Starvation-avoidance CAN Scheduling for Shorter Worst-case Response Time with Priority Queues

Haklin Kimm
Department of Computer Science
East Stroudsburg University of Pennsylvania
East Stroudsburg, PA, 18301
hkimm@esu.edu

Jeyaprakash Chelladurai
Department of Computer Science
East Stroudsburg University of Pennsylvania
East Stroudsburg, PA, 18301
jchelladur@esu.edu

***Abstract*—Controller Area Network is a message based event-triggered communication service, and primarily applied in automotive and various industries such as vehicles, multi-robot, real-time communication systems and others. On vehicle networks, the CAN bus connects multiple independent Electronic Control Units (ECU) and allows them to communicate and work together asynchronously and/or synchronously. The ECU nodes exchange data with the CAN bus concurrently so that multiple messages collide on the bus, which is resolved by a bitwise arbitration algorithm that works by assigning a low-priority node to switch to listening mode while a node with high-priority message remains in transmitting mode. However, this arbitration mechanism results in starvation of low-priority messages. In this paper, we present a real-time scheduling algorithm that provides starvation avoidance over CAN messages.**

I. Introduction

In a crowded Control Area Network (CAN) bus system, a steady stream of higher-priority messages from multiple devices blocks low-priority messages from occupying the CAN bus for transmission. There have been studies on resolving the CAN starvation problem using two approaches. The first group of researchers has approached the CAN starvation problem with priority scheduling [1]–[3], [5]. Their main solution to reduce starvations in CAN message scheduling is through aging. In this technique, the priority of a CAN message increases gradually after repeatedly loosing its turns during the competition for bus access with other CAN messages. There has been another group of researchers use real-time non-preemptive scheduling methods to schedule CAN messages. In previous studies of applying priority scheduling, candidate messages trying to access the CAN bus are compared, and the dominant message is selected and uses CAN bus for transmission [1], [4], [6], and more than two nodes are not considered for arbitration simultaneously. However, in the studies of CAN real-time scheduling, they mainly focus on generating various CAN schedules using schedulability analysis to guarantee all messages meet their deadlines. Most of these works ignore the problem of starvation caused by CAN bus arbitration and they focus on generating feasible schedules [8]–[12]. In this work, we address the problem of starvation in CAN real-time scheduling when multiple messages from different Electronic Control Units (ECUs) are concurrently trying to win the bus arbitration for bus access. All messages losing their turn in the current arbitration round gets preferential access even if their priorities are lower than the highest priority message(s) in other ECU(s) during the next arbitration round. In our work, priorities for messages are determined by the processor in each ECU using the rate monotonic policy, although theoretically any optimal static or dynamic priority algorithms can be used with our approach.

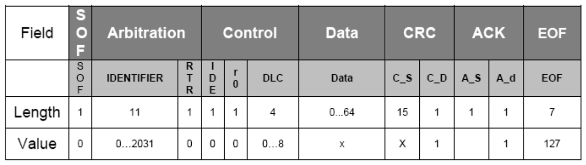
The remainder of this paper is organized as follows: Section 2 introduces Controller Area Network with its frame format of CAN 2.0 as well as arbitration method. Section 3 provides the related work on CAN scheduling regarding arbitration and starvation. Section 4 proposes and implements the proposed starvation-avoidance bit-wise arbitration algorithm applying real-time monotonic scheduling. Section 5 concludes the paper.

II. Controller Area Network (CAN)

Control Area Network is known to be very suitable for real time systems due its low cost and high reliability as a network. Since CAN uses a shared data bus, therefore it reduces a large amount of wiring between components or modules, more complex electric circuits and noise, compared to hardwired point-to-point connections. In a CAN system, CAN controllers, also known as ECUs or nodes and other components share the data bus for communication. Some salient features of CAN are prioritized bus access, reconfiguration facility, and high reliability in noisy environments through CRC checks and bit stuffing.

A data message in CAN has seven fields as shown in the Figure 1. CAN allow two message formats which can exist on the same bus. They differ in the length of the ID field. The standard format has a 11-bit ID, whereas an extended format CAN2.0B has a 29-bit ID. CAN makes use of a wired-AND to connect all the nodes (ECU's). Two logical bit representations are defined - dominant ('0') and recessive ('1'). If even a single ECU node sends a dominant bit on the bus, the bus will reflect a dominant bit, otherwise it will reflect a recessive bit. When a ECU has to send a message it first calculates the message ID based on some priority. The ID for each message must be unique to prevent a tie. The bus acquisition algorithm works as

978-1-7281-7374-0/20/$31.00 ©2020 IEEE
DOI 10.1109/WoWMoM49955.2020.00069



Field	SOF	Arbitration		Control				Data	CRC		ACK		EOF
	SOF	IDENTIFIER	RTR	IDE	r0		DLC	Data	C_S	C_D	A_S	A_d	EOF
Length	1	11	1	1	1		4	0...64	15	1	1	1	7
Value	0	0...2031	0	0	0		0...8	x	X	1		1	127

Fig. 1. Frame Format CAN 2.0

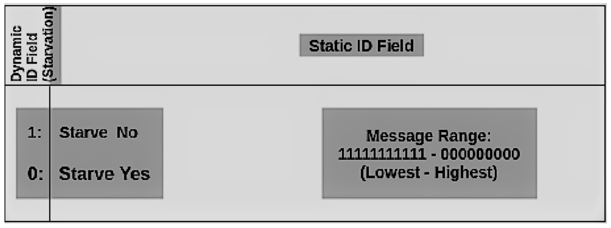


Fig. 2. Message Ranges with Dynamic and Static IDs

follows: Each ECU passes its message from the message queue along with their calculated IDs to the bus interface chips. If two or more ECUs are transmitting at the same time then a dominant bit overrides a recessive bit and the value on the CAN bus will be in the dominant state. If a chip had written a recessive bit but reads a dominant value, it means another ECU has a message with higher priority. The CAN protocol calls for ECUs to wait until a bus idle is detected before attempting to transmit again. As the message identifiers are unique, a node transmitting the last bit of the identifier field, without detecting a '0' bit that it did not transmit, must be transmitting the highest priority message that was ready for transmission at the start of arbitration. This node then continues to transmit the remainder of its message, all other nodes will back off.

III. Related works

CAN supports bounded message delays by sending highest priority message among multiple messages from different ECUs that are trying to access bus for transmission at the same time. To support message transmission with deadline guarantees on a CAN, it is essential to determine the Worst Case Response Time (WCRT) which describes the maximum response time among all the instances of a message. WCRT is determined based on the perioidic message characteristics such as priority, transmission time, and period. We need to ensure that all the messages are schedulable, or in other words WCRTs of messages are smaller than their deadlines. This is called as schedulability analysis. Algorithms for schedulability analysis were first proposed by Tindell et al., where messages are assigned priorities with 'deadline - jitter' in the monotonic order. It has been implemented by using an automotive industrial tool, Volcano Network Architect (VNA) [8], [9]. In the following studies Davis et al. provided the systematic approach of schedulability analysis, applying a suitable priority assignment policy, that was possible to meet deadline constraints of messages in CAN systems. It has also been implemented using an industrial tool, Robus-ICE [10]. Furthermore, many research studies focused on finding suitable priority assignment for CAN messages [22]–[24]. Most scheduling studies of scheduling CAN periodic and sporadic messages have been with the implementation of FIFO Queues (FQ's) or Priority Queues (PQ's) for the nodes connected to the CAN network. The main objective of these studies were to provide real-time deadline guarantees for messages using rigorous schedulability tests and they compared the WCRTs of periodic and sporadic messages. Using FQ's for message scheduling increased WCRT for higher priority messages unlike PQ's [11]. Messages with a longer WCRT implies that they are undergoing starvation due to the frequent of arrival higher priority messages. However, the WCRTs of messages is less for nodes that implements PQs than FQ's as explained later in Section V. Few other studies focused on reducing WCRTs by having offsets for message activations and they refined earlier schedulability analysis to include offsets as part of their calculations [18], [22], [24]. The main objective of assigning offsets to a message is to minimize the concurrent access to the CAN bus by decreasing simultaneous activation of messages from different ECUs. Therefore, one or few messages at any particular time compete for the bus access which reduces bus contention. This approach not only reduces the WCRT of messages, but distributes the workload over time [18]. However, computing offsets for messages are not trivial and most of these offset assignment algorithms use heuristic approaches which are time-consuming.

One of the earlier work on resolving starvation in CAN arbitration was done by employing a master node. The master node monitors CAN network all the time and remains silent as far as it receives messages from all the nodes. Master node become active if it finds that messages from some nodes are not contributing in the communication. The master node then changes the priorities of those non-contributing nodes from low to high. This ensures that messages from these nodes can access during the next arbitration round for broadcasting, making the network starvation free. The main problem is the overhead of using a master node as it should find out nodes who are unable to transmit their low-priority messages after the current arbitration. It is not determinant in finding the messages losing the current arbitration in this paper [2]. In [4], the starvation-free CAN scheme has been developed and implemented using the priority inversion scheme. In [5], CAN with flexible data rate (CAN FD) is applied to reduce the traffic congestion of current controller area network (CAN). In [7], this starvation is seen as a critical section problem with priority scheduling, where a synchronization technique is required at the entry and exit of the bus. The main contribution of this work is to provide a formal model of a starvation-free bitwise CAN arbitration protocol using non-preemptive real time scheduling methods to determine the messsage priorities. Our proposed method not only distributes the workload over time without offsets but also improves the WCRTs of lower priority messages and thereby, reduces the number of starvations.

IV. System Model and Assumptions

Each ECU transmits a set of periodic messages to other ECU's over the CAN bus. Figure 3 illustrates a CAN communication system. A set of n periodic messages from all ECU's is called a message set, $\Gamma = (m_1, m_2, ..., m_n)$. Each message m is characterized by a tuple $< C_k, T_k, D_k >$ where C is the transmission time, T is the transmission period, and D is the deadline of the message. The subscript k refers to the k^{th} instance of a periodic message, m. Note that $T_k > C_k \geq 0$. This requires that if the first instance of m is released at time t_a, the following instances are released periodically at $t_a + kT_k$, where $k = 1, 2, 3, ...$ and the message m must be allocated C units of CAN bus time in the interval $[t_a + (k-1)T_k], t_a + kT_k]$.

All message instances have a specified release time known as activation time. When a message is activated at time t and finishes at t', then t'- t is called the "response time" of the message instance. The WCRT of message m, denoted as R_m is defined as maximum possible response time among all instances $(j's)$ of the message, m. A message m is considered schedulable if and only if WCRT of the message is smaller than its transmission period i.e., $R_m \leq T$. We say that message set Γ is schedulable if $\forall m \in \Gamma$ is schedulable. The goal of any message scheduling algorithm should guarantee that all message sets Γ's are schedulable.

We now define the following:

Definition 4.1: **Hyperperiod** of a message set Γ is defined as the smallest interval of time after which the schedule repeats itself and is equal to the least common multiple of the task periods in the message set Γ [13]. If all message instances in the message set Γ are able to their deadlines within its hyperperiod, then the message set is considered schedulable.

Definition 4.2: **CAN bus utilization** is the sum of ratios C_i/T_i where i = $1, 2, ..n$ for a message set Γ.

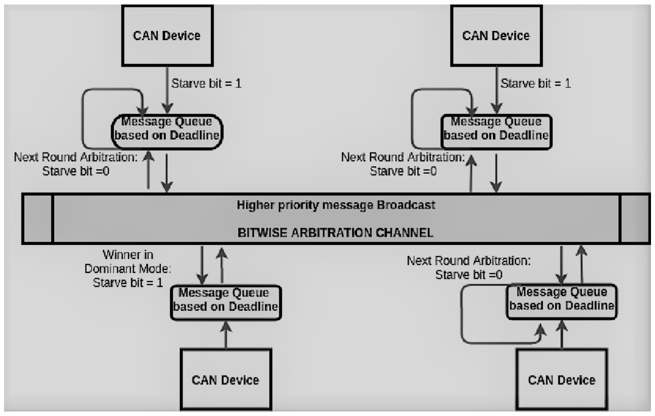
Definition 4.3: **Response time** of a message instance is defined as difference between message completion and message activation i.e., when a message is activated at time t and finishes at t', then t'- t is called "response time" of the message instance.

Definition 4.4: **Total response time** is the sum of response times of all the instances of a message. Note that response time for each message instance can be different.

Definition 4.5: **Average response time** is the ratio of total response time to the number of instances of a message during a hyperperiod.

V. Description of the CAN scheduling model

As stated before, access to the CAN bus is determined by IDs of competing messages. Then, now the question is how to assign ID's to CAN messages. The most popular and convenient of assigning is rate monotonic (RM) algorithm where lower periodic messages are assigned higher priority than the longer periodic messages [13], [15]. In RM, periods and deadlines are assumed to be equal. If the deadline of the message is less than the period, messages are assigned priority according to the deadline monotonic policy where priorities



CAN Device
Starve bit = 1
Message Queue based on Deadline
Next Round Arbitration: Starve bit =0
CAN Device
Starve bit = 1
Message Queue based on Deadline
Next Round Arbitration: Starve bit =0
Higher priority message Broadcast
BITWISE ARBITRATION CHANNEL
Winner in Dominant Mode: Starve bit = 1
Message Queue based on Deadline
CAN Device
Next Round Arbitration: Starve bit =0
Message Queue based on Deadline
CAN Device

Fig. 3. Starvation-avoid with Priority Queue Model

are based on the relative deadlines of the message. In both RM and DM, once message IDs are assigned by the processor, they don't change during consecutive activation and hence, they are known as fixed priority algorithms. Theoretically, any other meaningful static priority algorithms can be used for scheduling CAN messages. The advantage of fixed priority algorithms is it results in a predictable schedule and reliable schedulability analysis. On the other hand, fixed priority algorithms result in lower bus utilization [13]–[15]. To increase the bus utilization, dynamic priority algorithms such as Earliest Deadline First (EDF) have been proposed for scheduling CAN messages [16], [17]. In EDF scheduling, absolutes deadlines are used as priority and they are encoded in message ID's. This approach requires updating the message IDs periodically at every clock tick or arbitration round which causes more overhead on the processor at each ECU. In addition, absolute deadline values get larger and larger with time, therefore more bits are required eventually to encode these deadlines than available in CAN ID. Both fixed and dynamic priority assignment for CAN messages led to a large body of research into scheduling theory for CAN. This involved calculating the maximum queuing delay and worst-case response times for CAN messages to provide deadline guarantees and was used as the basis for commercial CAN schedulability tools [19], [20].

A. CAN Scheduling with FIFO and Priority Queues

Most scheduling studies of priority-based CAN messages were based on the implementation of FIFO queues (FQs) or priority queues (PQs) for ECUs connected to the CAN network [11], [18], [22]. In these studies, system is assumed to comprise a number of nodes (ECUs) connected to a single CAN bus. Message are classified as periodic or sporadic and nodes are classified according to the type of message queue used in their device driver. The FQ-nodes implement a FIFO message queue, where PQ-nodes implement a priority queue. FQ-nodes are assumed to be capable of ensuring that, at any given time when bus arbitration starts, the oldest message in the FIFO queue is entered into arbitration.

The main advantage of FQ is that it is simple to implement and use. Some examples of the CAN controllers that implements FQs are MicroChip, PIC32MX, Infineon XC161CS, Renesas R32C/160, and XILINX LogiCORE IP AXI controller [21], [22]. The main issue here is not with the bounded priority inversions. Suppose if the lower priority messages access the CAN bus for a longer duration, then there is a large likelihood that higher priority messages can miss their deadlines.

On the other hand, PQ-nodes are assumed to be capable of ensuring that, at any given time when the bus arbitration starts, the highest priority message queued at the node is entered into arbitration. Intuitively, the most natural choice for message scheduling in CAN controllers is the implementation of PQs. The BXCAN and BECAN for the ST7 and ST9 Microcontrollers from STMicroelectronics, which includes hardware support for both priority-queued and FIFO-queued message transmission [23]. The use of PQs may not solve the problem of the bounded priority inversions due to same reasoning stated for FQs. Compared to the FQs, PQs can reduce the number of priority inversions, thereby increasing the schedulablity of the message set while also minimizing the response times for higher priority messages.

Several WCRT schedulability analyzes have been proposed in the literature for providing deadlines guarantees using the FQ and PQ queues or a combination of both [12]. Subsequent work in these directions, relaxed the assumptions that deadlines and periods of the messages need not be equal i.e., deadlines can be shorter or longer than its corresponding transmission period. Many of these analyzes also include periodic and sporadic messages in their schedulability studies. Recently, Mubeen et al. [11], [12] started to analyze and schedule periodic, sporardic, and mixed messages in CAN where mixed messages are simultaneously periodic (time-driven) and sporadic (event-driven). Periodic, sporadic, and mixed messages can belong to either PQ's or FQ's and a detailed schedulability analysis for these possible scenarios can be found in [12].

VI. Starvation-avoidance Scheduling

A. Message ID with Starvation Bit

In our proposed system as shown in Figure 2, the 11-bit message IDs are broken into 10-bit static ID field for encoding priority and 1-bit dynamic ID field known as starvation bit field shown in Figure 2. With 10-bit static field, 1024 unique priority values can be assigned to message IDs. These priorities can be determined using any well-known optimal static or dynamic real-time scheduling algorithms such as RM or EDF. If the priority is static, designer has also the choice to use their own scheme of priority assignment using a well-established schedulability tests similar to the table-driven or cyclic schedulers in real-time systems. The starvation bit field is initialized with ("1") for all messages indicating they are not starving to begin with. At each arbitration round, all the CAN messages losing in the current CAN bit-wise arbitration switches its starvation bit field from ("1") to ("0") for the next round of arbitration. Thus each message in the proposed system is guaranteed to participate in the following round to transmit its message within their deadlines.

With the 1-bit dynamic ID field, any recessive back-off messages can retry to occupy the CAN bus not missing their own message deadline. The starvation bit field can be made more flexible i.e., larger than 1 bit depending upon various network applications. The messages using the starvation ID start from the lowest number of 1023 with bit "111111111" end with the highest number of 0 with bit "000000000".

B. Starvation Avoidance with Priority Queue Model

As seen in Figure 2, CAN messages have been created with 10 bits such that leftmost 1 bits are filled with 1 in order to provide dynamic priority, where message IDs are from 100000000 (=1024) to 1111111111 (=2047), and dynamic Starvation IDs are 1,0. Our proposed starvation-avoidance bit-wise model can be explained with a starvation bit. CAN messages which arrrive almost at the same time to enter the CAN bus, *Critical Section*, with their initial starvation bit = 1 and earliest deadline first. When the CAN bitwise arbitration is implemented on the bus, only the highest-priority message is broadcasted. The remaining lower-priority messages on the CAN, who could not be in *Critical Section*, go to the next round of transmission with their starvation bit = 0 when placed back to their corresponding queue with their newly updated deadlines, so that the priorities of remaining recessive messages are incremented as seen in Figure 3. The following is the algorithm.

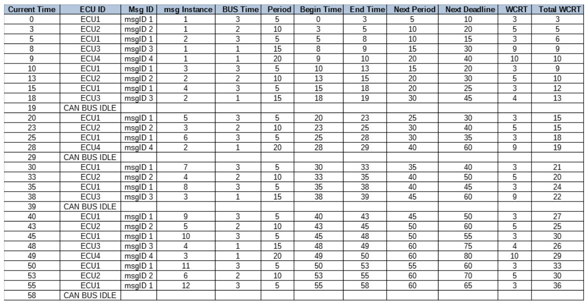
We demonstrate the working of proposal method with the example message set, consisting of four ECU's shown in the Table I. There are four messages with transmission time, T and period, P in this message set. Deadline, D and periods are assumed to be equal. Each of this message is allocated to one of the four ECUs denoted by their IDs. Message IDs correspond to its priority and they are allocated to the rate monotonic policy. The execution traces of non-preemptive RM and non-preemptive RM using a single starvation bit are shown in Figures 4 and 5. The execution traces clearly show that proposed starvation reduction model significantly reduces the total response times and average response times for all the four messages. These reductions are clearly due to the increase in the priorities of recessive messages at each arbitration round unlike the non-preemptive RM.

TABLE I
Message set

An example message set with a CAN bus utilization of 91.67%			
ECU ID	**Msg ID**	T (*ms*)	$P = D$ (*ms*)
ECU1	MsgID 1	2	5
ECU2	MsgID 2	2	10
ECU3	MsgID 3	1	15
ECU4	MsgID 4	1	20

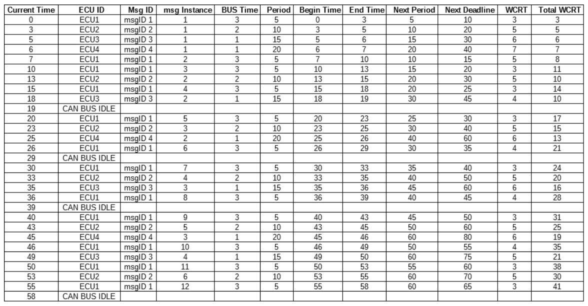
VII. Experimental Setup

We compare our proposed method with the non-preemptive scheduling of CAN messages which does not use starvation



Current Time	ECU ID	Msg ID	msg Instance	BUS Time	Period	Begin Time	End Time	Next Period	Next Deadline	WCRT	Total WCRT
0	ECU1	msgID 1	1	3	5	0	3	5	10	3	3
3	ECU2	msgID 2	1	2	10	3	5	10	20	5	5
5	ECU1	msgID 1	2	3	5	5	8	10	15	3	6
8	ECU3	msgID 3	1	1	15	8	9	15	30	9	9
9	ECU4	msgID 4	1	1	20	9	10	20	40	10	10
10	ECU1	msgID 1	3	3	5	10	13	15	20	3	9
13	ECU2	msgID 2	2	2	10	13	15	20	30	5	10
15	ECU1	msgID 1	4	3	5	15	18	20	25	3	12
18	ECU3	msgID 3	2	1	15	18	19	30	45	4	13
19	CAN BUS IDLE										
20	ECU1	msgID 1	5	3	5	20	23	25	30	3	15
23	ECU2	msgID 2	3	2	10	23	25	30	40	5	15
25	ECU1	msgID 1	6	3	5	25	28	30	35	3	18
28	ECU4	msgID 4	2	1	20	28	29	40	60	9	19
29	CAN BUS IDLE										
30	ECU1	msgID 1	7	3	5	30	33	35	40	3	21
33	ECU2	msgID 2	4	2	10	33	35	40	50	5	20
35	ECU1	msgID 1	8	3	5	35	38	40	45	3	24
38	ECU3	msgID 3	3	1	15	38	39	45	60	9	22
39	CAN BUS IDLE										
40	ECU1	msgID 1	9	3	5	40	43	45	50	3	27
43	ECU2	msgID 2	5	2	10	43	45	50	60	5	25
45	ECU1	msgID 1	10	3	5	45	48	50	55	3	30
48	ECU3	msgID 3	4	1	15	48	49	60	75	4	26
49	ECU4	msgID 4	3	1	20	49	50	60	80	10	29
50	ECU1	msgID 1	11	3	5	50	53	55	60	3	33
53	ECU2	msgID 2	6	2	10	53	55	60	70	5	30
55	ECU1	msgID 1	12	3	5	55	58	60	65	3	36
58	CAN BUS IDLE										

Fig. 4. Execution Trace - Non-preemptive RM CAN BUS scheduling



Current Time	ECU ID	Msg ID	msg Instance	BUS Time	Period	Begin Time	End Time	Next Period	Next Deadline	WCRT	Total WCRT
0	ECU1	msgID 1	1	3	5	0	3	5	10	3	3
3	ECU2	msgID 2	1	2	10	3	5	10	20	5	5
5	ECU3	msgID 3	1	1	15	5	6	15	30	6	6
6	ECU4	msgID 4	1	1	20	6	7	20	40	7	7
7	ECU1	msgID 1	2	3	5	7	10	10	15	5	8
10	ECU1	msgID 1	3	3	5	10	13	15	20	3	11
13	ECU2	msgID 2	2	2	10	13	15	20	30	5	10
15	ECU1	msgID 1	4	3	5	15	18	20	25	3	14
18	ECU3	msgID 3	2	1	15	18	19	30	45	4	10
19	CAN BUS IDLE										
20	ECU1	msgID 1	5	3	5	20	23	25	30	3	17
23	ECU2	msgID 2	3	2	10	23	25	30	40	5	15
25	ECU4	msgID 4	2	1	20	25	26	40	60	6	13
26	ECU1	msgID 1	6	3	5	26	29	30	35	4	21
29	CAN BUS IDLE										
30	ECU1	msgID 1	7	3	5	30	33	35	40	3	24
33	ECU2	msgID 2	4	2	10	33	35	40	50	5	20
35	ECU3	msgID 3	3	1	15	35	36	45	60	6	16
36	ECU1	msgID 1	8	3	5	36	39	40	45	4	28
39	CAN BUS IDLE										
40	ECU1	msgID 1	9	3	5	40	43	45	50	3	31
43	ECU2	msgID 2	5	2	10	43	45	50	60	5	25
45	ECU4	msgID 4	3	1	20	45	46	60	80	6	19
46	ECU1	msgID 1	10	3	5	46	49	50	55	4	35
49	ECU3	msgID 3	4	1	15	49	50	60	75	5	21
50	ECU1	msgID 1	11	3	5	50	53	55	60	3	38
53	ECU2	msgID 2	6	2	10	53	55	60	70	5	30
55	ECU1	msgID 1	12	3	5	55	58	60	65	3	41
58	CAN BUS IDLE										

Fig. 5. Execution Trace - Non-preemptive RM CAN BUS scheduling with a single starvation bit

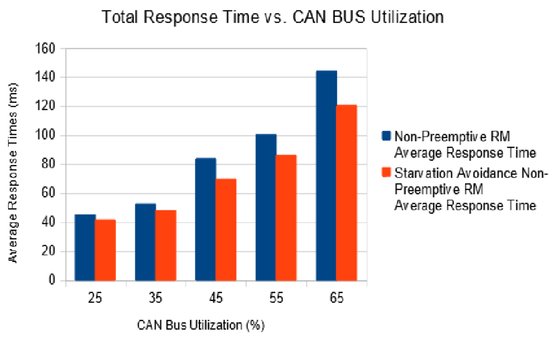


Fig. 6. Average response time (ms) vs. CAN Bus Utilization

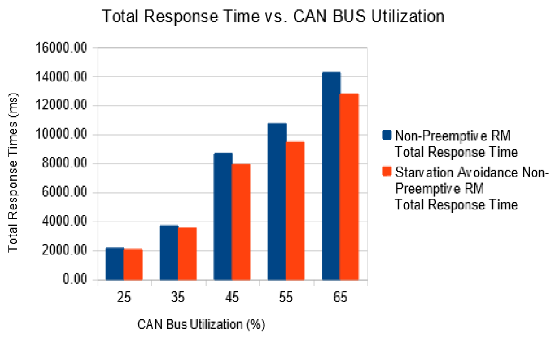


Fig. 7. Total response time (ms) vs. CAN Bus Utilization

bit field. Our system model consists of 5 ECUs connected to a single CAN network. We performed simulation using 100 randomly generated message for all our experiments and each point in graph represents the average for 100 message sets. Each message from the message set Γ was randomly allocated to one of the five ECU's. We make the following assumptions which are listed below:

- Each message set Γ consists of periodic messages which were generated following the condition such that $\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$ where n represents the number of messages.
- The periods for messages from ECU1, ECU2, ECU3, ECU4, and ECU5 were chosen uniformly from the following sets in ms:
  - ECU1 periods = $[10, 20, 30, 40, 50]$
  - ECU2 periods = $[40, 60, 80, 100, 120]$
  - ECU3 periods = $[120, 140, 160, 180, 200]$
  - ECU4 periods = $[140, 160, 180, 200, 220]$
  - ECU5 periods = $[160, 180, 200, 220, 240]$
- The worst case transmission times were chosen as $3ms$, $3ms$, $2ms$, $2ms$, and $1ms$ for ECU1, ECU2, ECU3, ECU4, and ECU5 respectively.
- The relative deadlines of messages were assumed equal to their periods.
- Each message was assigned a unique message ID by the host processor in each ECU using the rate monotonic algorithm i.e., messages with smaller periods have higher priority than messages with longer periods. For any pair of messages m_x and m_y, if $x < y$, then m_x has higher priority than m_y.
- Each ECU maintain its own priority queue. Messages in the priority queue are sorted according to the increasing order of message ID i.e., the message with lowest ID is at the front of the priority queue.
- Once messages are assigned priority by the processor in each ECU and their priorities do not change for subsequent activations.
- All messages instances generated by each ECU are periodic. The absolute transmission periods for message, m are: $[0, T_k], [T_k, 2T_k], [2T_k, 3T_k], \ldots$. The end of the periods $T_k, 2T_k, \ldots$, are defined as the absolute deadlines for m in the respective periods. Since the transmission period of each of message is equal to its deadline i.e., the next message instance, m_{k+1} is ready at the end of the current transmission period, T_k.
- Messages are non-preemptive i.e., once the message from an ECU access the CAN bus then it cannot be preempted even if a higher priority message arrives in same PQ or PQs of other ECUs.

VIII. Conclusion and Future Work

Our simulation study clearly suggests that proposed model reduces starvation by improving the total and average response

Algorithm 1 Starvation-avoidance CAN Scheduling

```
byte flagCAN = 1, busCAN = 1; //semaphores
byte cnt = 0, int n = total #_of_ CAN messages;
int m1 = total #_of_ECUs, m2 = size of ECU queues;
loop forever (while CAN is ON)
    $ECU_{i,k} \leftarrow message_j$, //messages arrive to ECUs;
            where i = 1..m1, k = 1.. m2, j = 1..n;
    $message_{i,HighestPriority}$ are polled from ECUs,
            where i = 1 .. m1;
    if (CAN BUS Arbitration) {
        cnt++;
        if (cnt $<$ m) $\rightarrow$ signal(flagCAN);
        signal(busCAN);
        wait(flagCAN); //simulated wait
        wait(busCAN);
        cnt++;
        if (cnt $<$ m) $\rightarrow$ signal(flagCAN)
        signal(busCAN);
            Critical Section (winner is scheduled)
        wait(busCAN); //simulated signal
        cnt−−;
        if (cnt $<=$ 0)
            signal(flagCAN);
        else
            wait(flagCAN);
        signal(busCAN);
        $messages_{loser}$ are updated for next round
                with setting starvation bit to 0;
    $\}_{CANArbitration}$
    else
        $message_{max}\ of\ message_{i,HighestPriority}\ is\ scheduled$;
    signal(busCAN);
```

times for peridoic jobs in the CAN. As shown in Figures 6 and 7, our proposed model significantly reduces starvation by reducing the total response times and average response times of messages and also, these results are consistent for different CAN bus utilization. This is due to the increase in the priorities of recessive messages at each arbitration round unlike the non-preemptive RM. We would like to extend our study to analyze whether we can use this starvation reduction model to improve the response times for mixed messages in CAN.

References

[1] Murtaza A. and Khan Z., "Starvation Free Controller Area Network using Master Node,"Proceedings of IEEE 2nd International Conference on Electrical Engineering, Lahore, Pakistan, March, 25−26, 2008.

[2] MSC8122: Avoiding Arbitration Deadlock During Instruction Fetch, FreeScale Semiconductor, INC., 2008.

[3] Lee K, et al, "Starvation Prevention Scheme for a Fixed Priority Pci−Express Arbiter with Grant Counters using Arbitration Pools," US Patent Application Publication, Jan. 14, 2009.

[4] Lin, Cheng-Min, "Analysis and Modeling of a Priority Inversion Scheme for Starvation Free Controller Area Networks", IEICE Transactions on Information and Systems, VOL. 93−D(6), JUNE 2010.

[5] Guilherme Zago and Edison deFreitas, "A Quantitative Performance Study on CAN and CAN FD Vehicular Networks", IEEE Transactions on Industrial Electronics Volume: 65, Issue: 5, May 2018.

[6] Park P, et al, "Performance Evaluation of a Method to Improve Fairness in In-vehicle Non-Destructive Arbitration Using ID Rotation", KSII Transactions on Internet and Information Systems, Vol. 11, No. 10, Oct. 2017.

[7] Haklin Kimm and Hanke Kimm, "Modeling and Verification of Starvation-Free Arbitration Technique for Controller Area Network Usig SPIN Promela", Springer Series in Advanceds in Intelligent Systems and Computing, Vol. 935, 2019.

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

[9] K.W. Tindell, H. Hansson, and A.J. Wellings, "Analysing real-time communications: Controller Area Network (CAN)". In Proceedings 15th Real-Time Systems Symposium (RTSS'94), pp. 259-263. IEEE Computer Society Press, December 1994.

[10] Robert I. Davis, Steffen Kollmann, Victor Pollex, Frank Slomka, "Controller Area Network (CAN) Schedulability Analysis with FIFO queues," 2011 23rd Euromicro Conference on Real-Time Systems, Porto, Portugal.

[11] Saad Mubeen, Jukka MÄki-Turja and Mikael SjÖdin, "Extending Worst Case Response-Time Analysis for Mixed Messages in Controller Area Network With Priority and FIFO Queues," IEEE Access, Volume 2:365-380,2014.

[12] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Support for end-to-end response-time and delay analysis in the industrial tool suite: Issues, experiences and a case study," Comput. Sci. Inform. Syst., vol. 10, no. 1, pp. 453–482, 2013.

[13] Aravind, Alex A and Chelladurai, Jeyaprakash, "Activation-Adjusted Scheduling Algorithms for Real-Times Systems", Advances in Systems, Computing Sciences and Software Engineering, pp. 425-432, 2006.

[14] Park, Moonju and Chae, Jinseok, "Utilization Bound of Non-preemptive Fixed Priority Schedulers", The Institute of Electronics, Information and Communication Engineers Transactions on Information and Systems, vol. 92(10), pp. 2152-2155, 2009.

[15] Andersson, Bjorn and Tovar, Eduardo, "The utilization bound of non-preemptive rate-monotonic scheduling in Controller Area Networks is 25%", IEEE International Symposium on Industrial Embedded Systems, pp. 11-18, 2009.

[16] Zuberi, Khawar M and Shin, Kang G, "Scheduling messages on controller area network for real-time CIM applications", IEEE Transactions on Robotics and Automation, vo. 13(2), pp. 310-316, 1997.

[17] Pedreiras, Paulo and Almeida, Luis, "EDF message scheduling on controller area network", Computing & Control Engineering Journal, vol. 13 (4), pp. 163-170, 2002.

[18] Alkan, Burak, "Controller area network response time analysis and scheduling for advanced topics: offsets, FIFO queues and gateways", MS Thesis, Middle East Technical University, 2015.

[19] *Volcano Network Architect* (Online). Available: https://www.mentor.com/products/electrical-design-software/networks Last Accessed- 25th February 2020.

[20] *Rubus-ICE* (Online). Available: https://www.arcticus-systems.com/ Last Accessed - 25th February 2020.

[21] Khan, Dawood A, Bril, Reinder J, and Navet, Nicolas, "Integrating hardware limitations in CAN schedulability analysis", IEEE International Workshop on Factory Communication Systems Proceedings, pp. 207-210, 2010.

[22] Davis, Robert I, Kollmann, Steffen, Pollex, Victor, and Slomka, Frank, "Schedulability analysis for Controller Area Network (CAN) with FIFO queues priority queues and gateways", Real-Time Systems Journal, 49(1), pp. 73-116, 2013.

[23] STMicroelectronics, "AN1077 Application note. Overview of enhanced CAN controllers for the ST7 and ST9 MCUS", 2001 (available from www.ST.com).

[24] Grenier, Mathieu, Havet, Lionel and Navet, Nicolas, "Scheduling messages with offsets on Controller Area Network-a major performance boost", The automotive embedded systems handbook, 2008.