Analysis and Optimization of Message Acceptance Filter Configurations for Controller Area Network (CAN)

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

Robert I. Davis
University of York, UK
& Inria, Paris, France
rob.davis@york.ac.uk

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

ABSTRACT

Many of the processors used in automotive Electronic Control Units (ECUs) are resource constrained due to the cost pressures of volume production; they have relatively low clock speeds and limited memory. Controller Area Network (CAN) is used to connect the various ECUs; however, the broadcast nature of CAN means that every message transmitted on the network can potentially cause additional processing load on the receiving nodes, whether the message is relevant to that ECU or not. Hardware filters can reduce or even eliminate this unnecessary load by filtering out messages that are not needed by the ECU. Filtering is done on the message IDs which are primarily used to identify the contents of the message and its priority. In this paper, we consider the problem of selecting filter configurations to minimize the load due to undesired messages. We show that the general problem is NP-complete. We therefore propose and evaluate an approach based on Simulated Annealing. We show that this approach finds near-optimal filter configurations for the interesting case where there are more desired messages than available filters.

CCS CONCEPTS

• **Networks** → **Network resources allocation**; **Network design and planning algorithms**; *Network performance analysis*; • **Computer systems organization** → **Real-time system specification**;

KEYWORDS

controller area network (CAN), message acceptance filter, optimization, simulated annealing, automotive, real-time

ACM Reference format:
Florian Pölzlbauer, Robert I. Davis, and Iain Bate. 2017. Analysis and Optimization of Message Acceptance Filter Configurations for Controller Area Network (CAN). In *Proceedings of RTNS '17, Grenoble, France, October 4–6, 2017,* 10 pages.
https://doi.org/10.1145/3139258.3139266

RTNS '17, October 4–6, 2017, Grenoble, France

ACM ISBN 978-1-4503-5286-4/17/10…$15.00
https://doi.org/10.1145/3139258.3139266

1 INTRODUCTION

Controller Area Network (CAN) is a multi-master serial data bus [3] with real-time capabilities [10, 11]. CAN is a broadcast bus, thus each message that is transmitted on the network can be received by every node connected to it. Each node then needs to decide if the message is relevant for it or not. This is done by examining the message ID, which uniquely identifies each message. Effectively, each node needs to perform *filtering* of all incoming messages, and then process only those messages relevant to it.

If a high number of messages are broadcast over the network, but only a small number of them are relevant to a particular node, then software-based filtering can become a substantial source of unnecessary runtime overhead. The high number of message receive interrupts and the execution of code for software-based filtering cause significant processor load; particularly for processors running at relatively low clock speeds. In order to tackle this problem, CAN-controllers are equipped with hardware based *message acceptance filters*. When configured appropriately, these filters can block most or all of the messages which are not relevant to the node. Since filtering is done via hardware, the ECU only processes relevant messages avoiding unnecessary overhead. If the hardware filter configuration is not perfect, then some undesired messages will still pass through. In general, a multi-stage approach is typically needed, involving (i) hardware-based message filtering, (ii) software-based message filtering, and finally (iii) message processing.

Hardware-based filtering is useful for any CAN-node, but is particularly important for low performance nodes that are connected to a heavily utilized network, and for gateway nodes which are connected to multiple networks. Clearly the goal is to do as much filtering as possible in hardware, as it comes with no processing cost. A perfect hardware filter configuration is one where no subsequent software filtering is needed and no desired messages are discarded. This raises an interesting and challenging engineering task: How best to configure the hardware-based message acceptance filters? To the best of our knowledge, this problem has not been addressed by the research community.

With CAN, as well as uniquely identifying the message, the message ID is also used as the priority for bus arbitration [10]. This dual purpose complicates the problem of ID assignment and message filtering. In some systems, for example based on Volcano[1], designers have full control over the configuration of all message IDs. In this case the two key problems of ID assignment and message filtering can be considered together. An effective way of doing this is to partition the message ID field into two sections, with the most significant bits (that are transmitted first) reserved for the unique message priority, and the least significant bits used for filtering.

[1] See https://www.mentor.com/products/vnd/in-vehicle_software/

With 29-bit IDs, 11 bits could be used for a unique message priority, with the remaining 18 bits used as an address field to indicate which nodes on the network need to receive the message (1 bit per node). With this scheme, perfect filtering can be supported for up to 18 nodes, which is sufficient for most automotive applications[2]. With 11-bit IDs the problem is more acute, since with typical numbers of messages (circa 100) there are too few bits left for effective filtering.

The ideal scenario of complete freedom to set the IDs of all messages is in any case rarely seen in practice [7]. The use of legacy software, ECUs transmitting messages with predefined IDs (for logistical reasons), and the allocation of message IDs reflecting only the ECU supplier and data content mean that in many systems the message IDs are fixed. This leaves the problem of determining, for each individual ECU, the best filter configuration, given the fixed set of messages transmitted on the network and the subset of those messages which it needs to receive. This is the problem that we address in this paper.

We focus on two related questions: 1. How to assess the quality of a filter configuration? 2. How to find an optimal filter configuration? By answering these questions, we make the following contributions: (i) We bring the CAN message acceptance filter problem to the attention of the real-time community. (ii) We formally define the problem, and its complexity. (iii) We provide a metric for determining the quality of filter configurations. (iv) We provide a method for finding optimal or near-optimal filter configurations.

1.1 Organization

The remainder of the paper is organized as follows: Section 2 introduces the system model, explains how hardware-based message acceptance filtering works, and provides a brief review of the hardware filters provided by different CAN controllers. Section 4 describes a quantitative metric that can be used to assess the quality of a given filter configuration. Section 5 considers the problem of optimizing the filter configuration. The general problem is shown to be NP-complete, optimal solutions are provided for special cases, and a generic approach based on simulated annealing is proposed. Section 6 evaluates the proposed approach on synthetically generated message sets, and on an industrial case study. Section 7 concludes with a summary and directions for future work.

2 MESSAGE FILTERING

2.1 System Model

We assume that the system comprises a set of ECUs or nodes connected via a CAN bus. Each node can broadcast messages which can be received by all other nodes on the network. Each message m is characterized by its unique message ID (which is 11 or 29 bits long), its period or minimal inter-arrival time T_m, deadline D_m, and its payload data which determines the maximum length of the message. The payload data comprises a set of signals which need to be received by one or more nodes. In general, each node has a subset of messages that it needs to receive, referred to as *desired* messages, and a subset of messages that it is not interested in, referred to as *undesired* messages.

[2]While many vehicles have upwards of 50 ECUs, these are typically connected in smaller groups via multiple CAN buses and other networks.

Within its CAN-controller each node has a set of buffers, which can be configured to either transmit or receive CAN messages. In the literature and CAN controller documentation, these buffers are often referred to as "message objects". Typically, each receive buffer has a hardware-based message acceptance filter that can be set. We model this by assuming that each node has f acceptance filters. Each broadcast message can either pass through one (or more) of the filters, or is blocked by them. The messages which pass through the filters raise a receive interrupt or set a bit in a control register, and are then processed by the node. Note that depending on the filter settings, messages of more than one ID can be received by the same buffer. Similarly, a single message may pass through more than one filter, in which case it is only received in one of the buffers, according to some implementation dependent policy.

2.2 Hardware-based Filtering

The hardware-based acceptance filtering works as follows: The ID of an incoming message is compared against the specified acceptance *filter pattern*. If the ID matches the filter pattern, the message passes through, and is stored in the receive buffer. If the ID does not match the filter pattern, the message is blocked. The filter pattern comprises two registers per filter: The `mask` specifies which bits of the ID are considered, and the `tag` specifies the corresponding ID-values that are allowed to pass.

The filter logic is shown in the pseudo-code below; however, note that the actual implementation is by shift registers and logic gates in hardware.

```
if (ID AND mask) == (tag AND mask)
  pass = true
else
  pass = false
end
```

Table 1 gives a simple example. By specifying the acceptance filter pattern `0001100xx00`, only messages with message IDs 192, 196, 200, or 204 will pass through the filter, and arrive in the receive buffer. All other messages will be blocked. Note that an `x` in the filter pattern means that the bit value at that bit position does not matter, also referred to as "don't care".

Field	Value (Bin.)	(Dec.)
mask	111 1111 0011	
tag	000 1100 0000	
filter	000 1100 xx00	
pass ID	000 1100 0000	192
pass ID	000 1100 0100	196
pass ID	000 1100 1000	200
pass ID	000 1100 1100	204

Table 1: Example of message acceptance filtering

Further examples that explain how acceptance filtering works can be found in [30] on page 45, or in [1] on page 246.

For the sake of simplicity, we use the term *filter* as a synonym for acceptance filter and filter pattern in the rest of the paper. Further, we make use of the abstract filter pattern notation (with 0, 1, and `x`) instead of the specific `mask` and `tag` values.

2.3 CAN-controller Filter Implementations

While the basic principle of CAN message acceptance filtering is the same for all CAN-controllers, there are differences in implementation and number of available filters.

The most effective implementation is where each message buffer has its own dedicated mask and tag.

ATMEL's AT90CAN128 [1] is a low-power 8-bit microcontroller, its CAN-controller has 15 message buffers, and each one has its own mask. Infineon's CAN-controllers (called MultiCAN) [18] offer several variants on XC16, TriCore and AURIX processors. This CAN-controller implementation also has a dedicated mask for each message buffer.

In contrast, there are some CAN-controllers which share a global mask between message buffers (instead of having a dedicated mask for each buffer). This limits flexibility and efficiency. Note each buffer still has its own tag.

Freescale's MPC555 [14] is a 32-bit microcontroller. It has 2 CAN-modules (called TouCAN). Mitsubishi's M32R microcontroller also has 2 CAN-modules, while Renesas' M16C [25] microcontroller has 1 CAN-module. All of these CAN modules have 16 message buffers; however, they only have 3 masks. One global mask (for buffers 0 to 13), and 2 local masks (for buffers 14 and 15 respectively). National's CR16 [21] microcontroller has 1 CAN-module with 15 message buffers. It has 2 masks. One global mask (for buffers 0 to 13) and one local mask (for buffer 14).

Note that the message buffers can be configured to transmit or receive messages. Thus for example if 7 out of 15 message buffers are used to transmit messages, then that leaves at most 8 message buffers for received messages.

Motorola's CAN-controller (called msCAN) offers a single 32-bit filter. However, the filter can be configured flexibly: 1x 32-bit, or 2x 16-bit, or 1x 16-bit and 2x 8-bit, or 4x 8-bit. The 32-bit option is suitable for 29-bit message IDs, and the 16-bit option for 11-bit message IDs. When using the 8-bit option, only a subset of the message ID bits can be utilized for filtering, making finding an effective filter pattern more difficult; however, on the positive side, more filters are available. The M68HC08 family [15] uses one msCAN module, while the M68HC12 and S08 [17] use two msCAN modules. Motorola's *msCAN Filter Configuration Tool* [16] automates the task of finding a suitable filter size (32, 16, or 8 bits) and filter values, based on the specification of all broadcast messages and desired messages. However, it does not consider any timing information, such as the transmission period. The tool's internal algorithms are not publicly available, and unfortunately the tool is no longer available either.

This brief review only covers a subset of available CAN-controllers. However it shows that almost all of them fall into one of two categories: "dedicated mask" CAN-controller (where each buffer has its own dedicated mask) or "shared mask" CAN-controllers (where a mask is shared between several buffers).

3 RELATED WORK

Schedulability analysis for CAN was first developed in the mid 1990's, with the flaws in that early work later corrected in 2007 [10]. This analysis provides guarantees that CAN messages will meet their deadlines, including under a prescribed error model, i.e accounting for re-transmissions when there are errors on the bus.

Message acceptance filtering is highly dependent on the message ID assignment. In industrial applications, message ID and thus priority assignment has often followed an ad-hoc approach, with message IDs allocated based on the ECU supplier and the type of signals contained in the message. This has led to priority assignments that leave automotive networks unschedulable at bus utilizations of more than about 30-35% [4], when more appropriate priority assignment would allow bus utilizations of around 80% [8] before any deadlines are missed.

Academic research into priority assignment for CAN has mainly focused on optimal priority assignment policies, with Audsley's OPA algorithm [2, 22] proved optimal for systems using *priority queues* [10], and deadline minus jitter priority assignment proved optimal with some common constraints on the sets of messages [7]. Further work has explored the issues that can arise if the priority-based arbitration mechanism is circumvented, for example by the use of non-abortable transmit buffers [20], or *FIFO queues* [8, 9]. In practical applications, using an optimal priority assignment policy is not in itself enough, since the ordering generated could leave the system only just schedulable, and thus vulnerable to deadline misses in the event that there is an increase in errors on the bus. Work on robust priority ordering [5, 6] 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). Later work provides a robust priority assignment for new messages added to a system where existing message IDs are fixed [7], addressing flaws in previous work in this area [26].

One might expect that a system that is robust would also be extensible, i.e. most able to accommodate additional messages; however, this is not necessarily the case [23]. Recent work in this area [24] provides an extensibility metric for CAN, and a message ID assignment which optimizes extensibility by assigning IDs according to ID-bands, aligned with timing requirements.

Other related works aim to provide holistic solutions for task allocation, signal to message mapping and priority assignment using Mixed Integer Linear Programming (MILP) [28, 29].

We note that none of the above works on message priority and ID assignment considers the impact on message acceptance filtering. In this paper, we consider the problematic case where message IDs have already been fixed without consideration for message filtering. As far as we are aware there is no prior work addressing this specific problem.

4 MESSAGE FILTER QUALITY

In this section, we focus on how to quantify the quality of a given filter configuration. We make use of the following notation: M^{all} is the complete set of m messages which are broadcast over the network. M^{des} is the subset those messages which must be received by the node of interest. F is the node's filter configuration, consisting of f filters. We need to determine (i) if the filter configuration is feasible, and (ii) a measure of its quality.

By applying the acceptance rules of the filter configuration F on the set of broadcast messages M^{all} the result is the set of messages M^{pass} that pass through the filters, and the set of messages $M^{block} = M^{all} \setminus M^{pass}$ that are blocked by the filters. By comparing these sets against the set of messages M^{des} that must be received, we obtain two further subsets:

$M^{UB} = M^{block} \cap M^{des}$ is the set of messages that are blocked by the filters, but are needed by the node (i.e. *unintended block*), and $M^{UP} = M^{pass} \setminus M^{des}$ is the set of messages that pass through the filters, even though they are not needed by the node (i.e. *unintended pass*). Based on these sets, we can analyse the filter configuration as follows. First we make a classification with respect to feasibility. The filter configuration is *infeasible* if there is at least one unintended blocked message (i.e. $M^{UB} \neq \emptyset$). It is *feasible* if there are no unintended blocked messages (i.e. $M^{UB} = \emptyset$). Further, we say that the configuration is *perfect* if there are no unintended blocked messages, and no unintended pass messages either (i.e. $M^{UB} = \emptyset$ and $M^{UP} = \emptyset$).

Clearly, the aim is to achieve perfect message filtering. However, due to the limited number of available filters and the way in which the filters work, this may not be possible in all cases. Thus, we propose a measure of the *quality* of a feasible filter configuration in terms of its imperfection, i.e. considering the set of unintended pass messages M^{UP}. Since each message is transmitted with a given period or minimum inter-arrival time, we normalize by T_m.

$$QoF = \frac{M^{UP}}{sec.} = \sum_{M^{UP}} \frac{1}{T_m} \tag{1}$$

This imperfection metric measures how many unintended pass messages per second are received, even though we would rather they were blocked. These messages cause undesired receive interrupts and additional processing load. The lower the imperfection is, the better the filtering. Ideally, imperfection becomes zero (i.e. perfect filtering).

5 DESIGNING OPTIMAL FILTERING

In this section, we focus on optimizing the filter configuration (i.e. obtaining a filter configuration that is both feasible and minimizes the imperfection metric QoF). First we consider the complexity of the problem, then we provide solutions for some special cases, and finally propose an approach to solving the general problem.

5.1 Problem Complexity

The number of possible filter configurations is exponential in the length (*len*) of the message IDs and the number of filters f. Since each filter bit can effectively take one of 3 values (0, 1, x) where x represents "don't care", there are $3^{(len \cdot f)}$ possible filter configurations. This value quickly becomes large, thus exhaustive enumeration of all possible filter configurations is intractable for realistic sized problems. (For example with 11-bit IDs and 3 filters, there are over 10^{15} possible configurations).

We now show that the general problem of filter selection is *NP-complete* via reduction to the SET COVER problem [19].

The SET COVER problem is as follows. Given a Universe of elements $X = \{1, 2, \ldots, n\}$ and a collection $S = \{S_1, S_2, \ldots, S_m\}$ of m subsets of X whose union equals X, determine if there is a set covering of size k (i.e. k or fewer sets from S) whose union is X.

The filter selection problem is as follows. Given a set of desired messages $M^{des} = \{M_1, M_2, \ldots, M_n\}$ and at least one undesired message M_u, with IDs given by bit patterns of length m, determine if there is a selection of at most k filters (selecting bits in the message IDs as must-match 1, must-match 0 or don't-care) that permit all desired messages to be received but exclude the undesired message.

Theorem 5.1. *The filter selection problem is NP-complete.*

Proof. We prove the theorem via reduction to the SET COVER problem which is known to be NP-complete [19].

First, we note that a solution to the filter problem may be trivially checked by a deterministic algorithm in polynomial time. For each desired message we check against the selected filters to ensure that it can be received via at least one of them, and for the undesired message we check that it cannot be received via any of the selected filters. The filter selection problem is thus in the NP complexity class.

Given an instance of the SET COVER problem, we construct an instance of the filter selection problem as follows. Each element in the set X maps to a message with that index in the set of desired messages $M^{des} = \{M_1, M_2, \ldots, M_n\}$. Hence there is a one-to-one mapping between elements in X and desired messages. Each subset S_i in S maps to a unique bit position i in the message IDs. (There are m bits in the message IDs, equal to the cardinality of the collection S). All desired messages with indices in S_i have a 1 in bit position i, all other messages, both desired and undesired, have zeros in bit position i. Note this completely defines the IDs of all messages, with the undesired message having all bits in its ID set to zero.

We note that without loss of generality the only filters that need to be considered in solving an instance of the filter selection problem, constructed from an instance of SET COVER in the way described above, are those with a single bit set to must-match 1 and don't-care for all other bits. If no bits are set to must-match 1, then the undesired message will be received. Setting more than one bit to must-match 1, e.g. bits i and j would result in receiving messages matching elements in the intersection of the two sets $(S_i \cap S_j)$ which confers no advantage over receiving either all of the messages matching elements in S_i or all of those matching elements in S_j. With a single bit i set to must-match 1, setting any other bit to must-match zero cannot include any further desired messages. Thus without loss of generality, we restrict the available filters to the set $F = \{F_1, F_2, \ldots, F_n\}$ where F_i indicates must-match 1 in bit position i and don't-care for all other bits. Note the precise one-to-one mapping between F_i and S_i. Filter F_i with a must-match 1 in bit position i and don't-care for all other bits receives only those desired messages with indices matching elements in S_i.

Now assume we have a black box that can solve the filter selection decision problem. Via the above construction, we may use this black box to solve the SET COVER decision problem. Correctness of this approach needs to be shown for both if and only if cases.

If case: For an instance of the SET COVER problem for which the answer is yes, there exists a sub-collection $S' \subseteq S$ of cardinality k that covers X. Mapping this instance to the filter selection problem, then S' implies that there is an equivalent collection of filters $F' \subseteq F$ of cardinality k that enables all desired messages to be received, without receiving the undesired message. The black box, which can solve all filter selection problems, therefore gives the answer yes.

Only if case: If the black box returns yes, then there exists a sub-collection of filters $F' \subseteq F$ of cardinality k that enables all desired messages to be received. This implies that there is an equivalent sub-collection S' of cardinality k that covers X. Similarly, if there is

no such sub-collection of filters $F' \subseteq F$ of cardinality k, then there is no such sub-collection S' of cardinality k that covers X.

We have shown that our algorithm solves the SET COVER problem using the black box for the filter selection problem. Since the construction takes polynomial time, and we have shown that the filter selection problem is in the NP complexity class, we conclude that the filter selection problem is NP-complete □

We reduced a simplified version of the filter selection problem to SET COVER, showing that the former is also NP-complete. The more general filter selection problem, with more than one must-match bit per filter, and optimization of the weighted sum of undesired messages which are received is at least as hard. While the decision problem of SET COVER is NP-complete, the optimization problem (finding the smallest cardinality k which achieves coverage) is NP-hard. Given the direct mapping to the filter selection problem, we expect that its optimization is also NP-hard.

5.2 Special Cases

The filter configuration problem comes in three variants, depending on the relationship between the number of available filters and the number of desired messages.

Case $f = 1$: There is only a single filter available. For this special case Algorithm 1 constructs an optimal feasible filter configuration. It sets the filter digit in position i to 0 if the bits in position i of IDs of all desired messages are 0. Similarly, if they are all 1, then it sets the filter digit to 1. Otherwise it sets the filter digit to x.

Algorithm 1: Optimal Solution for $f = 1$

```
   Input: M^des                          /* desired messages */
1  foreach ID-digit do
2  |  if ∀m_i ∈ M^des the ID-digit is 0 then
3  |  |  filter-digit = 0
4  |  else if ∀m_i ∈ M^des the ID-digit is 1 then
5  |  |  filter-digit = 1
6  |  else
7  |  |  filter-digit = x
   Output: F                             /* feasible filtering */
```

Theorem 5.2. *For the restricted case of a single filter, Algorithm 1 is optimal.*

Proof. We prove the theorem by showing that any changes to the filter pattern obtained by Algorithm 1 either make the filter configuration infeasible (i.e. block some desired messages) or allow additional undesired messages to be received. We consider each digit of the filter obtained by Algorithm 1 in turn.

Case filter-digit = 0: Changing this digit to 1 would exclude some desired messages, making the configuration infeasible. Changing it to x is unnecessary to allow all desired messages to be received and could potentially allow through undesired messages.

Case filter-digit = 1: Changing this digit to 0 would exclude some desired messages, making the configuration infeasible. Changing it to x is unnecessary to allow all desired messages to be received and could potentially allow through undesired messages.

Case filter-digit = x: Changing this digit to 0 or 1 would exclude some desired messages, making the configuration infeasible.

Since no changes to the filter configuration obtained by Algorithm 1 can improve the imperfection metric QoF, the filter configuration is optimal □

Case $f \geq |M^{des}|$: The number of available filters equals or exceeds the number of messages that must be received. For this special case, Algorithm 2 constructs a perfect filter configuration. It sets each filter equal to the unique ID of one desired message.

Algorithm 2: Optimal Solution for $f \geq |M^{des}|$ Filters

```
   Input: M^des                          /* desired messages */
1  foreach m_i ∈ M^des do
2  |  f_i = ID(m_i)
   Output: F                             /* perfect filtering */
```

Theorem 5.3. *For the special case of at least as many filters as desired messages, Algorithm 2 provides an optimal filter configuration which achieves perfect filtering.*

Proof. Since Algorithm 2 uses a specific filter for each desired message which enables only that message to pass, and all messages on CAN have unique IDs, then there are no unintended messages that pass, and thus the filter configuration is perfect and the imperfection metric QoF zero □

We note that Algorithm 2 has one minor drawback: It uses as many filters as there are desired messages. As we later show, it is sometimes possible to construct a perfect filter configuration using fewer filters.

Case $1 < f < |M^{des}|$: The number of available filters is smaller than the number of desired messages that must be received. This is the general case. Here, we aim to find a feasible filter configuration which minimizes the imperfection metric. We know that the problem is NP-complete, thus we propose the use of a meta-heuristic search-based solution, specifically Simulated Annealing (SA). We initially considered two different approaches, both using SA:

Direct: We set the pattern in each of the f filters directly. The downside of this approach is that it explores many patterns that are not useful, i.e. that do not allow through some desired messages. In fact most of the possible filters configurations are like this.

Two-step: We first divide the set of desired messages M^{des} into f groups, and allocate each group to one filter. Based on this allocation, we then derive the pattern for each filter by applying Algorithm 1. The benefit of this approach is that each local filter pattern is optimal. This does not,however, guarantee a globally optimal solution.

Preliminary experiments showed that the two-step approach is much more effective, we therefore tackle the problem in that way.

5.3 Generic Solution: Simulated Annealing

We use Simulated Annealing to solve the filter configuration problem for $1 < f < |M^{des}|$. SA has several benefits: It rarely gets stuck in local optima, thus it is likely that it will find the global optimum. It is relatively easy to understand, and thus it is likely that engineers will accept it. Further, it is relatively easy to adapt to a specific problem, since there are only a few specific parameters that need to be tuned. Finally, it is effective in solving

complex problems, as has already been demonstrated for several problems in the real-time systems domain (e.g. task allocation [12, 13, 27] and network configuration [23]).

SA starts with an initial candidate solution. It then uses the *neighbor-move function* to transform this solution into a new/different solution. For each generated solution, the solution's quality is evaluated via the *cost function*. If the new solution improves the quality, it becomes the new starting point for subsequent exploration. If not, it may still be chosen according to an ever decreasing probability. This ensures that the search does not get stuck in a local optima. After some time, the search reaches its termination criteria (specified in terms of a maximum number of iterations, or quality improvements below a certain threshold), and returns the best solution found.

In order to apply SA to the filter configuration problem, we need to encode the optimization criteria into the cost function, and implement an appropriate neighbor-move function.

5.3.1 Cost Function. The cost function measures the quality of a filter configuration. We encode it as a weighted sum as follows:

$$cost = \frac{\sum cost_i \cdot w_i}{\sum w_i} \rightarrow \min \tag{2}$$

The individual cost terms encode the constraints and optimization goals. We normalize each cost term so that it takes values from 0.0 to 1.0, where 0.0 represents the optimum.

$$cost_1 = \frac{M^{UB}}{M^{des}} \tag{3}$$

$$cost_2 = \frac{M^{UP}/sec.}{(M^{all} \setminus M^{des})/sec.} \tag{4}$$

$cost_1$ measures the number of unintended blocked messages, and normalizes them by the number of desired messages. It is responsible for avoiding infeasible filter configurations. $cost_2$ measures the number of unintended pass messages per second, and normalizes them by the number of intended blocked messages per second. It is responsible for optimizing the filter quality. Note that both cost terms are dimensionless quantities.

The weights of the cost terms were set during a manual tuning phase. During that phase we tried several different weights, and evaluated which ones led to a good results. The weights chosen were: $w_1 = 100$ and $w_2 = 3$. These are our recommendations, they could be set differently by engineers using this solution; however, such changes should be made with care, as they can impact the search performance and the quality of the best solution found.

5.3.2 Neighbour-Move Function. The neighbour-move function transforms one filter configuration into another. We used a simple transformation based on moving one desired message from one filter to another. A desired message is chosen at random from M^{des}; a filter is chosen at random from the set of filters to which the message is not currently assigned; the message is then assigned to that filter. Once the neighbour-move has transformed the filter configuration, Algorithm 1 is applied to derive a feasible pattern for each filter. Note only the two filters that have a modified allocation of messages need be re-evaluated.

5.3.3 SA Parameters. We used the following SA parameters: initial temperature = 0.05, cooling-factor = 0.95, iterations at same temperature = 100, max iterations = 10,000. (These parameters were set after a manual tuning phase). In addition, we improved the search time via a problem specific exit criteria; once SA finds a perfect filter configuration, there is no need to search any further, thus the search terminates.

Algorithm 3: Simulated Annealing

```
   Input: t                                   /* initial temperature */
   Input: s_cur                               /* initial solution */
 1 c_cur = cost(s_cur)                        /* initial cost */
 2 repeat
 3 |   iterAtT = 0
 4 |   repeat
 5 |   |   iter++
 6 |   |   iterAtT++
 7 |   |   /* generate new solution */
 8 |   |   s_new = neighbour(s_cur)
 9 |   |   c_new = cost(s_new)
10 |   |   /* accept move? */
11 |   |   if c_new < c_best then
12 |   |   |   /* cost is improved */
13 |   |   |   s_cur = s_new
14 |   |   else
15 |   |   |   /* cost is not improved */
16 |   |   |   if e^((c_cur - c_new)/t) > random(0, 1) then
17 |   |   |   |   s_cur = s_new
18 |   |   /* remember best solution */
19 |   |   if c_new < c_best then
20 |   |   |   c_best = c_new
21 |   |   |   s_best = s_new
22 |   until iterAtT == iterAtTmax;
23 |   t = t * coolingFactor
24 until iter == iterMax;
   Output: s_best                             /* best solution found */
```

5.4 Engineering Heuristic and Initial Solution

To the best of our knowledge, there are no approaches to solving the general filter configuration problem that are available in the literature. For the purposes of comparison, we make use of a heuristic approach which, in the absence of the method proposed in this paper, could be used in the form of a simple engineering solution. The approach is as follows. First, the desired messages are sorted by message ID. Next the desired messages are assigned in order to the available filters, so that $\lceil |M^{des}|/f \rceil$ messages are assigned to each filter. Due to the initial ordering, the messages assigned to each filter may have similar IDs (improving the filter quality). A feasible filter pattern is then derived for each filter using Algorithm 1. In the evaluation that follows, we indicate this heuristic solution by H. We also use it as the initial starting point for SA. In that way SA dominates the heuristic, since it always returns a solution that is at least as good.

6 EVALUATION

In this section we evaluate the performance of the proposed approaches for various scenarios. We use synthetic examples, which are randomly generated. Message IDs are 11-Bit (i.e. standard format), and their values are randomly chosen from 0 to 2047 (uniformly distributed). Message periods are randomly chosen from 10 ms to 1000 ms according to a log-uniform

distribution. The set of messages M^{des} that must be received by the node are a randomly chosen subset of the messages M^{all} that are broadcast on the network. We examine scenarios with 25 to 100 broadcast messages, from 5 to 40 desired messages, and from 1 to 16 filters. We assume that each message has the maximum length (i.e. 8 data bytes). Only those message sets that were schedulable at a bus speed of 1Mbit/sec were included in the evaluation.

Our evaluation criteria focus on the best solutions that are found by the different approaches. The *perfect rate* indicates the proportion of examples where a perfect filter configuration was found. *Filter quality* is the optimization goal, and is encoded as $1 - cost_2$, thus 1.0 represents an optimal solution. All values shown (points on the graphs) are the average over 100 examples for that specific scenario.

6.1 One Filter

First, we examine the case where only a single filter is available ($f = 1$). This is relevant for low-cost CAN-controllers. Algorithm 1 will (by definition) always construct a feasible filter, and with some luck this might be a perfect filter; however, this solely depends on the message IDs. This problem is the most constrained, which makes it hard to obtain a feasible filter with good filter quality. Figure 1 illustrates how often a *perfect* filter configuration is obtained. We observe that perfect filtering can only be achieved for a very small number of desired messages, and becomes virtually impossible for a larger number.

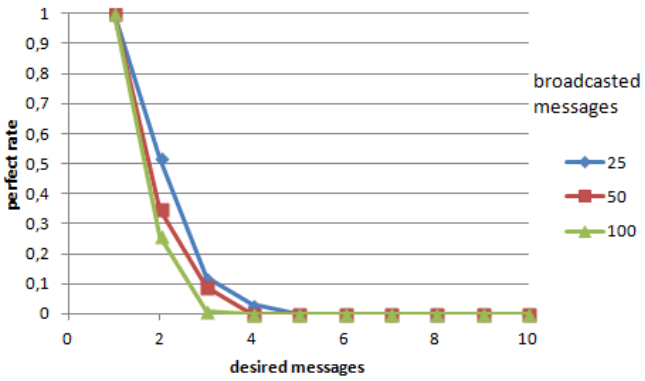

Figure 1: Perfect Filtering with 1 Filter

Figure 2 illustrates the *filter quality* ($1 - cost_2$). Here, we observe that filter quality significantly worsens with an increasing number of desired messages. Once we reach 10 desired messages, the filter is ineffective; it is letting all of the broadcast messages pass.

Based on these experiments, we conclude that with a single filter, it is unlikely that high quality or perfect filtering can be achieved once there are more than approx. 5 desired messages. (Obviously this is dependent on the actual IDs of the broadcast and desired messages). This means that ECUs with a CAN-controller with only a single filter are likely to have to filter many messages in software, and thus incur additional runtime overheads.

6.2 More Filters than Desired Messages

When the number of filters equals or exceeds the number of desired messages ($f \geq |M^{des}|$) then Algorithm 2 will (by definition) always construct a perfect filter configuration. Therefore evaluating this case and Algorithm 2 is of little interest.

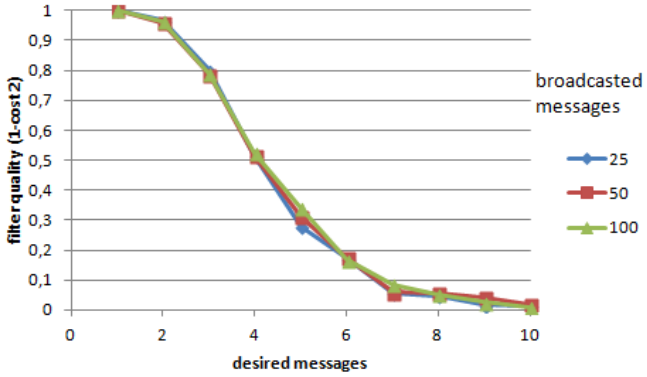

Figure 2: Filter Quality with 1 Filter

6.3 Fewer Filters than Desired Messages

Next, we evaluate the performance of the SA-based solution for the general case, where the number of available filters is smaller than the number of desired messages ($1 < f < |M^{des}|$). For comparison purposes, we also employed the engineering heuristic, described in section 5.4 (marked as H on the graphs).

Figure 3 illustrates how the number of filters and the number of broadcast messages impacts filter quality for a fixed number of desired messages ($|M^{des}| = 20$). We observe that as the number of filters increases from 2 to 16, then as expected the filter quality improves. Filter quality is also better for fewer broadcast messages. These trends hold for both SA and the heuristic; however, SA achieves significantly better filter performance.

Figure 5 illustrates how filter quality is impacted by the number of desired messages. In this case, the number of broadcast messages is set to 100, and we vary the number of available filters and the number of desired messages. Here, as expected the filter quality worsens as the number of desired messages increases. Again SA finds significantly better filter configurations than the heuristic.

Finally, Figure 7 illustrates sensitivity with respect to the number of available filters. Here, we set the number of broadcast messages to 100, and varied the number of desired messages and available filters. We observe that as the number of filters increases, so does the filter quality. Again, SA finds significantly better filter configurations than the heuristic. Notably, SA with 2 filters has similar performance to the heuristic with 4 filters, and again for 4 versus 8 filters.

6.4 Shared Global Filter Mask

Some CAN-controllers (e.g. the CAN modules on Freescales's MPC555, Mitsubishi's M32R, Renesas' M16C, and Nationale's CR16 microcontrollers) use a global filter mask that is shared between several message buffers, with one or two message buffers having separate local masks. Such systems are often configured with $(f - 1)$ message buffers (using the global mask) set to receive only 1 message each, and one message buffer (using a local mask) set to receive all the remaining desired messages.

In order to evaluate the effect of a shared global mask, we implemented this approach using SA via an alternative

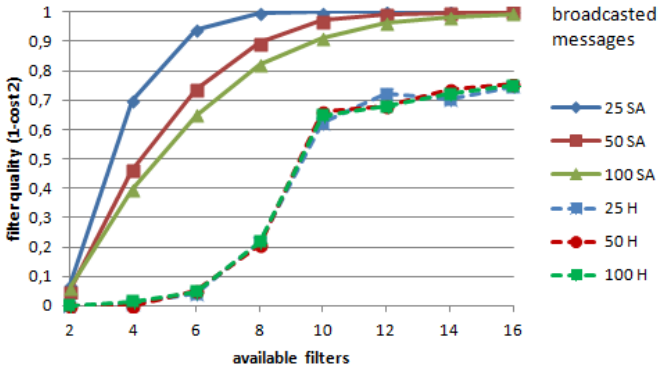

Figure 3: Filter Quality using f Filters ($1 < f < |M^{des}|$) for 20 desired messages

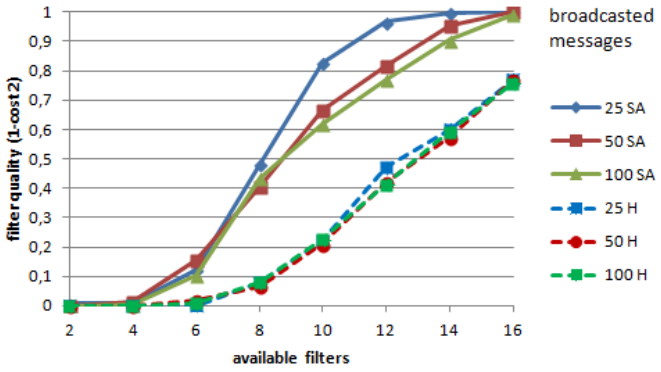

Figure 4: Filter Quality using f Filters ($1 < f < |M^{des}|$) for 20 desired messages, using *f-1* approach

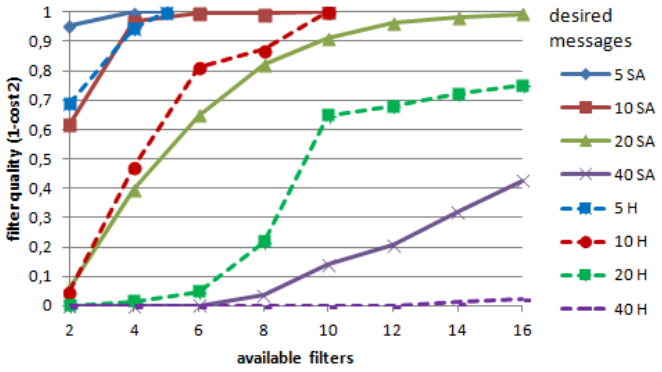

Figure 5: Filter Quality using f Filters ($1 < f < |M^{des}|$) for 100 broadcast messages

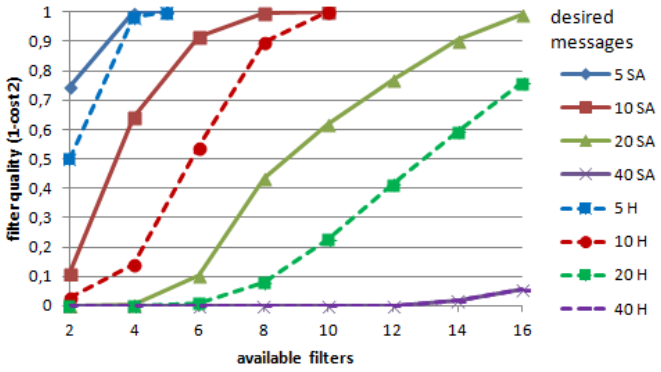

Figure 6: Filter Quality using f Filters ($1 < f < |M^{des}|$) for 100 broadcast messages, using *f-1* approach

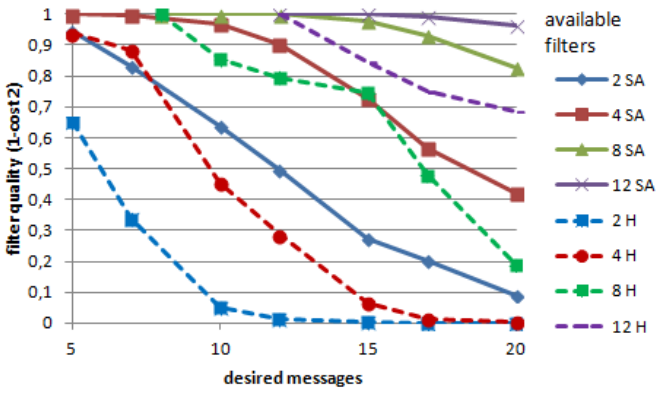

Figure 7: Filter Quality using f Filters ($1 < f < |M^{des}|$) for 100 broadcast messages

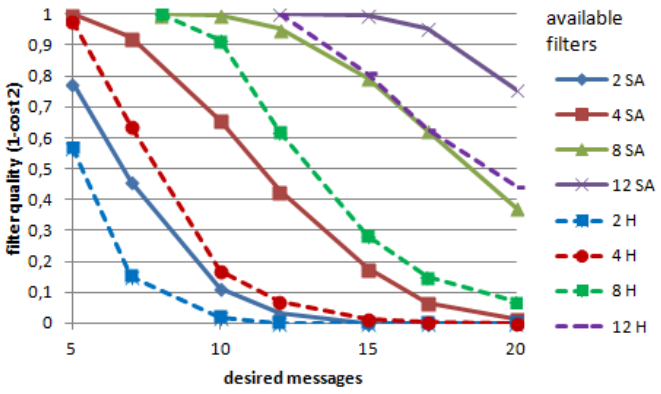

Figure 8: Filter Quality using f Filters ($1 < f < |M^{des}|$) for 100 broadcast messages, using *f-1* approach

In the above figures we only examine scenarios where the number of filters is smaller than the number of desired messages ($1 < f < |M^{des}|$). Once the number of filters equals or exceeds the number of desired messages ($f \geq |M^{des}|$) then the filtering is always perfect (i.e. filter quality = 1.0); however, we do not plot these scenarios here. This is why some lines in Figures 5-8 do not span the entire x-axis.

neighbour-move function. This ensures that for $f-1$ filters only 1 message is assigned. We also implemented a modified heuristic that assigns $|M^{des}|-(f-1)$ messages to one filter, and the remaining $f-1$ messages individually to the remaining filters.

Figures 4, 6, and 8, on the right hand side of the previous page, show the results for a repeat of the experiments shown in Figures 3, 5, and 7, but assuming a shared global filter mask. While the overall trends from the previous experiments hold, the overall filter quality is significantly worse, indicating an inherent loss of performance compared to having an individual filter mask for each message buffer. We conclude that it is preferable to have one filter (mask and tag) per message buffer, rather then needing to share a mask between buffers. A single global mask becomes detrimental to filter performance once the number of desired messages exceeds the number of receive buffers.

6.5 Industrial Case Study

Evaluation based on synthetic examples shows that the proposed SA-based solution is effective in finding optimal or near-optimal message filtering. In this section, we demonstrate its practicality by applying it to an industrial case study.

The system is a HVAC (heating, ventilation and air conditioning) controller for a lightweight battery electric vehicle. The vehicle was developed in the EU-funded project Epsilon[3]. We used an AT90CAN128 micro-controller [1] which has 15 message buffers; 4 of which were used for transmitting messages, leaving 11 that could be used for receiving messages. In total 55 messages are broadcast on the network, out of which the HVAC node must receive and process 11 messages, marked as "desired = yes" in Table 2.

Since the number of desired messages matches the number of available receive buffers, the engineers originally configured the system such that each of the buffers receives a single message, the same as Algorithm 2 does, thus resulting in a perfect filter configuration. Since this is somewhat uninteresting, we investigated whether perfect filtering can be achieved using fewer filters (receive buffers). We therefore reduced the number of filters from 11 down to 1 in steps of 1, and applied the SA algorithm at each step. We repeated each experiment 100 times, so that we could also examine how often perfect filtering was obtained. Figure 9 shows the results. With 3 filters, good filter quality is obtained, with for example only 48 undesired messages/sec requiring software filtering, compared to 227 desired messages/sec received, and 1407 messages/sec broadcast in total. With 7 filters perfect filtering can be achieved. Table 3 shows one such perfect filter configuration using 7 filters. With this configuration, 4 filters (message buffers) would be left for future proofing the HVAC-controller, for example allowing for further messages to be transmitted.

The CAN receive interrupt handler execution time is between 220 and 610 cycles (13.75 μs and 38.13 μs at 16MHz), not including software filtering. Assuming an average execution time of approx. 500 cycles (31.25 μs) with software filtering, then the overhead of filtering out all of the 1180 undesired messages/sec in software is approx. 36.9 ms. Using 3 hardware filters, the number of undesired messages/sec can be reduced to 48, implying overheads of just 1.5 ms. By comparison, receiving the 227 desired messages/sec takes approx. 7.1 ms of processing time.

[3]http://www.epsilon-project.eu

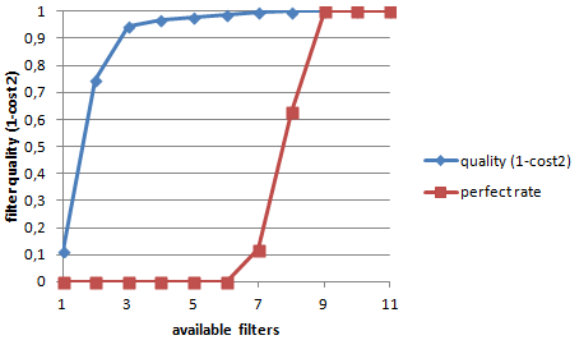

Figure 9: Filter Quality for HVAC-controller

ID [hex]	T [ms]	desired	ID [hex]	T [ms]	desired
0x00A	1000		0x2F2	100	
0x010	1000	yes	0x300	1000	
0x020	100		0x350	1000	
0x060	100		0x3AC	100	
0x06A	100	yes	0x400	100	
0x06E	10		0x410	100	
0x071	100		0x411	100	
0x078	100		0x420	1000	
0x07D	100		0x425	1000	
0x081	1000	yes	0x565	500	
0x08C	100		0x610	100	
0x091	100		0x611	1000	
0x096	500		0x612	100	
0x100	10		0x613	1000	yes
0x101	10		0x614	1000	
0x102	10		0x618	100	
0x110	10		0x620	100	yes
0x120	10		0x6F0	1000	yes
0x130	10		0x6F8	1000	
0x150	10		0x700	1000	
0x160	1000		0x702	1000	
0x161	1000		0x710	1000	yes
0x200	1000		0x711	1000	
0x201	10		0x720	1000	
0x210	10	yes	0x730	1000	yes
0x220	10	yes	0x770	1000	yes
0x2F0	10		0x771	1000	
0x2F1	100				

Table 2: Network Specification of Battery Electric Vehicle (BEV) available for HVAC-Controller

filter	0x000010000		filter	11100x10000	
ID	00000010000	(0x010)	ID	11100010000	(0x710)
ID	01000010000	(0x210)	ID	11100110000	(0x730)
filter	11xx1110000		filter	x1000100000	
ID	11011110000	(0x6F0)	ID	01000100000	(0x220)
ID	11101110000	(0x770)	ID	11000100000	(0x620)
filter	00001101010		filter	00010000001	
ID	00001101010	(0x06A)	ID	00010000001	(0x081)
filter	11000010011				
ID	11000010011	(0x613)			

Table 3: Perfect Filtering for HVAC-controller (using 7 rather than 11 Filters)

The SA algorithm requires knowledge of the IDs of both desired messages and undesired messages. This presents a potential issue when the system is extended later. New, previously *unknown*

messages may be undesired from the perspective of a node whose operation is unchanged; however, some of them may pass through the node's existing hardware filters, leading to additional processor load. In general, this issue can only be avoided completely if a separate filter is available for each desired message. Nevertheless, the use of Algorithm 1 within the SA-based solution locally optimizes each filter for the desired messages assigned to it. This has the effect of minimizing the number undesired *and* unknown messages that can pass through. As an example, with 3 filters, 1818 (91.2%) out of the 1993 unknown messages (i.e. unused IDs) are blocked, while with 7 filters, 1991 (99.9%) are blocked. In future, we aim to explore how filter configurations can be designed with extensibility in mind.

7 SUMMARY AND CONCLUSIONS

In this paper we presented the problem of configuring CAN message acceptance filters. We formally specified the problem, provided a metric for determining the quality of a filter configuration, analysed the problem complexity, and provided a solution based on Simulated Annealing. Further, we provided optimal algorithms for solving two specific special cases. A large-scale evaluation, based on synthetic examples, showed the effectiveness of the SA-based solution to the general problem, with significant performance improvements over a simple engineering heuristic. Finally, we demonstrated the practicality of the approach via an industrial case study. Here we were able to achieve perfect filtering using only 7 receive buffers and filters, compared to 11 in the original implementation, thus providing headroom for future upgrades in the form of extra transmitted or received messages.

The work in this paper considered the common industrial constraint where message IDs are fixed. When there is freedom to configure messages IDs, as in the case with Volcano[4], then a simple scheme can be employed to achieve perfect filtering in hardware provided that 29-bit IDs are used, as discussed in Section 1. In future, we aim to look at the joint problem of message ID assignment and filter configuration for systems that use 11-bit identifiers. This problem is more challenging, since the solution may need to compromise between a priority ordering that is desirable for network schedulability and an ID assignment that enables a high level of message filtering in hardware.

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 European Union's 7th Framework Programme project epsilon (grant agreement no 605460), the EPSRC project MCCps (EP/P003664/1) and the INRIA International Chair program. EPSRC Research Data Management: No new primary data was created during this study.

REFERENCES

[1] ATMEL. 2008. AT90CAN128. (2008). Data-sheet: http://www.atmel.com/images/doc7679.pdf.

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

[3] Bosch. 1991. Controller Area Network Specification 2.0. (1991).

[4] D. Buttle. 2012. Real-Time in the Prime Time. In *Euromicro Conference on Real-Time Systems (ECRTS).* Keynote.

[5] R.I. Davis and A. Burns. 2007. Robust Priority Assignment for Fixed Priority Real-Time Systems. In *IEEE Real-Time Systems Symposium (RTSS).* 3–14.

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

[7] R.I. Davis, A. Burns, V. Pollex, and F. Slomka. 2015. On Priority Assignment for Controller Area Network when some Message Identifiers are Fixed. In *International Conference on Real-Time Networks and Systems (RTNS).* 279–288.

[8] R.I. Davis, S. Kollmann, V. Pollex, and F. Slomka. 2013. Schedulability analysis for Controller Area Network (CAN) with FIFO queues priority queues and gateways. *Real-Time Systems* 49, 1 (2013), 73–116.

[9] R.I. Davis and N. Navet. 2012. 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).* 33–42.

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

[11] M. Di Natale, H. Zeng, P. Giusto, and A. Ghosal. 2012. *Understanding and using the controller area network communication protocol: theory and practice.* Springer Science & Business Media.

[12] P. Emberson and I. Bate. 2008. Extending A Task Allocation Algorithm For Graceful Degradation Of Real-Time Distributed Embedded Systems. In *29th Real-Time Systems Symposium (RTSS).* 270–279.

[13] P. Emberson and I. Bate. 2010. Stressing Search with Scenarios for Flexible Solutions to Real-Time Task Allocation Problems. *IEEE Transaction on Software Engineering* 36, 5 (2010), 704–718. https://doi.org/10.1109/TSE.2009.58

[14] Freescale. 2000. MPC555. (2000). Data-sheet: http://www.nxp.com/assets/documents/data/en/data-sheets/MPC555UM.pdf.

[15] Freescale. 2000. msCAN for HC08. (2000). Data-sheet: http://www.datasheetlib.com/datasheet/850324/an2010_motorola-semiconductor.html.

[16] Freescale. 2000. Using The Motorola msCAN Filter Configuration Tool. (2000). Application-note: http://www.datasheetlib.com/datasheet/850324/an2010_motorola-semiconductor.html.

[17] Freescale. 2005. msCAN for S08. (2005). Data-sheet: https://www.nxp.com/files-static/training_pdf/29041_S08_MSCAN_WBT.pdf.

[18] Infineon. 2015. Controller Area Network Controller (MultiCAN). (2015). Application-note: http://www.infineon.com/dgdl/Infineon-MultiCAN-XMC4000-AP32300-AN-v01_00-EN.pdf?fileId=5546d4624e765da5014ed91d6be32110.

[19] R. M. Karp. 1972. *Reducibility among Combinatorial Problems.* Springer US, 85–103. https://doi.org/10.1007/978-1-4684-2001-2_9

[20] D. A. Khan, R. I. Davis, and N. Navet. 2011. Schedulability analysis of CAN with non-abortable transmission requests. In *IEEE Emerging Technologies & Factory Automation (ETFA).* 1–8.

[21] National. 2002. CR16. (2002). Data-sheet: http://www.farnell.com/datasheets/46979.pdf.

[22] N.C. Audsley. 2001. On priority assignment in fixed priority scheduling. *Inform. Process. Lett.* 79, 1 (2001), 39–44.

[23] F. Pölzlbauer, I. Bate, and E. Brenner. 2013. On Extensible Networks for Embedded Systems. In *IEEE International Conference and Workshops on the Engineering of Computer Based Systems (ECBS).* 69–77.

[24] F. Pölzlbauer, R. I. Davis, and I. Bate. 2016. A Practical Message ID Assignment Policy for Controller Area Network that Maximizes Extensibility. In *24th International Conference on Real-Time Networks and Systems (RTNS).*

[25] Renesas. 2006. M16C. (2006). Data-sheet: http://www.symmetron.ru/suppliers/renesas/pdf/rej09b0101_16c29hm.pdf.

[26] K. W. Schmidt. 2014. Robust Priority Assignments for Extending Existing Controller Area Network Applications. *IEEE Transactions on Industrial Informatics* 10, 1 (2014), 578–585. https://doi.org/10.1109/TII.2013.2266636

[27] K. W. Tindell, A. Burns, and A. J. Wellings. 1992. Allocating Hard Real-time Tasks: An NP-hard Problem Made Easy. *Real-Time Syst.* 4, 2 (May 1992), 145–165. https://doi.org/10.1007/BF00365407

[28] Q. Zhu, Y. Yang, E. Scholte, M. D. Natale, and A. Sangiovanni-Vincentelli. 2009. Optimizing Extensibility in Hard Real-Time Distributed Systems. In *15th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS).* 275–284. https://doi.org/10.1109/RTAS.2009.37

[29] Q. Zhu, H. Zeng, W. Zheng, M. DI Natale, and A. Sangiovanni-Vincentelli. 2012. Optimization of task allocation and priority assignment in hard real-time distributed systems. *ACM Transactions on Embedded Computing Systems (TECS)* 11, 4 (2012), 85.

[30] W. Zimmermann and R. Schmidgall. 2008. *Bussysteme in der Fahrzeugtechnik: Protokolle und Standards* (3 ed.). Vieweg+Teubner.

[4] See https://www.mentor.com/products/vnd/in-vehicle_software/