Priority-Preserving Optimization of Status Quo ID-Assignments in Controller Area Network

Sebastian Schwitalla
Humboldt-Universität zu Berlin
Berlin, Germany
sebastian.schwitalla@informatik.hu-berlin.de

Lea Schönberger
TU Dortmund University
Dortmund, Germany
lea.schoenberger@tu-dortmund.de

Jian-Jia Chen
TU Dortmund University
Dortmund, Germany
jian-jia.chen@tu-dortmund.de

***Abstract*—Controller Area Network (CAN) is the prevailing solution for connecting multiple electronic control units (ECUs) in automotive systems. Every broadcast message on the bus is received by each bus participant and introduces computational overhead to the typically resource-constrained ECUs due to interrupt handling. To reduce this overhead, hardware message filters can be applied. However, since such filters are configured according to the message identifiers (IDs) specified in the system, the filter quality is limited by the nature of the ID-assignment. Although hardware message filters are highly relevant for industrial applications, so far, only the optimization of the filter design, but not the related optimization of ID-assignments has been addressed in the literature.**

In this work, we explicitly focus on the optimization of message ID-assignments against the background of hardware message filtering. More precisely, we propose an optimization algorithm transforming a given ID-assignment in such a way that, based on the resulting IDs, the quality of hardware message filters is improved significantly, i.e., the computational overhead introduced to each ECU is minimized, and, moreover, the priority order of the system remains unchanged. Conducting comprehensive experiments on automotive benchmarks, we show that our proposed algorithm clearly outperforms optimizations based on the conventional method simulated annealing with respect to the achieved filter quality as well as to the runtime.

***Index Terms*—Controller Area Network, broadcast bus, message filtering, priority order, automotive, real-time, design optimization**

I. Introduction

In the automotive sector, Controller Area Network (CAN) [1] is the prevailing solution for connecting multiple electronic control units (ECUs). Via this bus, each ECU is able to transmit data such as, e.g., the vehicle speed, motor temperature, and break pressure. All broadcast messages can by default be received by any other bus participant, but not every message is equally relevant to each of them. More precisely, only a subset of incoming messages is relevant to each ECU *(desired messages)*, so that it is sensible to reject all irrelevant messages *(undesired messages)* in order to reduce the computational overhead for handling the interrupt triggered at each message arrival. For this purpose, CAN controllers are typically equipped with configurable hardware message filters, which allow to specify a certain bit-pattern that is matched with the incoming messages. Resulting from this, the respective message is either accepted or blocked.

Although hardware message filters are highly relevant not only for automotive, but also for robotic systems, they have been addressed rather exceptionally. In 2017, Pölzlbauer et al. [2] published the first work specifically examining the design of hardware message filters for CAN, proposing a metric to measure the quality of such filters by accumulating the frequencies of all undesired messages accepted by a certain bus participant, which is defined as the *Quality of Filter* $QoF = \sum_{m \in M_s^U} \frac{[m \text{ is accepted}]}{T_m}$, where M_s^U denotes the set of undesired messages of a bus participant p_s, T_m the period of a periodic message $m \in M_s^U$, and $[m$ is accepted$]$ the Iverson bracket. The filter quality is optimal if the *QoF* is 0. Moreover, in their work, a set of heuristics was proposed to optimize the *QoF*. In 2019, Schönberger et al. [3] applied satisfiability modulo theories (SMT) to provide a more effective approach for solving this problem and proved the optimality of the resulting filter configuration for one common industrial setting.

De facto, the outcome of any of the above approaches is highly dependent on the identifiers *(IDs)* of the messages defined in the considered system. In CAN, each message holds a unique ID, i.e., a fixed-length bit-string (11 bits in the standard format, 29 bits in the extended format), which specifies the priority of a message and thus determines, which message wins the bus arbitration if multiple participants intend to send at the same time. Since IDs are unique, they are used for the hardware-based filtering process as well. If, however, the message IDs are not chosen carefully enough, e.g., according to a naive heuristic or randomly, the amount of filters required to achieve an optimal *QoF* is increased drastically and, in consequence, the best *QoF* achievable under hardware restrictions is significantly higher [3], i.e., worse.

In this work, we focus on the design of message IDs, aiming to decrease the computational overhead of each ECUs introduced by undesired but accepted messages. However, filter design must not be the main objective when assigning IDs. In fact, it is of utmost importance to preserve the priority order of messages to ensure that the system is not compromised. In a nutshell, we contribute the following:

- We propose an algorithm by means of which the message ID-assignment in a system can be optimized in such a way that the computational overhead introduced by

This work was partly supported by Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center SFB 876, project A3.

undesired but accepted broadcast messages is minimized for each bus participant, while the message priority order remains unchanged.
- We show by means of comprehensive experiments based on automotive benchmarks that our proposed approach reduces the *QoF* significantly.
- We point out the advantage of our optimization method in terms of runtime compared to simulated annealing.

II. System Model and Message Filtering

Consider a controller area network connecting a set of bus participants $P = \{p_1, \ldots, p_n\}$ with $n \in \mathbb{N}$, which are able to broadcast and receive messages. A broadcast message $m \in M$ is defined as a tuple $m = (b_m, T_m)$, where b_m denotes its unique message ID, which is represented as a bit-vector of length H, and T_m its period. According to the specifications of CAN [1][1], b_m is restricted to integer values within the interval $[0, 2031]$ in the standard format and within $[0, 532676607]$ in the extended format. All other parameters of a broadcast message are neglected at this point, since they are not relevant for the purpose of this work.

Let $\pi(m)$ indicate the priority of a message $m \in M$, which is encoded in b_m. For two messages $m_\alpha, m_\beta \in M$, $\pi(m_\alpha) > \pi(m_\beta)$ if m_α has higher priority than m_β. Nevertheless, according to the specifications of CAN [1], $b_\alpha < b_\beta$ if $\pi(m_\alpha) > \pi(m_\beta)$. For a set of messages M, a total order according to the message priorities *(priority order)* is given, i.e., $M = \{m_1, m_2, \ldots, m_{|M|}\}$ such that $\pi(m_1) < \pi(m_2) < \cdots < \pi(m_{|M|})$, denoted as $\Pi(M)$.

For each $p_s \in P$, a set of *desired messages* M_s^D as well as a set of *undesired messages* M_s^U are defined, such that $M_s^D \cup M_s^U = M$ and $M_s^D \cap M_s^U = \emptyset$. In order to reduce the number of undesired messages that need to be filtered out by software, each participant $p_s \in P$ is equipped with a set of hardware message filters $F = \{f_1, \ldots, f_n\}$, termed *filter configuration*. Each filter f is described by a tuple (x_i, Y_i) with $i \in \mathbb{N}$, where x_i is a *mask* and $y_{i,k}$ with $k \in \mathbb{N}$ is a set of *tags* $y_{i,k} \in Y_i$. Masks as well as tags are bit-vectors of length H, which can be configured in such a way that they represent specific bit-patterns that are matched with the ID of each incoming message. Based on the matching result, a message is either accepted or rejected. More precisely, an incoming message m is accepted if it holds that $b\_and(x_i, y_{i,k}) = b\_and(x_i, b_m)$, where b_and is a bit-wise and operation, and rejected otherwise.

For clarification, consider the filtering examples given in Table I and Table II: The particular contemplated bus participant p_s is equipped with one hardware message filter consisting of one mask x_1 with one tag $y_{1,1}$. The filter pattern of f_1 is retrieved as follows: All bits, which are not considered in the filtering process, are set to 0 in x_1. The corresponding bits in f_1 are indicated by a *don't care* $*$, i.e., no specific value is required at this bit-position of an incoming message ID. For all bits, which are marked by 1 in x_1, the required bit-value is indicated in $y_{1,1}$. For more detailed information regarding the filtering process, please refer to [2] and [3].

[1]According to the specifications, the 7 most significant bits of an ID must not be recessive, i.e., no ID must start with the pattern 1111111.

TABLE I
A filtering example with imperfect filter configuration.

	bit-vector	desired?	accepted?		bit-vector
b_1	001 111 000 00	yes	yes		
b_2	100 000 001 01	yes	yes	x_1	010 000 000 00
b_3	100 000 010 10	yes	yes	$y_{1,1}$	000 000 000 00
b_4	100 000 100 00	yes	yes	f_1	*0* *** *** **

TABLE II
A filtering example with perfect filter configuration.

	bit-vector	desired?	accepted?		bit-vector
b_1	100 000 000 00	yes	yes		
b_2	100 000 000 01	yes	yes		
b_3	100 000 000 10	yes	yes	x_1	111 111 111 00
b_4	100 000 000 11	yes	yes	$y_{1,1}$	100 000 000 00
b_5	100 000 010 11	no	no	f_1	100 000 000 **

If a filter configuration accepts all $m \in M_s^D$, it is termed *correct*. Moreover, if it accepts all $m \in M_s^D$ and rejects all $m \in M_s^U$, it is denoted *perfect*; otherwise, if not all $m \in M_s^U$ are rejected, the filter configuration is classified as *imperfect*.

III. Problem Definition

Although perfect filter configurations are preferable, they cannot always be achieved. This follows from the hardware limitation of a system as well as substantially from the design of its message IDs. More precisely, if the bit-patterns of message IDs vary largely within a set M_s^D, it is hardly possible to specify particular bit-values in the tag, so that multiple bits need to be set to *don't cares* in the resulting filter. Exempli gratia, reflect upon Table I: Due to the ID-design of M_s^D and the hardware restrictions of the contemplated bus participant p_s (one filter consisting of one mask x_1 with one tag $y_{1,1}$), only one bit-value can be specified, whereas all other bits are set to $*$. In consequence, the filter configuration is correct, but imperfect, since a great share of possibly existing messages can pass besides M_s^D. This, indeed, leads to a very poor *QoF* and introduces computational overhead, since all messages accepted by the filter must be handled in software. In Table II, however, a set M_s^D with more beneficial ID-design is considered. Since the IDs of all $m \in M_s^D$ vary only in two bits, more values can be precisely specified in $y_{1,1}$ and thus in f_1 than in Table I. Resulting from this, a perfect filter configuration is obtained.

The problem studied in this paper is defined as follows:
Input: Given a set of broadcast messages M with a total order of the priorities Π and a set of bus participants P with M_s^D and M_s^U $\forall p_s \in P$.
Output: Assign $b_m \forall m \in M$ such that for any two different messages $m_\alpha, m_\beta \in M$, $\pi(m_\alpha) > \pi(m_\beta)$ if m_α has higher priority than m_β.
Objective: Minimize the *QoF*.

IV. ID-Optimization

As illustrated above, it is desirable to design message IDs in such a way that for each $p_s \in P$ the IDs of all $m \in M_s^D$

exhibit a certain *similarity*. If IDs are not assigned accordingly, it is very likely that filter configurations cannot be obtained with reasonable *QoF*. To avoid design pitfalls as depicted in Table I, it is sensible to take hardware message filtering into consideration already at system and message design time. On this account, we subsequently propose two approaches that allow to optimize existing ID-assignments accordingly.

A. Set Code

The more filter-bits can be precisely specified instead of setting *don't cares*, the lower, i.e., the better, the *QoF* that can be obtained, as explained by means of the examples in Table I and Table II. This follows from the fact that a specific required bit-value reduces the number of possibly accepted messages compared to a *don't care*, resulting from which the respective bit is not considered for the filtering process at all. However, specifying as many filter-bits as possible does not necessarily lead to a rejection of all undesired messages. Depending on the ID-design, it may be the case that, although most filter-bits are specified, a message $m \in M_s^U$ matches exactly with the required bit-pattern. To prevent incidents of this kind, it is sensible to design message IDs in such a way that the IDs of all $m \in M_s^D$ for each $p_s \in P$ include a characteristic bit-pattern that must not occur within the ID of any $m \in M_s^U$ and can therefore be used for filtering. Subsequently, we explain how to integrate such patterns into existing IDs without falsifying the message priority and violating the priority order Π.

To enable system engineers to modify IDs and to simultaneously ensure that $\Pi(M') = \Pi(M)$, it is advisable to reserve a dedicated range of bits for encoding the message priority, denoted as the *priority-preserving* part b_m^π of an ID b_m. The number of bits available for the priority-preserving part depends on the ID length H (11 bits in the standard format, 29 bits in the extended format) as well as on length of the so-called *set code*, i.e., a characteristic bit-pattern used for message filtering only. A schematic representation of a message ID b_m split into b_m^π and b_m^p is given in Fig. 1.

When defining the set code of a b_m, it is not recommendable to enumerate the sets of desired messages of all $p_s \in P$ and to choose the corresponding number of M_s^D with $p_s \in P$. This follows from the fact that one message can be desired by more than one participant and therefore can be contained in multiple sets of desired messages. Consequently, a message could be assigned multiple set codes and thus multiple IDs, which does not comply with the specifications of controller area network [1]. Instead, a set of disjunct message sets $\mathbb{S} = \{S_1, S_2, \ldots\}$ can be obtained by intersecting the sets of desired messages of all $p_s \in P$, as illustrated in Fig. 2. Each of the resulting $S_\ell \in \mathbb{S}$ is thereon assigned a unique set code.

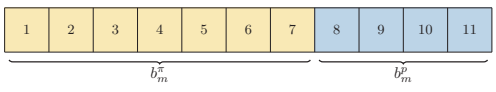

Fig. 1. CAN message ID in standard format split into set code b_m^p and priority-preserving part b_m^π.

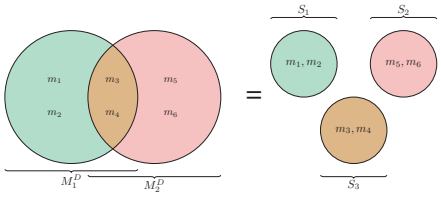

Fig. 2. Set $\mathbb{S}$ of sets of desired messages and their nonempty intersections.

The set code algorithm is given in pseudo-code in Fig. 3 and operates as follows: As the input, the set of messages M defined in the system sorted decreasingly with respect to its priority order $\Pi(M)$, the maximum ID length H (cf. Sec. II) and the maximum legal ID according to the CAN specifications b_{CAN} are given. Initially, the set $\mathbb{S}$ of disjunct message sets is computed (cf. l. 4). Based on $\mathbb{S}$, the number of bits required for encoding the set code, denoted as $len(p)$, as well as the number of bits remaining for the priority-preserving part of each message, denoted as $len(\pi)$, are calculated (cf. ll. 5-6). Thereon, it is determined if the input data set M is suitable to be optimized by the set code approach (l. 7). More precisely, it is verified if the number of bits available for encoding the set code suffices to provide a unique bit-pattern for each message set $S_{cntr} \in \mathbb{S}$ and if the number of bits remaining for the priority-preserving part allows to cover all required priority levels. If one of these conditions is not satisfied, M cannot be optimized using the set code approach. In this case, the optimization approach discussed in Sec. IV-B can be applied. To create a set code for each message set $S_{cntr} \in \mathbb{S}$, the sets are enumerated and their ordinal numbers are assigned to all messages contained in the respective sets (cf. ll. 8-12). To generate the priority-preserving part of each ID b_m^π, equally spaced integers are chosen out of a range from 0 to the maximum value that can be encoded by $len(\pi)$ bits. For this purpose, first a step size is computed (cf. l. 13), which is then multiplied with the index of each message $m \in M$ (cf. ll. 14-19). Finally, it is checked if the resulting ID does not violate the specifications of CAN, i.e., if the ID resulting of the concatenation of priority-preserving part and set code does not exceed the maximum legal CAN ID b_{CAN} (cf. l. 16-18). In the successful case, a set of messages M' with optimized IDs is retrieved as the output with $\Pi(M') = \Pi(M)$ (cf. l. 20).

B. Simulated Annealing

If a given data set is not suitable to be optimized by the set code approach (cf. Sec. IV-A), it is possible to apply simulated annealing, a heuristic approximation technique for solving optimization problems that simulates the process of cooling heated material until a freezing point (for details refer to [4]). In the course of this simulated annealing process, it is necessary to evaluate the similarity of distinct IDs and to assess the quality of different ID-assignments. For this purpose, we henceforth define a number of so-called *similarity metrics*.

Aiming to formally express the similarity of two message IDs, we benefit from the fact that each ID is a bit-vector of length H, which enables us to compare the elements of two

```
Input: M with total priority order (sorted decreasingly),
ID length H, maximum ID l_CAN
Output: M' with optimized IDs
create a set 𝕊 of disjunct message sets
len(p) ← log_2(|𝕊|)
len(π) ← H − len(p)
if |𝕊| ≤ 2^(len(p)−1) and |M| ≤ 2^(len(π)−1) then
    for cntr = 0; cntr ≤ |𝕊|; cntr++; do
        for each m ∈ S_cntr do
            b^p_m ← cntr
        end for
    end for
    stepsize ← ⌊2^len(π) / |M|⌋
    for each m ∈ M do
        b^π_m ← stepsize · index(m)
        if concat(b^π_m, b^p_m) > b_CAN then
            return 'not suitable for this data set'
        end if
    end for
    return M' with optimized IDs
else
    return 'not suitable for this data set'
end if
```

Fig. 3. A pseudo-code representation of the set code algorithm.

vectors b_α and b_β at a particular index based on their exclusive logical sum, i.e., applying the xor operator, which is 0 if the respective bits match, and 1 otherwise. By accumulating the exclusive logical sums for all H bit-indices, the number of bits in which the IDs differ can be determined. We denote this metric as the *binary distance* between two messages, which is defined as follows:

Definition 1 ($b\_dist(b_\alpha, b_\beta)$): For two messages $m_\alpha, m_\beta \in M$, the binary distance between their message IDs b_α and b_β is defined as $b\_dist(b_\alpha, b_\beta) = ||b\_xor(b_\alpha, b_\beta)||_1$, where b_xor is the bit-wise *xor* operation and, for a vector $v = (v_1, \ldots, v_n), n \in \mathbb{N}$, $||v||_1$ is the 1-norm given as $||v||_1 = \sum_{i=1}^{n} |v_i|$.

Based on the binary distance, it is possible to make a statement regarding the similarity of *two* IDs. If, however, the similarity of *a larger set* of IDs M is the subject of interest, the binary distance $b\_dist(b_\alpha, b_\beta)$ can be evaluated and added up for all $b_\alpha, b_\beta \in M, b_\alpha \neq b_\beta$. We denote this as the *distance* of the considered set of messages M:

Definition 2 ($distance(M)$): For a set of messages M, the distance between the ID of each message $m_\alpha \in M$ and the ID of each message $m_\beta \in M$, where $\alpha \neq \beta$, denoted as *distance of* M, is defined as $distance(M) = \sum_{\alpha=1}^{|M|-1} \sum_{\beta=\alpha+1}^{|M|} b\_dist(b_\alpha, b_\beta)$.

Besides $distance(M_s^D)$, another factor is of interest here, namely, the similarity of IDs within two distinct sets or, more specifically, the similarity of all $m \in M_s^D$ to all $m \in M_s^U$ for a participant p_s. This value can be determined by accumulating the binary distance of each $m \in M_s^D$ to each $m \in M_s^U$ and is denoted as *distance* between the set M_s^D and the set M_s^U:

Definition 3 ($distance(M_s^D, M_s^U)$): For two sets of messages M_s^D, M_s^U with $M_s^D \cap M_s^U = \emptyset$ and $p_s \in P$, the distance between the ID of each message $m_\alpha \in M_s^D$ and the ID of each message $m_\beta \in M_s^U$, denoted as *distance between* M_s^D *and* M_s^U, is defined as $distance(M_s^D, M_s^U) = \sum_{\alpha=1}^{|M_s^D|} \sum_{\beta=1}^{|M_s^U|} b\_dist(b_\alpha, b_\beta)$.

Making use of these similarity metrics, simulated annealing can be applied to optimize a given ID-assignment. As explained in Sec. III, it is sensible that the IDs of all $m \in M_s^D$ differ as little as possible from each other, i.e., exhibit a high similarity. Consequently, it is desirable to minimize $distance(M_s^D)$ for all $p_s \in P$. Adopting the idea from Sec. IV-A that a certain pattern used for filtering all $m \in M_s^D$ must not occur in the ID of any $m \in M_s^U$, i.e., that similarities between IDs of desired and undesired messages of a $p_s \in P$ should be avoided, it is recommendable to maximize $distance(M_s^D, M_s^U)$ for all $p_s \in P$.

The usefulness of this general-purpose optimization approach will be evaluated with respect to the quality of the resulting ID-assignment as well as to the runtime in the course of comprehensive experiments in Sec. V.

V. Evaluation

Having elucidated how to optimize the ID-assignment of a system with respect to the design of hardware message filters, we henceforth investigate to which extent the optimization objective is satisfied, i.e., how much the *QoF* of hardware message filters designed based on the resulting ID-assignments and thus the computational overhead introduced by undesired but accepted messages is reduced. In the following, we first explain our experiment setup in Sec. V-A and describe the considered benchmarks in Sec. V-B, before we finally discuss the outcome of our evaluation in Sec. V-C.

A. Experiment Setup

All experiments were performed on a machine with Xeon X5650 processor (12 cores, 2 threads per core, 2.67GHz) and 56473MB RAM. As a hardware message filter setup, we considered the Microchip MCP2515 [5], a low-cost CAN controller compatible with standard and extended message format, which provides 2 masks: one with 2 and one with 4 associated tags. For the sake of clarity and representability, however, we restrict our investigations to the mask with 4 tags.

We examined the proposed optimization algorithms under various network settings, out of which we present the most expressive ones: i) 25 messages in standard format (11 bit IDs) in a network of 3 bus participants, ii) 50 messages in standard format in a network of 5 bus participants, iii) 100 messages in extended format (29 bit IDs) in a network of 10 bus participants, and iv) 200 messages in extended format in a network of 20 bus participants. For each setting, the experiment was repeated 10 times.

For the purpose of comparison, we implemented an optimization approach using simulated annealing with the similarity metrics defined in Sec. IV-B. In each iteration of simulated annealing, the IDs of $\max(1, \lfloor |M| \cdot 0.1 \rfloor)$ arbitrarily chosen messages are modified randomly such that the prespecified priority order as well as the CAN specifications are not violated. After a manual tuning phase, we set the initial temperature, i.e., the temperature a heated material on which

the simulation process is based has at the beginning of the optimization process, the freezing point, i.e., the temperature to which the heated material is cooled down, the cooling factor, i.e., the temperature decrement rule, as follows (for details refer to [4]): *initial temperature* = 1000, *freezing point* = 10, *cooling factor* = 0.99. The number of iterations was chosen depending on the size of each particular data set.

The optimal filter configurations under the respective ID-assignments were computed based on the SMT formulation provided in [3] using the Z3 solver from Microsoft [6].

B. Data

The benchmarks to be optimized by our proposed approaches have been obtained using Netcarbench 3.4 [7], a GPL-licensed software for the generation automotive message sets. Although a broad range of parameters is provided within the created data sets, only message IDs as well as message periods are considered in the experiments. Since Netcarbench does not support the case that one message is desired by multiple bus participants, we modified the data sets accordingly. Due to the lack of real industrial data available for research purposes, we solved this problem by specifying a threshold of 0.4 and generating a random number in the interval $[0, 1]$ for each message. If the number was lower than the threshold, the respective message was randomly inserted into two sets of desired messages, otherwise in one. Netcarbench only supports messages in standard format, so that we created data sets in extended format by randomly creating 29 bit IDs for a set of messages in standard format such that the original priority order was maintained. All data sets used for the evaluation of our optimization approaches will be published with the release of this paper.

C. Results

From the outcome of the conducted experiments, it is evident, that the set code approach is more efficient than the simulated annealing optimization in all test settings with respect to the quality of the resulting ID-assignments, i.e., to the *QoF* of hardware message filters configured based on the respective IDs, as well as regarding the runtime.

The depicted results regarding the *QoF* must be understood as follows: The lower the *QoF*, the better, since the *QoF* factors in the periods of all undesired but accepted messages (cf. Sec. I). However, since every two data sets contain different messages and are designed for different networks, *QoF* values for two data sets cannot be directly compared, but rather indicate a tendency. Moreover, a reduction of the *QoF* by a certain percentage does not imply that a respective percentage of undesired messages is blocked in addition.

In Fig. 4, it is discernible that the set code approach allows to generate IDs that lead to an optimal *QoF*, i.e., to $QoF = 0$, for a data set with 25 messages in standard format, whereas simulated annealing only reduces the *QoF* to approximately half of the original *QoF*, i.e., the *QoF* resulting from unoptimized IDs. For a data set of 50 messages in standard format, the set code approach also reduces the *QoF* significantly, although not to 0, which results from the limited

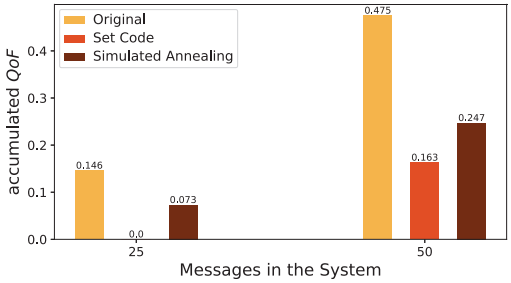

Fig. 4. The best *QoF* (accumulated over all bus participants) achievable under the ID-assignments optimized by the set code approach and simulated annealing for CAN IDs in standard format (11 bits).

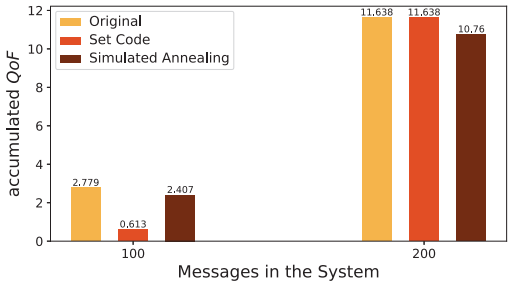

Fig. 5. The best *QoF* (accumulated over all bus participants) achievable under the ID-assignments optimized by the set code approach and simulated annealing for CAN IDs in extended format (29 bits). Please note that the data set comprising 200 messages cannot be optimized by the set code approach.

number of hardware filters in the considered scenario (cf. Sec. V-A), while simulated annealing again achieves only a reduction to circa half of the original *QoF*.

For a data set of 100 messages in extended format, the gap between the quality of the ID-assignments resulting from the set code approach and from simulated annealing is even more clearly visible, as illustrated in Fig. 5. While the IDs retrieved by simulated annealing just barely improve the *QoF* of the hardware message filters, the set code approach leads to a high reduction. Under a setting with 200 messages, however, no reduction of the *QoF* by means of the set code approach is perceptible, which follows from the fact that the contemplated data set is not suitable to be optimized by this method (cf. Sec. IV-A). Here, simulated annealing leads to at least a moderate improvement of the *QoF*.

Observing the runtimes of both approaches represented in Fig. 6, it becomes apparent that the set code approach clearly outperforms simulated annealing in all test cases, except for a data set of 200 messages, to which it cannot be applied.

Overall, the set code approach outclasses simulated annealing, whenever both methods are applicable. To understand, why the set code approach especially surpasses simulated annealing regarding the reduction of the *QoF*, we make use of the similarity metrics introduced in Sec. IV-B. Although these metrics are specified to be used within the simulated annealing optimization process, they can also serve for the purpose of comparing different ID-assignments for one set of messages.

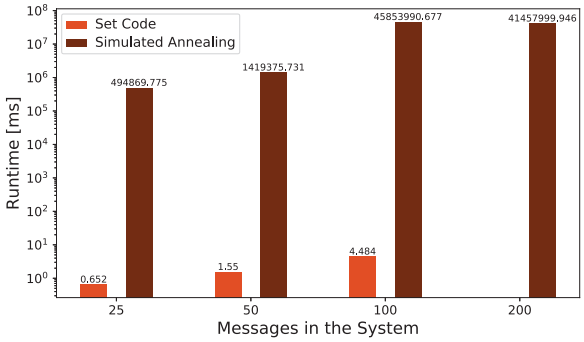

Fig. 6. The runtimes the set code approach and simulated annealing for all considered network settings. Please note that the set code approach is not applicable to the data set comprising 200 messages.

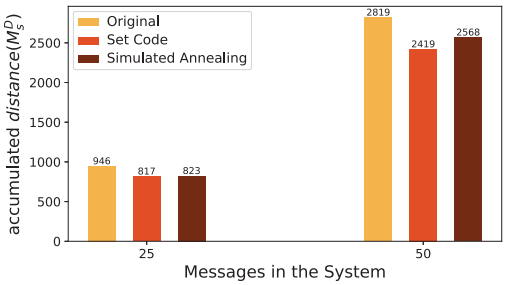

Fig. 7. The $distance(M_s^D)$ accumulated over all $p_s \in P$ for the original ID-assignment and those optimized by the set code approach and simulated annealing.

In Fig. 7, the accumulated $distance(M_s^D)$ over all $p_s \in P$ is portrayed for sets of 25 and 50 messages in standard format. Here, it is recognizable that the set code approach increases $distance(M_s^D)$ to a much higher extent than simulated annealing, i.e., that the IDs created by the set code approach exhibit a much higher similarity within each M_s^D than those retrieved by simulated annealing. In addition, Fig. 8 shows the accumulated $distance(M_s^D, M_s^U)$ over all $p_s \in P$. Surprisingly, it is noticeable that the set code approach increases the $distance(M_s^D, M_s^U)$, i.e., decreases the similarity between desired and undesired messages, for a data set of 25 messages, while simulated annealing decreases it. This may result from the individual characteristics of the respective data sets and underlying networks. For a data set of 50 messages, however, both approaches decrease $distance(M_s^D, M_s^U)$, i.e., increase the similarity between desired and undesired messages.

Combining this observation and the results illustrated in Fig. 4, which, as discussed above, indicate an improvement of the *QoF* in all contemplated cases, it can be concluded that for the design of hardware message filters rather $distance(M_s^D)$ is relevant, while $distance(M_s^D, M_s^U)$ can be neglected. More precisely, this reveals that it is of higher importance that all messages within one set of desired messages are highly similar than that the IDs of the messages within each M_s^D and M_s^U are extremely different.

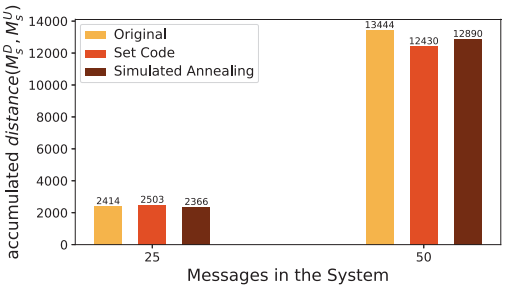

Fig. 8. The $distance(M_s^D, M_s^U)$ accumulated over all $p_s \in P$ for the original ID-assignment and those optimized by the set code approach and simulated annealing.

VI. Conclusion

In this work, we addressed the issue of optimizing status quo ID-assignments against the background of hardware message filtering in controller area network, aiming to reduce the overhead introduced to resource-constrained ECUs by undesired but accepted broadcast messages. De facto, we proposed an optimization algorithm in Sec. IV-A, that allows to transform a given ID-assignment in such a way that the *QoF* of hardware message filters generated according to the modified ID-assignment is optimized, while the priority order of the system is not changed. We showed by means of comprehensive experiments using automotive benchmarks that our proposed approach clearly outperforms simulated annealing, a common optimization method, with respect to the obtained results as well as to the runtime. Nevertheless, if a data set does not comply with the requirements of the set code approach, simulated annealing can be applied, but, however, leading to only a minor reduction of the *QoF*. Not least, we observed that, in order to configure optimal hardware message filters, it is of greater importance that the IDs of messages within a set M_s^D have a high degree of similarity than that the IDs of messages within two sets M_s^D and M_s^U differ widely.

References

[1] Bosch, "Controller Area Network Specification 2.0," 1991.

[2] F. Pölzlbauer, R. I. Davis, and I. Bate, "Analysis and Optimization of Message Acceptance Filter Configurations for Controller Area Network (CAN)," in *Proceedings of the 25th International Conference on Real-Time Networks and Systems*, ser. RTNS '17. New York, NY, USA: ACM, 2017, pp. 247–256. [Online]. Available: http://doi.acm.org/10.1145/3139258.3139266

[3] L. Schönberger, G. von der Brüggen, H. Schirmeier, and J. Chen, "Design Optimization for Hardware-Based Message Filters in Broadcast Buses," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2019, pp. 606–609.

[4] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing." *Science*, vol. 220 4598, pp. 671–80, 1983.

[5] Microchip, "MCP2515 Stand-Alone CAN Controller with SPI Interface," http://ww1.microchip.com/downloads/en/DeviceDoc/MCP2515-Stand-Alone-CAN-Controller-with-SPI-20001801J.pdf, 2018.

[6] L. de Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems*, C. R. Ramakrishnan and J. Rehof, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 337–340.

[7] Netcarbench, "Netcarbench 3.4," http://www.netcarbench.org/, 2012.