Implementing Off-line Message Scheduling on Controller Area Network (CAN)

Radu Dobrin and Gerhard Fohler
Department of Computer Engineering
Mälardalen University, Sweden
{radu.dobrin,gerhard.fohler}@mdh.se

***Abstract* - Controller Area Network (CAN) is widely used in a number of industrial applications. The message scheduling on CAN is based on identifiers (ID) assigned to the messages according to their priorities. Fixed priority scheduling protocols (FPS) have a number of advantages, some of them being the efficient exploitation of the channel bandwidth, small overhead and the simple implementation. On the other hand, a number of industrial applications demand temporal properties that are typically achieved by using off-line scheduling. In addition, complex constraints can be solved off-line, such as distribution, end-to-end deadlines, precedence, jitter, or instance separation, but this scheduling strategy is not suitable for CAN.**

In this paper we present a method that shows how off-line scheduled messages can be scheduled on CAN. The paper assumes that a schedule, for a set of tasks transmitting messages on CAN, has been constructed off-line. It presents a method that analyzes the off-line schedule and derives a set of periodic messages with fixed priorities, which can be scheduled on CAN. Based on the information provided by the off-line schedule, the method derives inequality relations between the priorities of the messages under FPS. In case the priority relations of the messages are not solvable, we split some messages into a number of artifacts, to obtain a new set of messages with consistent priorities. We use integer linear programming to minimize the final number messages.

1. Introduction

Controller Area Network (CAN), has gained wider acceptance as a standard in a large number of industrial applications. The priority based message scheduling used in CAN has a number of advantages, some of the most important being the efficient bandwidth utilization, flexibility, simple implementation and small overhead. Early results on message scheduling on CAN have been presented in [13] and [12], in which the authors focused on fixed priority scheduling based on work presented in [9] and [8]. Later on, Zuberi [14] showed that static priority scheduling is not always the most suitable strategy. Earliest Deadline (EDF) can prove significantly better then fixed priority scheduling [11].

Off-line scheduling for time triggered systems, on the other hand, provides determinism [6], [7], and, additionally, complex constraints can be solved off-line, but this scheduling strategy is not suitable for CAN.

In this paper we present a method that transforms off-line scheduled transmission schemes into sets of messages that can be scheduled on CAN. It assumes a schedule has been constructed for a set of off-line scheduled messages to meet their complex constraints. Our method takes the off-line schedule with derived time intervals in which messages must be transmitted, from now on referred to as *Target Windows*, and assigns FPS attributes, (i.e., priorities) to the messages. It then, provides information about periods and offsets the messages have to be sent with by the sending nodes, such that the message transmission at runtime matches the off-line schedule. It does so by deriving priority inequalities, which are then resolved by integer linear programming.

FPS cannot reconstruct all schedules with periodic messages with the same priorities for all instances (invocations) directly. The constraints expressed via the off-line schedule may require different message sequences for invocations of the same message, as, e.g., by earliest deadline first, leading to inconsistent priority assignment. This phenomenon can be expressed as a cycle of inequalities. Our algorithm detects such situations, and circumvents the problem by splitting a message into its invocations. Then, the algorithm assigns different priorities to the newly generated "artifact" messages, the former invocations.

Key issues in resolving the priority conflicts are the number of artifact messages created. Depending on where a priority conflict circle is "broken", the number may vary, depending on the periods of the split messages. Our algorithm minimizes the number of artifact messages by solving the priority inequalities with integer linear programming (ILP).

Priority assignment for FPS tasks has, for example, been studied in [1], [10] and [5]. [2] study the derivation of task attributes to meet a overall constraints, e.g., demanded by control performance. Instead of specific requirements, our algorithm takes an entire off-line schedule and all message requirements to determine message attributes. A method to transform off-line schedules into earliest deadline first tasks has been presented in [4]. A related paper [3] deals with priority assignment for off-line CPU scheduled tasks. It uses a constructive, heuristic approach, potentially creating large numbers of artifacts, while the approach presented here presents a general algorithm applied to message scheduling on CAN using ILP for optimum solutions.

The rest of the paper is organized as follows. In section 2 we give a brief overview of message scheduling on CAN. The method we are proposing is presented in section 3 and, then, illustrated with an example in section 4. Section 5 concludes the paper.

2. Controller Area Network (CAN) and message scheduling

CAN consists of the physical and data link layers. Each CAN frame consist of seven fields. In this paper we focus on the identifier field (ID). The identifier field may have two lengths: 11 bits, which is the standard format, and 29 bits, the extended format, and it controls message addressing and bus arbitration . In this paper we focus only on the former since the nodes can set message filters in order to receive only the identifiers they are interested in.

The nodes are connected via a wired OR (or wired AND) CAN bus. The time axis is divided in slots which must be larger or equal to the time it takes the signal to propagate back and forth the bus, $t = \frac{2L}{V}$, where L is the bus length and V is the propagation speed of the signal. When a node has to send a message, it calculates the message ID which may be based on the priority of the message. The message ID must be unique in order to prevent eventually ties. The message is then sent to the bus interface chips, which, further on, write the message ID on the bus, bit by bit, whenever the bus is idle at the beginning of a time slot. After writing a bit on the bus, the chip waits for the signal to propagate along the bus and, then, reads the bus. If the bit read is different from the bit sent, then there is another message on the bus with a higher priority, and the sending node aborts the transmission. Otherwise the node gets the right to send the message without being pre-empted.

In our paper, we assume a schedule has been constructed for a set of off-line messages. The proposed method transforms the off-line scheduled messages into as set of messages suitable for priority-based CAN message scheduling. Since we do not want to assign any other attributes then priorities to the message ID's (e.g., periods and offsets), due to the restrictions enforced by the ID format, we rather provide information about attributes that have to be assigned by the programmer to the sending nodes and messages (periods, offsets and priorities) in order to ensure the run-time transmission of the messages according to the specifications expressed in the off-line schedule. Furthermore, we assume that the nodes are clock synchronized and the off-line schedule has been constructed by taking into account the increased bandwidth consumption due to the exchange of messages required by the time synchronization method.

3. Attribute assignment algorithm

3.1. Overview

Figure 1 gives an overview of the algorithm.

1) and 2) Initially, the off-line schedule table for a set of messages with constraints, is given.

3) Target windows for each invocation of each message are derived from the original message constraints and the off-line schedule.

4) Sequences are now straightforward to derive from the target windows and the transmission order expressed in the off-line schedule.

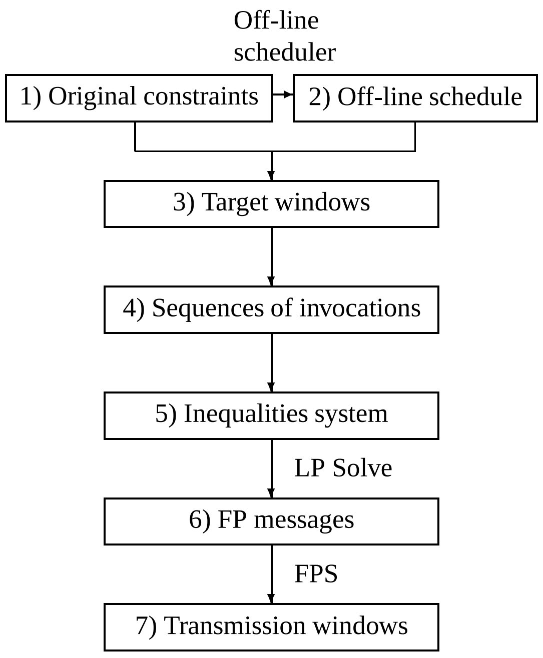


Figure 1: Algorithm overview.

5) The analysis of each sequence provides a set of inequalities between priorities of invocations of different messages.

6) We use integer linear programming to solve the system of inequalities and the result is the final set of messages with fixed priorities.

Off-line schedule: The input to our method is the off-line schedule expressing the constraints specified for the messages to be sent on CAN. The schedule is usually created up to the least common multiple, LCM, of all message periods. We have $LCM/T(M_i)$ invocations of each message M_i with period $T(M_i)$ in the off-line schedule.

The off-line scheduler resolves constraints such as distribution, end-to-end deadlines, precedence, etc, and creates scheduling tables for each node in the system, listing start- and finishing-times of all message invocations. These scheduling tables are more fixed than required by the original constraints, so we can replace the exact sending- and receiving-times of messages with target windows, taking the original constraints into account.

Target windows $(TW(M_i^j))$ of each invocation M_i^j of each message M_i,are derived from the off-line schedule and the original constraints transformed into earliest start times and deadlines.

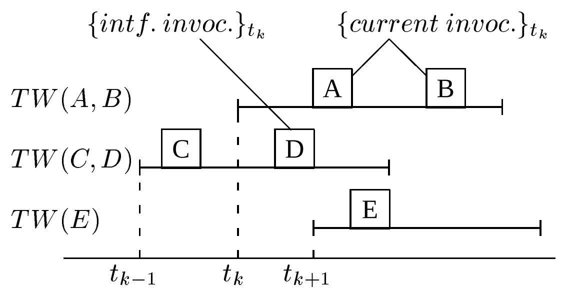
$$TW(M_i^j) = [t_m, t_n]$$

where

$$t_m = begin(TW(M_i^j)) \; and \; t_n = end(TW(M_i^j))$$

The *earliest transmission start time*, $est(M_i^j)$, of an invocation M_i^j of a message M_i, is provided by the message constraints expressed in the off-line schedule. The *scheduled* *receiving time*, $srt(M_i^j)$, of an invocation M_i^j of a message M_i, is the time when M_i^j is received by the receiving node according to the off-line schedule. The *scheduled transmission start time*, $start(T_i^j)$, of an invocation M_i^j of a message M_i, is the time when M_i^j is sent on the bus, according to the off-line schedule.

A **sequence** $S(t_k)$ consists of invocations of messages M_i^j ordered by increasing scheduled transmission start times according to the off-line schedule. A sequence may contain invocations M_i^j such that $begin(TW(M_i^j)) = est(M_i^j) = t_k$, current invocations of $TW(M_i^j)$, and invocations M_p^q from overlapping target windows such that $est(M_p^q) < t_k$ and $start(M_p^q) > t_k$, interfering invocations of $TW(M_i^j)$. Additionally: $first(S(t_k)) = S(t_k)^1$ = first message invocation in the sequence $S(t_k)$, and $last(S(t_k)) = S(t_k)^N$ = last message invocation in $S(t_k)$. The derivation of a sequence corresponding to a time t_k is illustrated in figure 2. Note that, in figure 2, message 'E' is not included in the sequence corresponding to the time t_k, since it's earliest transmission start time is greater then t_k. 'E' will be instead a *current invocation* in the sequence corresponding to the time t_{k+1}.



$$S(t_k) =< S(t_k)^1, S(t_k)^2, S(t_k)^3 >=< D, A, B >$$

Figure 2: Sequence of messages.

We refer to an *transmission window*, $W_{trans}(M_i^j)$, of an invocation M_i^j of a message M_i, as the time interval in which M_i^j will be sent and received *at runtime*. We want to find fixed priorities, offsets, and deadlines such that the transmission window of each message invocation M_i^j, $W_{trans}(M_i^j)$, will be contained within the respective target window $TW(M_i^j)$, and transmission order specified off-line, kept.

3.2. Priority inequalities

Our algorithm derives relations (inequalities of priorities) among the invocations of the messages by traversing the off-line schedule represented by the series of target windows in increasing order of time. It determines priority inequalities between invocations according to the sequences $S(t_k)$ associated with target windows, such that:

$$P(S(t_k)^1) > P(S(t_k)^2) > \ldots > P(S(t_k)^N)$$

where

$$S(t_k)^1 = first(S(t_k))\ and\ S(t_k)^N = last(S(t_k))$$

Note that the inequalities have to take into account relations between priorities of invocations of the current target window and possibly interfering target windows.

3.3. Attribute assignment - conflicts

Our goal is to provide messages with fixed priorities periodically sent on the bus. It may happen, however, that we have to assign different priorities or/and offsets to different invocations of the same message in order to reenact the off-line schedule at run time. These cases cannot be expressed directly with fixed priorities and fixed offsets and are the sources for *offset assignment conflicts* or *priority assignment conflicts*. In both cases, we split the conflicting message into artifacts, such that, further on, each artifact will be considered an independent message, invoked only once during LCM. Thus we create a number of artifact messages equal to the number of invocations during LCM of the message to be split minus one (since the original message will be replaced by a number of messages equal to the number of its invocations).

By *offset assignment conflict* we mean that different invocations of the same message may have to be invoked at different points in time, relative to the sending task period, in order to ensure the run-time transmission of each one of them in the derived target window.

```
for 1 $\le i \le nr\_of\_off-line\_sched\_messages$
    for 1 $\le j \le n,\ where\ n = LCM/T(M_i)$
        if $begin(TW(M_i^j)) - (j-1) * T(M_i) \neq$
            $\neq begin(TW(M_i^{j+1})) - j * T(M_i)$,
            *(where $T(M_i)$ is the period of the message $M_i$)*
        then split $M_i$ into $M_{i,1}, M_{i,2}, \ldots, M_{i,n}$
```

By splitting M_i, we remove it from the original set of messages, *orig_messages*, and we insert $M_{i,1}$, $M_{i,2}$, ..., $M_{i,n}$ into *orig_messages*.

Priority assignment conflicts are detected after the derivation of the sequences, and occurs in the cases when two different invocations of the same task may have to be sent with different priorities in order to ensure the run-time transmission of each one of them in the derived target window, and in the right position in the sequence the message belongs to. In this case, since a priority assignment involves more than one message, there is typically a choice of which message to split.

In our method, we split messages that causes offset assignment conflicts into artifacts *before* deriving the sequences of invocations. By that, we reduce the probability of priority assignment conflict eventually caused by the same messages since the new created messages will be invoked only once during LCM.

3.4. Minimizing the final number of messages

In order to minimize the number of artifact messages, we create an integer linear programming problem from the derived system of priority inequalities to first identify which messages to split, if any, and to derive priorities for the resulting fixed priority messages. We aim for the minimum amount of artifact messages, and implicitly priorities, due to the limited amount of priorities available when scheduling messages on CAN.

The inequalities obtained from the execution order within the sequences, may form a circular chain of priority relations between messages invocations, e.g.,

$$P(M_i^j) > P(M_m^n) > \ldots > P(M_i^{j+k}) > \ldots > P(M_m^{n+q})$$

We use a higher value to represent a higher priority.

In this case we cannot assign the same priority to both invocations j and $(j+k)$ of M_i, nor to invocations n and $(n+q)$ of M_m. We have to break the chain by splitting either M_i or M_m into artifacts and considering each one of them as individual messages, which will result in a larger number of messages compared to the number of original off-line scheduled messages. We formulate a goal function for an integer linear programming solver to identify the minimum amount of messages with fixed priorities.

$$G = \#final\_msgs = \#orig\_msgs + \sum_{i=1}^{N}(|M_i| - 1) * b_i$$

where $\#final\_msgs$ is the number of final messages, $\#orig\_msgs$ is the number of original messages, $|M_i|$ = number of invocations of M_i in LCM and b_i is a boolean variable associated to each message M_i, $b_i \in \{0,1\}$. b_i = 1 means that M_i needs to be split into $|M_i|$ messages. Additionally, the solver provides priority values for the messages (split or non-split).

At this point we have a set of messages with fixed priorities, *final_msgs*, produced by the LP-solver. Finally, we assign periods and offsets to each message (i.e., provide information about when the messages are to be sent by the controller interface) provided by the LP-solver in order to ensure the run time transmission of the messages within their respective target windows, as following:

$$\begin{aligned} for \quad & 1 \le i \le \#(final\_msgs) \\ & T(M_i) = \frac{LCM}{nr\_of\_invocations(M_i)} \\ & offset(M_i) = begin(TW(M_i^1)) \end{aligned}$$

4. Example

We illustrate the method with an example. Assume that we have the set of messages, sent from two nodes, shown in figure 3. Additionally we assume that we have a precedence constraint between the $(4m+1)^{th}$ invocation of A and the $(2m+1)^{th}$ invocation of B,

Message	Node	message size	period (T)
A	1	1	5
B	2	3	10
C	1	4	20

Figure 3: Original set of messages

$$A^{4m+1} \rightarrow B^{2m+1}$$

where $m = 0, 1, 2, \ldots$, and a precedence constraint between the $(2n+2)^{th}$ invocation of B and the $(4n+3)^{th}$ invocation of A,

$$B^{2(n+1)} \rightarrow A^{4n+3}$$

where $n = 0, 1, 2, \ldots$.

The off-line schedule for the messages and the derived target windows are illustrated in figure 4.

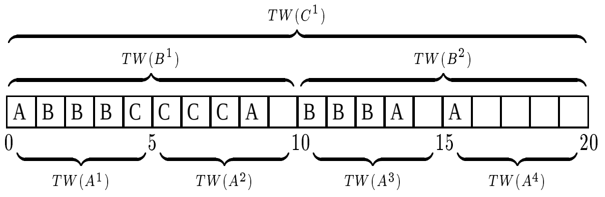


Figure 4: Off-line Scheduled Messages and Target Windows

The derivation of the inequalities, performed as described in section 3.2, is illustrated in the figure 5.

t_k	Message invocations	$S(t_k)$	inequalities
0	A^1, B^1, C^1	A^1, B^1, C^1	$P(A^1) > P(B^1)$ $P(B^1) > P(C^1)$
5	A^2	A^2	
10	A^3, B^2	B^2, A^3	$P(B^2) > P(A^3)$
15	A^4	A^4	

Figure 5: Inequalities

At time t_k=10, we have the inequality $P(B^2)>P(A^3)$added to the relations obtained at t_1=0 and t_2=5: $P(A^1)>P(B^1)$, and $P(A^2)>P(C^2)$. That gives a circular chain of priorities that must be solved: $P(A^1)>P(B^1)$,..., $P(B^2)>P(A^3)$. In this case, we can either choose to split message A, or message B.

Splitting B will create two artifact messages, while splitting A will result in four. The integer linear programming solver with the goal function described in section 2.3 provides the solution:

- $b_A = b_C = 0$
- $b_B = 1$, meaning message B is to be split,
- $p_A = 2$
- $p_{B^1} = 3$
- $p_{B^2} = 1$

- $p_C = 4$
- $\#final\_msg = \#orig\_msg + \sum_{i=1}^{N}(|T_i| - 1) * b_i = 4$

After assigning offsets and periods to the artifact messages (i.e., B1 and B2), as described in section 3.3, the final set of messages is shown in figure 6. The lowest value represents the highest priority.

msg	node	msg size	T	offset	dl	prio
A	1	1	5	0	5	2
B1	2	3	20	0	10	3
B2	2	3	20	10	20	1
C	1	4	20	0	20	4

Figure 6: FP messages

5. Conclusions and future work

In this paper we have presented a method that shows how off-line scheduled messages can be scheduled on CAN. We use off-line schedules and target windows to express complex constraints and predictability for selected messages. We, then, derive attributes for the off-line scheduled messages, such that the messages will be transmitted within the specified target windows while fulfilling the original constraints, when scheduled on CAN.

Our method analyzes the off-line transmission scheme and the target windows and derives priority relations between the invocations of the messages, expressed in a set of inequalities. In certain cases, the method splits messages into instances, creating artifact messages with fixed priorities, as not all off-line schedules can be expressed directly with FPS. We use standard integer linear programing to solve the priority inequalities and minimize the number of artifact messages created. Finally, offsets and periods can be assigned in the implementation, to the set of sending tasks provided by ILP in order to ensure the run-time transmission of the messages within the derived target windows.

In same cases, we may perform additional splits, due to violation of the periodicity in the off-line schedule, which gives different offsets at which different instances of the same message have to be sent. The number of artifact messages caused by offset assignment conflicts, could be decreased by reducing target windows, if the resulting loss in flexibility is acceptable. The priority inversion phenomenon, due to the non-preemption of message transmission, can be solved by modifying the start of the target windows of the messages with precedence relations considering the precision achieved in the global time synchronization.

Our method does not introduce artifacts or reduce flexibility unless required by constraints: the fixed priority messages provided by our method, with input consisting of a set of messages with fixed priorities, scheduled off-line according to FPS, will be transmitted within the derived target windows and in the off-line specified transmission order.

To this point, we have concentrated on reconstructing the off-line scheduled messages. Using the flexibility of the ILP solver, we can add objectives by inclusion in the goal function. Future work will address the issue of message relations at run-time as well.

6. Acknowledgements

The authors wish to express their gratitude to the reviewers for their useful comments on the paper.

References

[1] N. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical report, Departament of Computer Science, University of York, 1991.

[2] J. L. D. Seto and L. Sha. Task period selection and schedulability in real-time systems. In *Proceedings of Real-Time Systems Symposium*, pages 188–198, 1998.

[3] R. Dobrin, Y. Ozdemir, and G. Fohler. Task attribute assignment of fixed priority scheduled tasks to reenact off-line schedules. In *Conference on Real-Time Computing Systems and Applications, Korea*, December 2000.

[4] G. Fohler. *Flexibility in Statically Scheduled Hard Real-Time Systems*. PhD thesis, Technische Universität Wien, Austria, Apr. 1994.

[5] R. Gerber, S. Hong, and M. Saksena. Guaranteeing real-time requirements with resource-based calibration of periodic processes. *IEEE Transactions on Software Engineering*, 21(7), July 1995.

[6] H. Kopetz. Why time-triggered architectures will succeed in large hard real-time systems. In *Proceedings of the Fifth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*, pages 2–9, 1995.

[7] H. Kopetz and G. Grunsteidl. TTP - a protocol for fault-tolerant real-time systems. *Computer*, 27(1):14–23, 1994.

[8] J.-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2(4):237–250, Dec. 1982.

[9] C. L. Liu and J. W. Layland. Scheduling algorithms for multi-programming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, Jan. 1973.

[10] K. N. Aundsley and A. Burns. The end of the line for static cyclic scheduling? In *Proceedings of the Fifth Euromicro Workshop on Real-Time Systems*, pages 36–41, 1993.

[11] M. D. Natale. Scheduling the CAN bus with earliest deadline techniques. In *Proceedings of the 21st IEEE Real-Time Systems Symposium*, pages 259–268, Dec 2000.

[12] K. Tindell, A. Burns, and A. Wellings. Calculating controller area network (CAN) message response times. *Contr. Eng. Practice*, 3(8):1163–1169, 1995.

[13] K. Tindell, H. Hansson, and A. Wellings. Analizing real-time communications: Controller area network (CAN). In *Proceedings of Real-Time Systems Symposium*, pages 259–263, Dec. 1994.

[14] K. Zuberi and K. Shin. Scheduling messages on controller area network for real-time CIM applications. *IEEE Transactions on Robotics and Automation*, 13(2):310–314, Apr. 1997.