. equal to, or less than its period. 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 *queuing jitter* of the message. The *worst-case response time* R_m of message m is defined as the maximum possible delay from the initiating event for an instance of that message, until it is received at the receiving nodes. A message is said to be *schedulable* if its worst-case response time is less than or equal to its deadline $(R_m \le D_m)$. A system is said to be schedulable if all of the messages in the system are schedulable. The *utilisation* U_m of a message m is given by its maximum transmission time divided by its period ($U_m = C_m / T_m$). The total utilisation U of a set of messages is the sum of their individual utilisations.

Definition 1: *Optimal Priority Assignment (OPA)*: A priority assignment policy P is referred to as *optimal* with respect to a schedulability test S and a given system model, if and only if there is no set of messages that are compliant with the model that are deemed schedulable by test S using another priority assignment policy, that are not also deemed schedulable according to test S using policy P.

The following definition of Robust Priority Assignment (from [8]) relies on the idea of a general additional interference function $E(\alpha, w, i)$, where α is a scaling factor, used to model variability in the amount of interference, w is the length of the time interval over which the interference occurs and i is the priority level affected by the interference. The function $E(\alpha, w, i)$ is required to be a monotonically non-decreasing function of its parameters. We assume that the function $E(\alpha, w, i)$ is due to errors on the bus that cause messages to be re-transmitted[3].

Definition 2: *Robust Priority Assignment (RPA)* [7], [8]: For a given system model and a given additional interference function $E(\alpha, w, i)$, a priority assignment policy P is referred to as *robust* with respect to a schedulability test S if there are no systems, compliant with the system model, that are both schedulable according to test S and can tolerate additional interference characterized by a scaling factor α using another priority

[2] http://www.mentor.com/products/vnd/in-vehicle_software/volcano_target_package/

[3] An example of a realistic additional interference function for CAN is given by (1) in Section 3 based on the delay due to error recovery.

assignment policy *Q* that are not also both schedulable according to test *S* and can tolerate additional interference characterized by the same or larger scaling factor using priority assignment policy *P*.

We note that the above definitions are applicable to both sufficient and exact schedulability tests. While optimality and robustness w.r.t. an exact test is desirable, for problems where no tractable optimal or robust priority assignment algorithm is known for an exact test, then there is the option of either using a simple heuristic priority assignment with an exact test, or using a priority assignment policy that is optimal w.r.t. an effective sufficient test.

3. SCHEDULABILITY ANALYSIS

In this section we recapitulate the exact and sufficient schedulability analysis for CAN presented by Davis et al. [6] including extensions to account for errors on the bus.

We consider a very general error model. We assume that the maximum number of errors present on the bus in some time interval *t* is given by an error arrival function $F(t)$. We assume no specific details about this function; save that it is a monotonic non-decreasing function of *t*, which is required in order for the fixed point iteration used in response time analysis to be valid. In practice, this is no restriction, since in any reasonable error model one cannot have fewer errors in a longer interval of time than in a shorter one.

The worst-case impact of a single bit error is to cause transmission of an additional 31 bits of error recovery overhead plus re-transmission of the affected message. Only errors affecting message *m* or higher priority messages can delay message *m* from being successfully transmitted[4]. The maximum additional delay caused by the error recovery mechanism is given by:

$$E_m(t) = \left(31\tau_{bit} + \max_{k \in hep(m)}(C_k)\right)F(t) \tag{1}$$

As shown in [6], the worst-case response time of a message *m* can be determined by examining the response time of all instances of message *m* that occur within a priority level-*m* busy period; assuming that message *m* and all higher priority messages are released with their maximum jitter at the start of the busy period, and then subsequently re-released as soon as possible. Further, immediately before the initial release of these messages, the longest message of lower priority than *m* begins transmission. B_m is the blocking factor at priority *m*, equivalent to the longest transmission time of any message of lower priority:

$$B_m = \max_{k \in lp(m)}(C_k) \tag{2}$$

In the following, we use the index variable *q* to represent an instance of message *m*. The first instance, released at the start of the busy period corresponds to $q = 0$. The longest time from the start of the busy period to instance *q* beginning successful transmission is given by the solution to the following fixed point equation:

$$w_m^{n+1}(q) = E_m(w_m^n + C_m) + B_m + qC_m + \sum_{\forall k \in hp(m)} \left\lceil \frac{w_m^n(q) + J_k + \tau_{bit}}{T_k} \right\rceil C_k \tag{3}$$

Note τ_{bit} is the time for one bit to be transmitted on the bus. The summation term represents *interference* from higher priority messages that can win arbitration over message *m* and so delay its transmission. Further, as errors can impact the transmission of message *m* itself, the time interval considered in calculating the error recovery overhead includes the transmission time of the final instance of message *m*. Iteration starts with a value of $w_m^0(q) = B_m + qC_m$, and ends when $w_m^{n+1}(q) = w_m^n(q)$, or when $J_m + w_m^{n+1}(q) - qT_m + C_m > D_m$ in which case the message is unschedulable. The response time of instance *q* is given by:

$$R_m(q) = J_m + w_m(q) - qT_m + C_m \tag{4}$$

and the worst-case response time of message *m* by:

$$R_m = \max_{q=0..Q_m-1}(R_m(q)) \tag{5}$$

where Q_m is the number of instances of message *m* in the priority level-*m* busy period (see [6] for details of how this value is computed). We refer to the exact schedulability test given by (5) as **E1**.

As shown in [6], when messages have constrained deadlines, an upper bound on the worst-case response time of message *m* may be found by computing the maximum queuing delay w_m using the following fixed point iteration, where the revised blocking term $\max(B_m, C_m)$ accounts for *push-through blocking* from previous instances of the same message (see [6] for the full derivation):

$$w_m^{n+1} = E_m(w_m^n + C_m) + \max(B_m, C_m) + \sum_{\forall k \in hp(m)} \left\lceil \frac{w_m^n + J_k + \tau_{bit}}{T_k} \right\rceil C_k \tag{6}$$

Here, iteration starts with a suitable initial value such as $w_m^0 = C_m$, and continues until either $w_m^{n+1} + J_m + C_m > D_m$ in which case the message is not schedulable, or $w_m^{n+1} = w_m^n$ in which case the message is schedulable and an upper bound on its worst-case response time is given by:

$$R_m = w_m^{n+1} + J_m + C_m \tag{7}$$

We refer to the sufficient test given by (7) as **S1**. Davis et al. [6] also gave a simpler sufficient test which replaces the term $\max(B_m, C_m)$ in (6) by B^{MAX}, where B^{MAX} is the longest transmission time for any message on the network. We refer to this sufficient test as **S2**.

We note that both **S1** and **S2** are used in commercial schedulability analysis tools, for example Mentor Graphics Volcano Network Architect[5] toolset, due to their ease of implementation, speed of operation, and extensibility. In systems with soft real-time diagnostic messages of the maximum length at the lowest priorities, then schedulability tests **S1** and **S2** are *equivalent* to the exact test **E1** for hard real-time messages. We note that other commercial tools such as RTaW[6] NetCAR-Analyzer and Symtavision SymTA/S[7] make use of exact tests.

Some of the results in this paper are derived using an approximation **A1** whereby all messages are considered to have a length equal to that of the longest message in the system i.e. the same maximum transmission time. (Note we do *not* require that all messages are actually of the same length). Since a shorter message takes less time to transmit, approximation **A1** leads to pessimism in the analysis when there are messages of different lengths; however, this is mitigated by the fact that it is common in automotive systems for a substantial majority of messages to be of the maximum possible length. This is because packing as many signals as possible into 8 data byte CAN messages reduces the

[4] An error could occur during the transmission of a lower priority message that blocks message *m*, but then that message would not be re-transmitted until after *m* had been sent.

[5] http://www.mentor.com/products/vnd/communication-management/vna/

[6] http://www.realtimeatwork.com/

[7] https://www.symtavision.com/products/network-timing/

relative overheads of the arbitration, control, and CRC fields. Schedulability tests **S1** and **S2** are equivalent under approximation **A1**, since the blocking term for all messages has the same value.

4. OPTIMAL PRIORITY ASSIGNMENT WHEN SOME PRIORITIES ARE FIXED

While in some CAN systems, for example those using Volcano Target Package, it is possible to perform post-deployment re-configuration of message identifiers, often systems are constructed where the identifiers of messages sent by certain nodes (ECUs) are fixed. In fact this may apply to the messages sent by all nodes, with only the identifiers of new messages available for configuration (i.e. priority assignment). This is a common occurrence since very rarely are there clean sheet new designs, thus legacy ECUs requiring fixed message identifiers are used in the construction of new systems.

Let us assume that there are two subsets of messages, those in set *M* have fixed identifiers which we may not change, while those in set *N* are new, and are yet to be assigned identifiers. We separate the priority assignment problem into two cases:

P1: The gaps between the identifiers of messages in the fixed subset are sufficiently large that all of the new messages could fit into any gap, including the gap above the highest priority (lowest numeric identifier) fixed message, and the one below the lowest priority (highest identifier) fixed message.

P2: Not all the gaps are large enough to accommodate all of the new messages.

For these problems, we define an optimal priority assignment algorithm as follows: A priority assignment algorithm is optimal if it can find a schedulable priority ordering, whenever such an ordering exists, subject to the following constraints:

- For cases **P1** and **P2**, the messages with fixed IDs are in the priority order specified by their IDs;
- For case **P2**, in addition the number of new messages that by virtue of their priorities must fit into a gap in the physical priorities between, above, or below messages with fixed IDs do not exceed the size of the gap (i.e. the number of available message IDs in that range).

Similarly, a robust priority ordering algorithm can be defined as follows: A priority assignment algorithm is defined as robust if it finds a schedulable priority ordering which tolerates the maximum additional interference of any such ordering, whenever a schedulable priority ordering exists, subject to the above constraints.

4.1 Problem P1: Fixed IDs and Large Gaps

We now show that variants of Audsley's OPA algorithm [1], [2] and the RPA algorithm [7], [8] are respectively optimal and robust for problem **P1** where the gaps between message IDs are large. The OPA algorithm is shown below (Algorithm 1), with the modifications to the standard algorithm in italics. The same form of modification applies in the RPA algorithm.

Proof that the OPA algorithm (Algorithm 1) is optimal for problem **P1** assuming schedulability test **E1**, **S1** or **S2** is easily obtained via the standard proof strategy used for such algorithms. This proof strategy starts from an initial schedulable priority ordering *Q*, which in this case must also be valid, i.e. have the messages with fixed IDs in the partial priority order defined by those IDs. The logical priorities of pairs of messages are then swapped to move the message that the OPA algorithm first selects down to the lowest priority, while showing that all messages remain schedulable and the priority ordering remains valid. (We note that since both the initial priority ordering *Q* and that produced by the OPA algorithm have the messages with fixed IDs in the same logical priority order, there is never any need to swap a pair of such messages, which would in any case result in an invalid assignment). The swapping process is then repeated for the second message selected by the OPA algorithm, and so on until the priority ordering is transformed into that generated by the OPA algorithm, with no loss of schedulability. Proof of robustness for the modified RPA algorithm follows a similar approach.

```
for (each logical priority level k, lowest first) {
    for (each message m selected from a set containing the
    unassigned new  messages in N and the message with the
    lowest physical priority (highest message ID) of those that
    are currently unassigned in M ){
        if m is schedulable at priority k with all unassigned
        messages assumed to have higher priorities) {
        assign m to priority k
        }
    }
    if (no messages are schedulable at priority k) {
        return unschedulable
    }
}
return schedulable
```

Algorithm 1: OPA with fixed message IDs and large gaps

Note that during the swapping process, it may be the case that all of the new messages have logical priorities between two of the messages with fixed IDs; however, by definition of problem **P1**, the gap between these fixed IDs must be large enough to accommodate all of the new messages, and so the message set remains schedulable and valid. This aspect of the proof sketch leads to a suspicion that the OPA and RPA algorithms may not be applicable to problem **P2**.

4.2 Problem P2: Fixed IDs and Small Gaps

With the interpretation that all messages are deemed unschedulable if the priority assignment is invalid, we can see that the OPA and RPA algorithms are not applicable to problem **P2**. This is the case because the Conditions required for compatibility with Audsley's OPA algorithm, given in section 4.1 of [9] are broken. For completeness, these conditions are given below modified to apply to messages rather than tasks.

Condition 1: The schedulability of a message *m*, may, according to test *S*, depend on any independent properties of other messages of higher priority, but not on any properties of those messages that depend on their relative priority ordering.

Condition 2: The schedulability of a message *m*, may, according to test *S*, depend on any independent properties of other messages of lower priority, but not on any properties of those messages that depend on their relative priority ordering.

Condition 3: When the priorities of any two messages with adjacent priorities are swapped, then the message being assigned the higher priority cannot become unschedulable according to test *S*, if it was previously schedulable at the lower priority. (As a corollary, the message being assigned the lower priority cannot become schedulable according to test *S*, if it was previously unschedulable at the higher priority).

When considering these Conditions for problems **P1** and **P2**, we may safely ignore any cases where the priorities of two messages with fixed IDs are swapped, since such an action can only turn an initially valid priority ordering into an invalid one. However, we must consider what happens when the logical priority of a new message is swapped with that of a message with a fixed ID, since this is essential in transforming one valid priority ordering onto another. For problem **P1**, as all of the gaps are large enough, all of the Conditions hold; however for problem **P2**, all three Conditions can be broken. This is because such a swap can overload one of the gaps between the messages with fixed IDs, making the priority assignment invalid and thus effectively causing the schedulability of a message to depend on the relative priority ordering of higher (Condition 1) or lower (Condition 2) priority messages, or a message to effectively become unschedulable, due to the assignment being invalid, when moved to a higher priority (Condition 3).

In [23] Schmidt applies a variant of the RPA algorithm to problem **P2** using schedulability test **S1**, claiming in Theorem 1 of [23] that it gives an optimal and robust priority assignment with respect to that test. (Note, for reasons of space, and due to their complexity, we do not recapitulate the three algorithms from [23] which are used to address problem **P2**. The interested reader is referred to that paper for a full description). We do; however, show via a simple counter example, that the claim of optimality with respect to schedulability test **S1** made in [23] does not hold.

Our counter example has just 4 messages, one of which has a fixed ID (of 2). We assume for the sake of simplicity that there are only 4 physical priorities available: 1 (highest), 2, 3, and 4 (lowest). The messages are shown in Table 1 below.

Table 1: Message Parameters

Message	C	T	D
M_A	125	1000	750
M_F	125	1000	350
M_B	125	1000	750
M_C	75	1000	1000

Note that the parameters are given in terms of bit transmission times. Thus M_C corresponds to a message with 2 data bytes, and M_A, M_B, and M_F to messages containing 7 data bytes, on a bus using 11-bit identifiers. M_F has a fixed ID of 2 and can therefore only ever have the second highest priority. The other messages may fit around the priority of M_F in any order. The problem is to find a schedulable priority order that can tolerate the most additional interference, modelled in this case simply as α bit transmission times i.e. $E(\alpha, w, i) = \alpha$ as done in [23].

Table 2: Computed values of α

	Message			
Priority	M_A	M_F	M_B	M_C
4	300	NS	300	**550**
3	**300**	NS	300	-
2	375	NS		-
1		-	-	-

Table 2 shows the computed values of α as we attempt to apply the modified version of the RPA algorithm using exact test **E1** to this problem. Assigning any of the messages to priority level 4 results in a worst-case response time of 450, and hence the values of α given in the first row of the table. As message M_C tolerates the largest value of α, then it is assigned priority 4. (Note NS means 'Not Schedulable').

At priority level 3, the worst-case response time of the remaining messages is again 450. Hence message M_A is assigned to that priority (or M_B, it makes no difference since their timing parameters are identical). At priority level 2, we must assign message M_F since the gap below it is now full; however, the message is unschedulable at that priority with message M_B (or M_A) at the highest priority, since it has a worst-case response time of 375 and a deadline of 350. Thus the variant of the RPA algorithm gives up and declares the set of messages unschedulable.

The alternative priority ordering M_C (highest), M_F, M_B, M_A (lowest) results in response times of 200, 325, 450, 450 and thus a schedulable system, with an overall tolerance to additional interference α of 25 (800, 25, 300, 300 for each message respectively). This priority ordering also meets the constraint that message M_F has a fixed ID of 2. It is the optimal and robust priority ordering; however, it is not found by our modified RPA algorithm, or by the variant of the RPA algorithm given in [23].

In the above example, we have used exact analysis (**E1**). We note that the same logic and conclusions still hold if instead we use the sufficient analysis **S1** assumed in [23] for the robust priority assignment algorithms described there. Using **S1**, the variant of the RPA algorithm given in Algorithm 2 of [23] finds that at priority level 4, messages M_A, M_B, and M_F have an upper bound response time of 575 and message M_C an upper bound response time of 525. Message M_C is therefore assigned since it tolerates a value of α of 475. At priority level 3, the upper bound response time of the remaining messages is 500 (compared to 450 with exact analysis) and hence message M_A or M_B is assigned to that priority. At priority level 2, message M_F is again found to be unschedulable, and the system declared unschedulable. By contrast, the alternative priority ordering M_C (highest), M_F, M_B, M_A (lowest) results in upper bound response times of 200, 325, 450, and 575 and hence a schedulable system.

We note that the above counter example is not meant to be representative of a real CAN system; rather, it is simplified as far as possible to illustrate the issues involved. The reader should; however, be in no doubt that the same problems apply to priority assignment in more complex systems. For example, if we assume that the full range of 11-bit identifiers are available, then we may add a further 11 messages of length 75 bits (with long periods and deadlines) at physical priorities which are the 5th to 15th highest. None of the 3 freely assignable messages in our example can then be placed at priorities below these messages, thus the problem effectively becomes the same as the one described for just 4 physical priority levels.

5. SOLUTIONS TO PROBLEM P2

In this section, we consider optimal and robust solutions to problem **P2**, using sufficient tests **S1** or **S2** with approximation **A1**. (Note the tests are equivalent in this case). We prove that if there is some priority ordering that is schedulable according to **S1** or **S2** then there exists a schedulable priority ordering that has all of the new messages in a relative priority order which is DJMPO with respect to each other, with the fixed messages with some interleaving between them. First, we introduce a Lemma that assists in the proof of the theorem.

Lemma 1: Let *i* and *j* be two priority levels in a priority order for a set of messages that is schedulable according to **S1** or **S2** under approximation **A1** with some additional interference function $E(\alpha, w, i)$ with a fixed value of α. Assume that *i* is of higher priority than *j*, and message *X* initially at priority *i* has a value of deadline minus jitter larger than that of message *Y*, which is initially at priority *j* (i.e. $D_X - J_X > D_Y - J_Y$). If the priorities of *X* and *Y* are swapped, so that *X* is at the lower priority *j*, and *Y* is at the higher priority *i*, and the priorities of all other messages are undisturbed, then all of the messages remain schedulable.

Proof: See Appendix A of the technical report [12] on which this paper is based. (We note approximation **A1** is required for the Lemma to hold).

Theorem 1: For problem **P2** with a mix of fixed and new messages compliant with our model, then a priority ordering with the new messages in deadline minus jitter relative priority order is optimal and robust with respect to sufficient tests **S1** and **S2** under approximation **A1** (same length assumed for all messages), *independent* of the additional interference function $E(\alpha, w, i)$ (and hence the error arrival function) provided only that $E(\alpha, w, i)$ is monotonically non-decreasing in its parameters.

Proof: We show that any mixed set of fixed and new messages compliant with the model that is schedulable under some priority order Q and that is the most robust i.e. tolerates the most additional interference (i.e. largest value of $\alpha = \sigma$) of any schedulable ordering is also schedulable with the same scaling factor σ (same robustness) under a priority ordering P that has the new messages in deadline minus jitter relative priority order.

Base case: The set of messages is schedulable with additional interference function $E(\sigma, w, i)$ assuming priority order Q_k since we set $Q_k = Q$ and Q is the schedulable priority ordering assumed in the theorem.

Inductive step: We select a pair of new messages that are at some priorities *i* and *j* in priority ordering Q_k, but out of deadline minus jitter relative priority order. Lemma 1 proves that irrespective of the details of the additional interference function $E(\sigma, w, i)$, we may swap the priorities of these two messages forming priority ordering Q_{k-1} without loss of schedulability. Note that since we directly swap these message priorities, there is no change to the number of new messages in any of the gaps between fixed messages, and thus any constraints due to the size of these gaps continue to hold. At most $k = v(v-1)/2$ steps (effectively a bubble sort of the *v* new messages) are required to transform priority ordering Q into *P* without any loss of schedulability or robustness □

Theorem 1 tells us that we can obtain both optimal and robust priority orderings by sorting the new messages into DJMPO and then merging (interleaving) them in some way with the fixed messages; however, it gives no information on how to do so. We examine that problem in the next two subsections.

Corollary 1: (From Theorem 1) DJMPO is optimal and robust with respect to sufficient tests **S1** and **S2** under approximation **A1** when all messages are freely assignable.

Corollary 1 appears similar to the claim made by Tindell et al.[24], [25] that DJMPO is optimal for the test given in those papers; however, it should be noted that the test given by Tindell et al. was flawed (neither sufficient, nor exact [6]) and that DJMPO has been shown to be not optimal for exact tests [6].

5.1 Optimal Priority Assignment for P2

Algorithm 2 provides Optimal Priority Assignment for problem **P2** with a mix of fixed and new messages, with respect to sufficient tests **S1** (and **S2**) under approximation **A1.** (Note approximation **A1** is required so that Lemma 1 and Theorem 1 hold and thus the new messages may be kept in DJMPO partial order).

Algorithm 2 first sorts the set of new messages *N* into deadline minus jitter relative priority order, and the set of fixed messages in order of their physical priorities and thus message IDs, lowest priority (i.e. highest message ID) first. The algorithm then attempts to assign message IDs starting with the highest possible ID (lowest priority) and working upwards through the physical priority levels (possible CAN IDs). Effectively the algorithm gives precedence to the fixed messages. It always fills low priority levels with new messages (in the reverse of DJMPO) until this is not possible because either the next new message in reverse DJMPO is not schedulable, or the ID reached corresponds to that of the highest ID (lowest priority) as yet unassigned fixed message. In which case, there is no option but to assign the fixed message. In these cases if the fixed message is unschedulable, then there is no schedulable priority assignment possible.

```
All messages in N and M are assumed to be unassigned
Sort the new messages in N into DJMPO
Sort the fixed messages in M by their IDs, highest first
NextID = Highest_CAN_ID()
while (there are unassigned messages) {
    m = unassigned message from M with the highest ID
    if (fixed ID of m is NextID) {
        if (m is schedulable with all unassigned messages
        assumed to have higher priorities) {
            assign message m with ID = NextID.
        }
        else {
            return unschedulable
        }
    }
    else {
        k = unassigned message from N with the largest value of
            deadline minus jitter
        if (k is schedulable with all unassigned messages
        assumed to have higher priorities) {
            assign message k with ID = NextID.
        }
        else if (m is schedulable with all unassigned messages
        assumed to have higher priorities) {
            NextID = ID of message m
            assign message m with ID = NextID.
        }
        else {
            return unschedulable
        }
    NextID = Next_higher_CAN_ID (NextID)
    }
}
return schedulable
```

Algorithm 2: OPA for problem P2 with fixed message IDs and small gaps

We note that at each step the algorithm only checks the unassigned new message with the largest value of deadline minus jitter. This is sufficient to achieve optimality since Theorem 1 tells us that if a schedulable ordering exists, then there will also be a schedulable ordering with the new messages in deadline minus jitter relative priority order.

We note that Algorithm 2 is similar to Algorithm 2 from [23] in that it seeks to place new messages at the lowest priorities (highest possible IDs) and hence fixed messages at the highest possible relative priorities. Algorithm 2 uses the helper functions Highest_CAN_ID() which returns the highest valid CAN ID (lowest physical priority), and Next_higher_CAN_ID (NextID) which takes a CAN ID (NextID) as input returns the next valid ID with a higher physical priority (smaller numerical value).

Theorem 2: Algorithm 2 provides optimal priority assignment for problem **P2** with a mix of fixed and new messages compliant with our model, with respect to sufficient tests **S1** and **S2** under approximation **A1**. (We assume an arbitrary but known error arrival function and hence an additional interference function that is monotonic in its other parameters and has a fixed value of α).

Proof: We prove the theorem by showing that any schedulable priority ordering *Q* may be transformed into the priority ordering *P* found by Algorithm 2 without loss of schedulability. First, by Theorem 1, we swap pairs of new messages thus transforming priority order Q into order Q_k with all new messages in deadline minus jitter monotonic partial order without loss of schedulability.

Base case: The set of messages is schedulable assuming priority order Q_k .

Inductive step: For $h = v$ down to 1, where v is the number of new messages. Select the hth highest priority new message and change its ID to the ID selected by Algorithm 2 forming priority order Q_{k-1} . Note this does not affect the relative priority order of the new messages; however, message h may now be at a lower priority than one or more fixed messages.

We note that by starting with the lowest priority new message and only ever moving messages down in priority, then at each step the constraints on the number of new messages in the gaps between, above and below fixed messages are respected. This follows from the fact that they are respected in the priority order produced by Algorithm 2.

To prove that priority order Q_{k-1} is schedulable, we need to consider the schedulability of the following groups of messages:

(i) Fixed messages in $hp(h, Q_k)$ i.e. those with higher priority than message h in Q_k , and fixed messages in $lp(h, Q_{k-1})$. Each of these messages is subject to exactly the same interference (has the same set of higher priority messages and the same set of lower priority messages) in both priority orders Q_k and Q_{k-1} , hence their response times are unchanged.

(ii) All new messages except for h. These messages did not change their relative priority order with respect to either h or the fixed messages, hence each of these messages is subject to exactly the same interference in both priority orders Q_k and Q_{k-1} , hence their response times are unchanged.

(iii) Message h is schedulable in priority order Q_{k-1} otherwise it would not have been placed in that position relative to the fixed messages by Algorithm 2.

(iv) Fixed messages in $lp(h, Q_k) \cap hp(h, Q_{k-1})$. These messages are schedulable when at a lower priority than h in Q_k . The only change in Q_{k-1} is that they are now at a higher priority than h, hence they also remain schedulable.

At most v steps where v is the number of new messages are required to transform priority ordering Q_k into $Q_1 = P$ without any loss of schedulability □

Algorithm 2 provides optimal priority assignment for problem **P2**; however, the priority ordering obtained gives precedence to messages with fixed IDs effectively forcing the set of new messages to the lowest relative priorities at which they are schedulable. This is problematic in that one or more of these messages may be on the brink of unschedulability and thus the ordering obtained is likely to be fragile rather than robust. (We note that the process of checking freely assignable messages at the lowest priorities is necessary in order to obtain an ordering that is optimal using a greedy and therefore tractable approach with complexity that is $O(n^2)$ in the number of schedulability tests).

5.2 Robust Priority Assignment for P2

In this subsection, we build on the previous results to provide a Robust Priority Assignment algorithm (Algorithm 3) for problem **P2** using the sufficient tests **S1** or **S2** under approximation **A1.**

From Theorem 1, we know that if any schedulable priority ordering exists then there is a robust ordering with all of the new messages in deadline minus jitter relative priority order. Further, Algorithm 2 provides us with an initial schedulable priority ordering if one exists, with the new messages in deadline minus jitter relative priority order and assigned the lowest possible priorities commensurate with schedulability. Intuitively, we need to move from the initial ordering provided by Algorithm 2 towards a robust ordering while preserving deadline minus jitter partial order among the new messages.

```
CurrentOrder = Priority order from Algorithm 2 with $\alpha = 0$.
**if** (CurrentOrder not valid) { // *no schedulable ordering*
    **return** unschedulable
}
RobustOrder = CurrentOrder
Determine max value of $\alpha$ for each message in CurrentOrder
MaxAlpha = min $\alpha$ over all messages in CurrentOrder
**while** (true) {
    $x$ = message with min $\alpha$ in CurrentOrder // *break ties in*
    // *favour of fixed messages, then lower priority messages*
    **if** ($x$ is a fixed message) {
        **return** schedulable
    }
    **if** (there is no fixed message at a higher priority than $x$ in
         CurrentOrder) {
        **return** schedulable
    }
    NewOrder = RippleUpwards($x$, CurrentOrder)
    **if** (NewOrder not valid) { // *i.e. cannot create a valid*
    // *NewOrder due to no space at highest physical priorities*
        **return** schedulable
    }
    Determine max value of $\alpha$ for each message in NewOrder
    NewMaxAlpha = min $\alpha$ over all messages in NewOrder
    **if** (NewMaxAlpha $\geq$ MaxAlpha) {
        RobustOrder = NewOrder
        MaxAlpha = NewMaxAlpha
    }
    CurrentOrder = NewOrder
}
```

Algorithm 3: RPA for problem P2 with fixed message IDs and small gaps

This is the approach taken in Algorithm 3, which first uses Algorithm 2 to determine if there is a schedulable priority ordering. The main body of Algorithm 3 is then a loop which promotes new messages upwards in the priority order with respect to fixed messages so as to improve overall robustness, while retaining their deadline minus jitter partial order. The helper function RippleUpwards(x, CurrentOrder) forms a NewOrder of messages from CurrentOrder by moving message x up in priority to the first ID above the next fixed message of higher priority than x that is not occupied by a fixed message. Note that ID may be occupied by another new message, if so then RippleUpwards() ripples the higher priority new messages upwards in priority as little as possible to preserve their deadline minus jitter partial order and to avoid ID clashes with fixed messages. If this is not possible, due to insufficient space at the highest physical priorities (lowest numerical CAN IDs) then the function returns invalid. A detailed walk-through of the operation of Algorithm 3 and the RippleUpwards() function is given in Appendix B of [12].

The fact that Algorithm 3 terminates can be seen by considering that on each iteration of the loop, the algorithm either exits or it moves at least one new message to a priority that is higher than that of the next higher priority fixed message. Further, once all new messages have higher priorities than fixed messages, the algorithm exits (if it did not already do so due to running out of message IDs at high priorities). Therefore in the worst case, with v new messages and n-v fixed messages, moving just one new message past one fixed message each time, then we have $v(n-v)$ iterations. This value is maximised when we have equal numbers of fixed and new messages, thus an upper bound on the number of iterations is given by $n^2/4$. On each iteration, we (somewhat pessimistically assume) that the value of α needs to be recomputed for each message. This may be done to a reasonable level of precision via a fixed number of iterations (e.g. 10) of a binary search. The number of single message schedulability tests required is therefore $O(n^3)$.

Theorem 3: Algorithm 3 provides robust and optimal priority assignment for problem **P2** with a mix of fixed and new messages compliant with our model, with respect to sufficient tests **S1** (and **S2**) under approximation **A1**.

Proof: Proof of optimality follows from Theorem 2 and the use of Algorithm 2. We now prove robustness. We assume for contradiction that there is some priority ordering Q^* that has the maximum robustness of $\alpha = \sigma$ of any valid priority ordering, and that robustness is strictly higher than that achieved by priority ordering *P* produced by Algorithm 3. Since Algorithm 2 is optimal we can apply that algorithm assuming an additional interference function with scaling factor $\alpha = \sigma$ to obtain a schedulable priority ordering *Q* with the same robustness as Q^*. We now show that the priority ordering *P* obtained by Algorithm 3 is equivalent to *Q* thus contradicting the hypothesis and proving the theorem.

Let P_k be the priority ordering produced on the *k*th iteration of Algorithm 3. (P_0 is the priority ordering produced by Algorithm 2 with $\alpha = 0$ used to initialise Algorithm 3). Let $hpN(X,m)$ be the set of new messages that have a higher priority than fixed message *m* in some priority order *X*. Similarly, let $hp(X,m)$ be the set of all messages with higher priority than message *m* in priority order *X*.

We note that in the following, all priority orders (*P*, P_k, P_0, *Q*) have the new messages in deadline minus jitter partial order (Theorem 1) and the fixed messages in the partial order determined by their fixed message IDs, thus the priority orders only differ in the interleaving of those two sets of messages.

We prove the theorem by showing that on each iteration *k* of Algorithm 3, then either the following condition (Condition 4) holds, in which case a further iteration occurs, or the priority ordering produced equates to *Q*.

Condition 4: *Q* has robustness greater than that of P_k, and for every fixed message *m*, the set of higher priority new messages in P_k is a subset of those in *Q* i.e. $hpN(P_k,m) \subseteq hpN(Q,m)$. Since the new and fixed messages are in the same partial orders in P_k and *Q* then it follows that $hp(P_k,m) \subseteq hp(Q,m)$. Further, for at least one fixed message, the relationship is a strict subset thus $hpN(P_k,m) \subset hpN(Q,m)$ and $hp(P_k,m) \subset hp(Q,m)$

We note that if for every fixed message *m*, the set of higher priority new messages is the same in both P_k and *Q* i.e. $hpN(P_k,m) = hpN(Q,m)$, then this implies that the two priority orders are equivalent and hence have the same robustness.

Base case: Condition 4 holds by construction for P_0 since that order is produced by Algorithm 2 with $\alpha = 0$, which places new messages at the lowest possible priorities assuming no additional interference. Alternatively, P_0 is equivalent to *Q*, in which case we already reached a contradiction.

Inductive case: From P_k to P_{k+1}. Since the overall robustness of *Q* is σ then it follows that all of the fixed messages in *Q* must tolerate additional interference equating to $\alpha \geq \sigma$. Since, by the induction hypothesis Condition 4 holds for P_k then since $hp(P_k,m) \subseteq hp(Q,m)$ all fixed messages in P_k must also tolerate additional interference equating to $\alpha \geq \sigma$. As P_k has lower overall robustness than *Q* there must be one or more new messages in P_k that tolerate additional interference equating to some value of $\alpha < \sigma$. Let message *j* be such a message with the least value of α of any message in P_k. This is the message that Algorithm 3 selects to move up in priority.

Since the overall robustness of *Q* is greater than that of P_k, it must be the case that message *j* is at a higher priority in *Q* than the first (i.e. lowest priority) fixed message *h* with a higher priority than message *j* in P_k i.e. where $h \in hp(P_k, j)$. This follows because the new messages are in the same partial order in *Q* and P_k as are the fixed messages, thus if *j* were at a lower priority than *h* in *Q* then we would have $hp(P_k, j) \subseteq hp(Q, j)$ and hence by virtue of the additional interference tolerated by message *j* in P_k, the robustness of *Q* would also be $< \sigma$.

Priority order *Q* attests that there is a schedulable and valid priority ordering with message *j* at a higher priority than message *h*. It follows that the process of moving message *j* to a priority above message *h* and rippling up the priorities of the new messages with smaller values of deadline minus jitter than *j as little as possible* to maintain the deadline minus jitter partial ordering must result in a schedulable priority order P_{k+1} (The schedulability of priority order *Q* attests that this change cannot result in message *h* or any of the other higher priority fixed messages becoming unschedulable, and since the new messages have their priorities increased, they cannot become unschedulable either). Further, since these messages are moved up in priority as little as possible to accommodate message *j* at higher priority than *h*, then either Condition 4 continues to hold or P_{k+1} and *Q* are equivalent. If P_{k+1} and Q are equivalent then we again reached a contradiction.

Iterating over increasing *k*, we note that either P_{k+1} and *Q* become equivalent and so a contradiction is reached, or Condition 4 continues to hold, and there exists a new message *j* that Algorithm 3 can select to move to a higher priority above some fixed message *h*, and this move is valid (i.e. there is space at the lower numerical message IDs (higher physical priorities) to maintain the partial ordering of new messages). Since we know that Algorithm 3 terminates after a maximum of $v(n-v)$ iterations, where *v* is the number of new messages, this again results in a contradiction. Iteration cannot go on indefinitely, and therefore the final priority ordering *P* produced by Algorithm 3 must equate to *Q*. This contradicts the hypothesis that priority order *Q* and hence *Q**is strictly more robust than *P* □

The RPA algorithm (Algorithm 3) gives preference to messages with fixed IDs in that if there are multiple orderings with the same robustness, it will pick the one with the lowest possible priorities for the freely assignable messages. Of course if the system can be made more robust by a priority ordering with some of those messages at higher priorities, then the algorithm will find such an ordering.

We note that although Algorithm 2 and Algorithm 3 assign message IDs to the new messages, effectively they produce a logical priority ordering. The system designer is free to adjust the IDs of new messages, and this will not affect schedulability, provided the logical priority ordering is unchanged. Such adjustments may be useful to set bits for message filtering, or to leave gaps for subsequent additions to the set of messages.

5.3 Automotive Case Study

We examined the performance of the OPA and RPA algorithms for problem **P2** on a real automotive case study, first presented by Kollman et al. in [17]. This system comprises a CAN bus connecting 10 ECUs, with 85 messages sent over the network. The number of messages sent by each ECU is given by the annotations in Figure 1. All messages are sent periodically and share a common release time. The length of the messages varies from 1 to 8 data bytes. (There are 60 messages with 8 bytes of data, 3 with 5 bytes, 9 with 4 bytes, 10 with 2 bytes and 3 with 1 byte). The original bus speed intended for the network was 500 kBit/s.

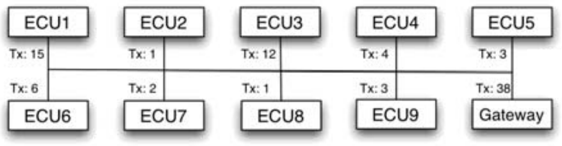

Figure 1: Case study network architecture.

Table 3. provides some preliminary results, comparing different schedulability tests and priority assignments. In Expts 1-3 all of the messages were assumed to be new and hence their priorities freely assignable, while Expt 4 assumes the original priority order specified by the automotive supplier. The minimum schedulable bus speed and breakdown utilisation for the exact test **E1** with Optimal Priority Assignment (OPA) [6] are 275.8 Kbit/s and 84.7% respectively, whereas for the sufficient test **S2** with DJMPO [29], they are 276.3 Kbit/ and 84.5%, applying approximation **A1**, these values become 302.5 Kbit/ and 77.2%. All of which compare favourably with values of 750.9 kBit/s and 31.1% using the priority assignment originally specified.

Table 3: Case Study: Summary of Results

Expt	Schedulability test	Priority order	Min bus speed	Max bus util.
1	Exact E1	OPA	275.8 Kbit/s	84.7%
2	Sufficient S2	DJMPO	276.3 Kbit/s	84.5%
3	Sufficient S2 & A1	DJMPO	302.5 Kbit/s	77.2%
4	Exact E1	Specified	750.9 Kbit/s	31.1%

To evaluate the performance of the OPA and RPA algorithms on problem **P2**, we assumed a 500 kBit/s bus, that messages sent by ECUs 1 and 6 were fixed, and that messages sent by other ECUs were freely assignable. First, we determined an optimal priority order using Algorithm 2. This is illustrated in Figure 2.

The transmission deadlines of the new messages are shown in black and those of the fixed messages in red. As expected, the new messages are placed at the lowest possible priorities. For this priority order, we computed the maximum additional interference that could be tolerated by all messages according to a simple additional interference function: $E(\alpha, w, i) = \alpha$ bit times. This resulted in a maximum tolerance of $\alpha = 863$ bit times, equivalent to the re-transmission and error recovery overhead for up to 5 maximum length messages. The message response times are shown in Figure 2 with and without this additional interference. Using this priority assignment, the minimum schedulable bus speed is 408.6 kBit/s and the max. bus utilisation 57.1% (computed using test **E1** or **S1**, both of which gave the same result).

We then used the RPA algorithm (Algorithm 3) to determine a robust priority ordering assuming the same simple additional interference function. The results are illustrated in Figure 3. Notice that now some of the new messages are interleaved with the fixed messages. The robust priority ordering obtained tolerated a maximum additional interference of 1268 bit times equating to the re-transmission and error recovery overhead of up to 7 maximum length messages. Again, the message response times are shown with and without this additional interference. Using this priority assignment, the minimum schedulable bus speed is 378.8 kBit/s, and the max. bus utilisation 61.7% (computed using test **E1** or **S1**, both of which again gave the same result).

We note that comparison with a robust and optimal priority assignment produced using an exact test is not possible since no solutions are currently known which are tractable. A brute force approach would potentially need to test $n!/(n-v)!$ combinations (where n is the number of messages, and v is the number of new messages). For our case study, this amounts to more than a googol (10^{100}) combinations.

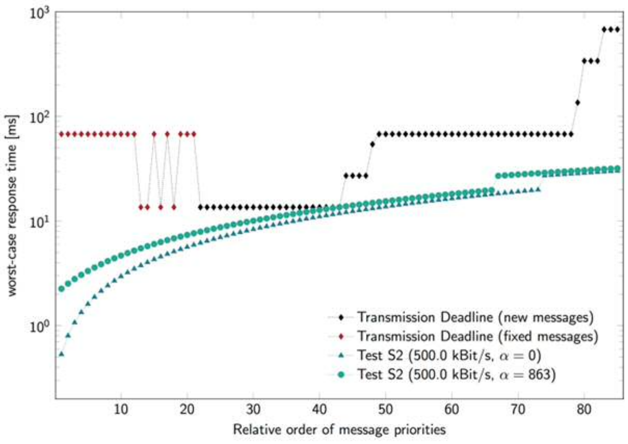

Figure 2: Priority assignment using OPA algorithm for P2.

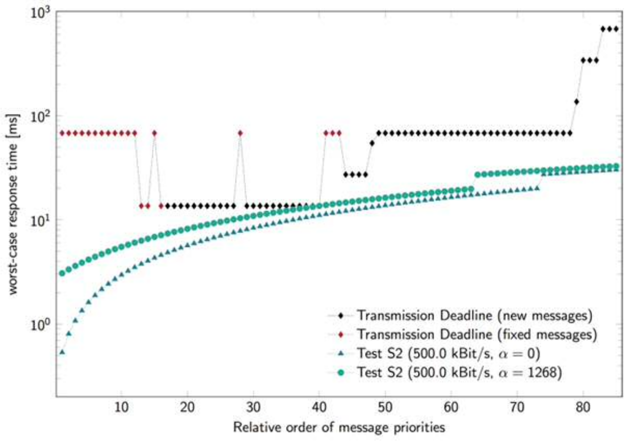

Figure 3: Priority assignment using RPA algorithm for P2.

6. SUMMARY AND CONCLUSIONS

In this paper, we investigated the problem of priority assignment for CAN when some messages have fixed identifiers and only a subset may have their priorities (identifiers) modified.

We showed that there are two flavours of this problem **P1** and **P2**. In **P1**, the gaps between the identifiers of the fixed messages are large enough to potentially accommodate all of the messages whose priorities may be freely set. In **P2** these gaps are insufficient. The main contributions of this paper are as follows: We proved that problem **P1** using exact **E1**, or sufficient tests **S1** or **S2**, may be solved optimally with respect to those tests using variants of the standard OPA [1], [2] and RPA [7], [8] algorithms. We proved by means of a counter example, that problem **P2** is *not* amenable to the same form of solution. This shows that a recent algorithm proposed for this problem by Schmidt [23] using sufficient test **S1** is not optimal w.r.t. that test as was suggested. We derived an optimal and robust solution to problem **P2** with respect to a simple form of schedulability analysis which uses sufficient test **S1** and assumes the same upper bound on the length of all messages (approximation **A1)**. Finding such solutions for more precise analysis, i.e. for **S1** without this approximation and also for exact test **E1**, remains an interesting open problem.

Problems **P1** and **P2** are strongly motivated by industrial practice. Nearly all new automotive systems are built using at least some legacy components. Thus unless message identifiers can be reprogrammed (as with the Volcano implementation), then there will be some ECUs that transmit CAN messages with fixed IDs that cannot be changed as part of the design. Further, once a system has been built using new ECUs, what were once messages with freely assignable IDs then become fixed, thus upgrades and extensions to the system suffer the same or worse problems of priority assignment; further strengthening the case for a solution that allows message IDs to be reprogrammed.

When message IDs cannot be reprogrammed, then the choice of which IDs to assign to new messages added for a particular upgrade has an effect on the future priority assignments possible and hence the schedulability and robustness of the system on future upgrades. In this case, design for extensibility [21], [27] for example by leaving appropriate gaps between message IDs, becomes an additional concern. This problem merits further research.

While the research detailed in this paper focuses on priority assignment for messages on a single CAN bus, many automotive systems make use of two or more such networks connected together via gateways. In these systems, timing requirements are specified as end-to-end deadlines on functionality implemented by tasks that communicate over the networks. Here, a simple approach can be taken, splitting the end-to-end deadlines into sub-deadlines on individual messages, thus dividing the larger problem into a set of smaller independent ones (one for each CAN bus). This enables the priority assignment policies discussed in this paper to be used. Such subdivision can however lead to sub-optimal solutions. An alternative is to use search techniques to solve the overall problem holistically [22], [28].

7. ACKNOWLEDGMENTS

This work was funded in part by 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.

8. REFERENCES

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

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

[3] Bosch. 1991. CAN Specification version 2.0. Robert Bosch GmbH, Postfach 30 02 40, D-70442 Stuttgart.

[4] Buttle, D. 2012. Real-Time in the Prime Time *Keynote talk at Euromicro Conference on Real-Time Systems*. Presentation: http://ecrts.eit.uni-kl.de/index.php?id=69 .

[5] Casparsson, L., Rajnak, A., Tindell, K., and Malmberg, P. 1998. Volcano - a revolution in on-board communications. *Volvo Technology Report, 1998/1*.

[6] Davis, R.I., Burns, A., Bril, R.J., and Lukkien, J.J. 2007. Controller Area Network (CAN) Schedulability Analysis: Refuted, Revisited and Revised. *Real-Time Systems*, Vol. 35, No. 3, pp. 239-272.

[7] Davis, R.I., and Burns, A. 2007. Robust Priority Assignment for Fixed Priority Real-Time Systems. *In proceedings Real-Time Systems Symposium (RTSS)*, pp. 3-14. 2007.

[8] Davis, R.I., and Burns, A. 2009.Robust priority assignment for messages on Controller Area Network (CAN). *Real-Time Systems*, Vol. 41, No. 2, pp. 152-180.

[9] Davis, R.I., and Burns, A. 2011. Improved Priority Assignment for Global Fixed Priority Pre-emptive Scheduling in Multiprocessor Real-Time Systems. *Real-Time Systems*, Vol. 47, No. 1, pp.1-40.

[10] Davis, R.I., and Navet, N., 2012. Controller Area Network (CAN) Schedulability Analysis for Messages with Arbitrary Deadlines in FIFO and Work-Conserving Queues. *In proceedings Workshop on Factory Communication Systems (WFCS)*. pp. 33-42.

[11] Davis, R.I., Kollmann, S., Pollex, V., and Slomka, F. 2013. Schedulability Analysis for Controller Area Network (CAN) with FIFO Queues Priority Queues and Gateways. *Real-Time Systems*, Vol. 49, No. 1, pp. 73-116.

[12] Davis, R.I., Burns, A.,Pollex, V., and Slomka, F. 2015. On Priority Assignment for Controller Area Network when some Message Identifiers are Fixed. *Technical Report YCS-2015-498* Dept. of Computer Science, University of York. https://www.cs.york.ac.uk/ftpdir/reports/2015/YCS/498/YCS-2015-498.pdf

[13] Ferreira, J., Oliveira, A., Fonseca, P., and Fonseca, J.A. 2004. An Experiment to Assess Bit Error Rate in CAN. *In Proceedings International Workshop of Real-Time Networks (RTN)*, pp. 15-18.

[14] Hansson, H., Nolte, T., Norstrom, C., and Punnekkat, S. 2002. Integrating Reliability and Timing Analysis of CAN-based Systems. *IEEE Transaction on Industrial Electronics*, 49(6): 1240-1250.

[15] Khan, D.A., Bril, R.J., and Navet, N. 2010. Integrating hardware limitations in CAN schedulability analysis, *In proceedings Workshop on Factory Communication Systems (WFCS),* pp.207-210.

[16] Khan, D.A., Davis, R.I., and Navet, N. 2011. Schedulability Analysis of CAN with Non-abortable Transmission Requests. *In proceedings Emerging Technologies and Factory Automation (ETFA)*.

[17] Kollmann, S., Pollex, V., Kempf, K., Slomka, F., Traub, M., Bone, T.,and Becker, J. 2010. Comparative Application of Real-Time Verification Methods to an Automotive Architecture. *In Proceedings Real Time and Network Systems (RTNS)*

[18] ISO 11898-1. 1993. Road Vehicles – interchange of digital information – controller area network (CAN) for high-speed communication, *ISO Standard-11898, International Standards Organisation*.

[19] Di Natale, M. 2006. Evaluating message transmission times in Controller Area Networks without buffer preemption. *In proceedings Brazilian Workshop on Real-Time Systems*.

[20] Nolte, T., 2006 Share-driven scheduling of embedded networks, *PhD Thesis*, Malardalen University Press.

[21] Polzlbauer, F.; Bate, I.; Brenner, E., 2013.On Extensible Networks for Embedded Systems *In proceedings Workshops on the Engineering of Computer Based Systems (ECBS),* pp.69,77.

[22] Richard, M., Richard, P., and Cottet, F. 2001. "Task and message priority assignment in automotive systems." *In proceedings IFAC Conference on Fieldbus Systems and their Applications*, pp. 105-112.

[23] Schmidt, K.W., 2013. Robust Priority Assignments for Extending Existing Controller Area Network Applications. *IEEE Transactions on Industrial Informatics*, Vol.10, No.1, pp.578,585.

[24] Tindell, K.W., and Burns, A. 1994. Guaranteeing message latencies on Controller Area Network (CAN), *In Proceedings International CAN Conference*, pp. 1-11.

[25] Tindell, K.W., Burns, A., and Wellings, A.J. 1995. Calculating Controller Area Network (CAN) message response times. *Control Engineering Practice*, 3(8): 1163-1169.

[26] Tindell, K.W., Hansson, H., and Wellings, A.J. 1994. Analysing real-time communications: Controller Area Network (CAN). *In proceedings Real-Time Systems Symposium (RTSS)*, pp. 259-263.

[27] Zhu, Q., Yang, Y., Scholte, E., Di Natale, M., and Sangiovanni-Vincentelli, A. 2009. “Optimizing Extensibility in Hard Real-Time Distributed Systems”. In *Proceedings Real-Time and Embedded Technology and Applications* Symposium *(RTAS), pp* 275-284.

[28] Zhu, Q., Zeng, H., Zheng, W., Di Natale, M., Sangiovanni-Vincentelli, A. “Optimization of task allocation and priority assignment in hard real-time distributed systems”. *ACM Trans. Embed. Comput. Syst.* 11, 4, Article 85 (January 2013).

[29] Zuhily, A., and Burns, A. 2007. Optimality of (D-J)-monotonic priority assignment. *Information Processing Letters*, Vol. 103 No. 6.