Scheduling Messages on Controller Area Network for Real-Time CIM Applications

Khawar M. Zuberi and Kang G. Shin

Abstract— **Scheduling messages on the Controller Area Network (CAN) corresponds to assigning identifiers (IDs) to messages according to their priorities. In this paper we present the mixed traffic scheduler (MTS), which provides higher schedulability than fixed-priority schemes like deadline-monotonic (DM) while incurring less overhead than dynamic earliest-deadline (ED) scheduling. Through simulations, we compare the performance of MTS with that of DM and ED* (an imaginary scheduler which works like ED, except it incurs less overhead). Our simulations show that MTS performs much better than DM and at the same level as ED*, except under high loads and tight deadlines, when ED* is superior.**

I. Introduction

Local area networks (LANs) are becoming increasingly popular in industrial automation and other real-time control applications [1], [2]. LANs allow field devices like sensors, actuators, and controllers to be interconnected at low cost — using less wiring and requiring less maintenance than point-to-point interconnections [2]. Several architectures have been proposed for such LANs, including Controller Area Network (CAN) [3], SP-50 FieldBus [4], MAP [5], TTP [6], etc. Of these networks, CAN has gained widespread acceptance in the industry [7] — first in the automotive industry and then for industrial automation and computer-integrated manufacturing (CIM) as well. CAN is popular because of its low cost (a CAN interface chip costs about $5) and its useful features like reliability in noisy environments and priority-based bus arbitration.

Control networks must carry both periodic and sporadic real-time messages, as well as non-real-time messages. All these messages must be properly scheduled on the network so that real-time messages meet their deadlines while co-existing with non-real-time messages (we limit the scope of this paper to scheduling messages whose characteristics like deadline and period are known *a priori*). Previous work regarding scheduling such messages on CAN includes [8], [9], but they focused on fixed-priority scheduling. Shin [10] considered ED scheduling, but did not consider its high overhead which makes ED impractical for CAN. In this paper, we present a dynamic scheduling scheme for CAN called the *mixed traffic scheduler* (MTS) which increases schedulable utilization and performs better than fixed-priority schemes while incurring less overhead than ED.

The next section describes the CAN protocol in detail. Section III describes the various types of messages in our target application workloads. Section IV gives the MTS algorithm and its schedulability conditions. Section V gives simulation results and the paper concludes with Section VI.

The work reported in this paper was supported in part by the NSF under Grants MIP-9203895 and DDM-9313222, and by the ONR under Grant N00014-94-1-0229.

The authors are with the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109-2122

II. Controller Area Network (CAN)

The CAN specification defines the physical and data link layers (layers 1 and 2 in the ISO/OSI reference model). Each CAN frame has seven fields, but we are concerned only with the *identifier* (ID) field. It can be of two lengths: the *standard* format is 11-bits, whereas the *extended* format is 29-bits. It controls both bus arbitration and message addressing, but we are interested only in the former which is described next.

CAN makes use of a wired-OR (or wired-AND) bus to connect all the nodes (in the rest of the paper we assume a wired-OR bus). When a processor has to send a message it first calculates the message ID which may be based on the priority of the message. The ID for each message must be unique to prevent a tie. Processors pass their messages and associated IDs to their bus interface chips. The chips wait till the bus is idle, then write the ID on the bus, one bit at a time, starting with the most significant bit. After writing each bit, each chip waits long enough for signals to propagate along the bus, then it reads the bus. If a chip had written a 0 but reads a 1, it means that another node has a message with a higher priority. If so, this node drops out of contention. In the end, there is only one winner and it can use the bus.

III. Workload characteristics

Some of the devices in CIM exchange periodic messages (such as drives) while others are more event-driven (such as smart sensors). Also, operators may need status information from various devices, thus generating messages which do not have timing constraints. So, we classify messages into three broad categories, (1) hard-deadline periodic messages, (2) hard-deadline sporadic messages, and (3) non-real-time (best-effort) aperiodic messages. A periodic message has multiple invocations, each one period apart (note that whenever we use the term *message* to refer to a periodic, we are referring to *all* invocations of that periodic). Sporadic messages have a minimum interarrival time (MIT) between invocations, while non-real-time messages are completely aperiodic, but they do not have deadline constraints.

Low-speed vs. high-speed real-time messages

Messages in a manufacturing setting can have a wide range of deadlines ranging from tens of microseconds for drive control to several seconds or more for temperature sensors. Thus, we further classify real-time messages into two classes: *high-speed* and *low-speed*, depending on the tightness of their deadlines.

Note that "high-speed" is a relative term — relative to the tightest deadline d_0 in the workload. So, all messages with the same order of magnitude deadlines as d_0 (or within one order of magnitude difference from d_0) can be considered to be high-speed messages. All others will be low-speed.

IV. The Mixed Traffic Scheduler

Fixed-priority deadline monotonic (DM) scheduling [11] can be used for CAN by setting each message's ID to its unique priority as in [8], [9]. However, in general, fixed-priority schemes give lower utilization than other schemes such as non-preemptive earliest-deadline[1] (ED). This motivates us to use ED to schedule messages on CAN, meaning that the message ID must contain the message deadline (actually, the logical inverse of the deadline for a wired-OR bus). But as time progresses, absolute deadline values get larger and larger, and eventually they will overflow the CAN ID. This problem can be solved by using some type of a wrap-around scheme (which we present in Section IV-A), but even then, puting the deadline in the ID forces one to use the extended CAN format with its 29-bit IDs. Compared to the standard CAN format with 11-bit IDs, this wastes 20–30% bandwidth, negating any benefit obtained by going from fixed-priority to dynamic-priority scheduling. This makes ED impractical for CAN.

In this section we present the MTS scheduler which combines ED and fixed-priority scheduling to overcome the problems of ED.

A. Time epochs

As already mentioned, using deadlines in the ID necessitates having some type of a wrap-around scheme. We use a simple scheme which expresses message deadlines relative to a periodically increasing reference called the *start of epoch* (SOE). The time between two consecutive SOEs is called the *length of epoch*, ℓ. Then, the deadline field in the ID of message i will be the logical inverse of $d_i - \mathrm{SOE} = d_i - \left\lfloor \frac{t}{\ell} \right\rfloor \ell$, where d_i is the deadline of message i and t is the current time (it is assumed that all nodes have synchronized clocks). Value of ℓ depends on what fraction of CPU-time the designer is willing to allow for ID updates. Let this fraction be x. Let M be the MIPS of the CPU and n be the number of instructions required to do the update. Then $\ell = \frac{n}{xM \times 10^6}$. So at every node, there is a periodic (timer-driven) process which wakes up every ℓ seconds and updates IDs of all ready messages according to the above equation.

[1] Non-preemptive scheduling under release time constraints is NP-hard in the strong sense [12]. However, Zhao and Ramamritham [13] showed that ED performs better than other simple heuristics.

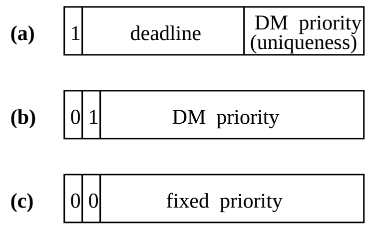

Fig. 1. Parts (a) through (c) show the IDs for high-speed, low-speed, and non-real-time messages, respectively.

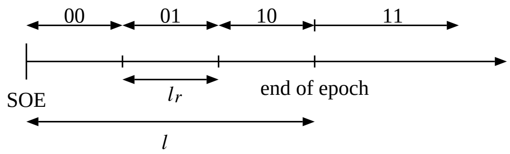

Fig. 2. Quantization of deadlines for $m = 2$.

B. MTS

MTS attempts to give high utilization (like ED) while using the standard 11-bit ID format (like DM). High-speed messages consume most of the bus bandwidth, so the idea behind MTS is to try to use ED for high-speed messages and DM for low-speed ones. First, we give high-speed messages priority over low-speed and non-real-time ones by setting the most significant bit to 1 in the ID for high-speed messages (Figure 1a). This makes sense because high-speed messages have tighter deadlines, so they should have higher priority than low-speed messages.

A uniqueness field is needed within the ID to ensure that no two IDs are the same. Intuitively, its length should be $\lceil \log_2(\text{num high-speed messages}) \rceil$ which would typically be about 5 bits (this is discussed further in Section V). This leaves 5 bits for the deadline field which are not enough to encode message deadlines. Our solution to this problem is to quantize time into *regions* and encode deadlines according to which region they fall in. To distinguish messages whose deadlines fall in the same region, we use the DM-priority of a message as its uniqueness code. This makes MTS a hierarchical scheduler. At the top level is ED: if the deadlines of two messages can be distinguished after quantization, then the one with the earlier deadline has higher priority. At the lower level is DM: if messages have deadlines in the same region, they will be scheduled by their DM priority.

We can calculate length of a region (l_r) as $l_r = \frac{\ell}{2^m - 1}$, where m is the length of the deadline field. This is clear from Figure 2 (shown for $m = 2$). We must reserve one coding for messages whose deadlines fall beyond the end of the current epoch. This leaves $2^m - 1$ codings for deadlines before the end of epoch.

A uniqueness field of 5 bits allows at most 32 real-time messages to be treated as high-speed. To accommodate the remaining (possibly large number of) messages, we use DM scheduling for low-speed messages and fixed-priority scheduling for non-real-time ones, with the latter being as-

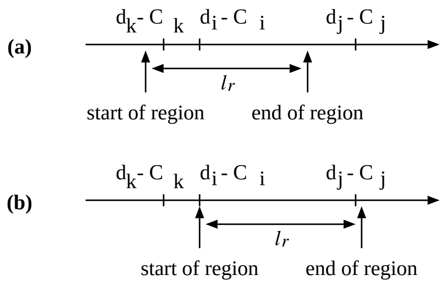

Fig. 3. Suppose j has higher DM-priority than i but k does not. Then in (a), i has the highest priority, whereas in (b), it has the lowest.

signed priorities arbitrarily. The IDs for these messages are shown in Figures 1b and 1c. The second most significant bit gives low-speed messages higher priority than non-real-time ones. In each case, the priority field also acts as the uniqueness code, allowing 512 low-speed and 480 non-real-time messages (32 IDs of non-real-time messages are illegal under the CAN protocol, leaving $512-32 = 480$ valid IDs).

C. Schedulability conditions

For MTS, we want off-line schedulability conditions which, when satisfied, will guarantee that all real-time messages will meet their deadlines. We will first review similar conditions for non-preemptive DM, and then develop those for MTS.

C.1 Deadline monotonic

For the non-preemptive case, a message i is feasible if all higher-priority messages are feasible and message i finds an opportunity to start transmission sometime during $[0, d_i - C_i]$, where C_i is the length of message i. If messages are numbered according to their priority with $j = 0$ being the highest-priority message, then i is schedulable [14] if:

> $\exists t \in S,\ \ \sum_{j=1}^{i-1} \lceil t/T_j \rceil C_j + C_p \leq t$, where S ={set of all release times of messages $0, 1, \cdots, i-1$ through time $d_i - C_i$} $\cup \{d_i - C_i\}$; T_j, C_j, and d_j are the period, length, and deadline of message j; and C_p is the length of the longest possible packet.

C.2 MTS

First, we will discuss the schedulability check for high-speed messages and then look at low-speed ones. The worst-case loading conditions for a high-speed message i result when there is

1. worst possible traffic congestion, and
2. worst possible deadline encoding.

The first situation is created by releasing all messages at the same time $t = 0$. The second occurs when deadline-to-start of i falls at the start of a region as illustrated in Figure 3.

Now, we can draw a parallel between schedulability conditions for MTS and those for DM. To determine schedulability of some message, consider its first invocation i and all j with priority greater than that of i under worst-case situations. Then, from the above discussion, message invocations j will have priority over i if:

1. $(d_i - C_i) > (d_j - C_j)$, or
2. (a) $(d_i - C_i) < (d_j - C_j) \leq (d_i - C_i + l_r)$, and
 (b) DM priority of j is greater than that of i, and
 (c) j is released before $d_i - C_i$.

Note that j represents individual invocations, not entire "messages." Then, if we consider *only* those invocations j which satisfy the above conditions and schedule them according to MTS and the bus ever becomes idle during interval $[0, d_i - C_i]$, then message i will get a chance to run. Formally, a high-speed message is schedulable under MTS if and only if $\sum_{j=1}^{n} C_j/T_j \leq 1$ and its first invocation i satisfies the condition:

> $\exists t \in S,\ \sum$(lengths of all j released before t)$+C_p < t$, where j represents invocations which satisfy the above conditions, S ={set of release times of each j} $\cup \{d_i - C_i\}$, and C_p is the size of a longest possible packet.

Checking schedulability of low-speed messages is simple — just check DM schedulability for each. Since high-speed messages have shorter deadlines than low-speed ones, they will automatically have higher DM priority (which is exactly what we want).

V. Evaluation

We have designed MTS to offer better schedulability than DM. Since deadlines in MTS are quantized, we would expect the performance of MTS to be close to that of ED if somehow message length did not increase when using ED. So, let ED* be an ideal (imaginary) scheduling policy which works the same as ED but requires only an 11-bit ID. Then ED*'s performance should be an upper bound on MTS's.

A. Simulation workload model

Consider a carriage-mounted drilling machine with an attached triple-jointed robot arm to move workpieces. Suppose the robot has a two-fingered gripper. This gives a total of seven drives which must be controlled to varying degrees of precision as shown in Table I (note that a pair of messages are needed to control each drive such that if the first is released at t_0, the second is released at $t_0 + 0.5c$, where c is the drive control cycle time).

For each finger there is a smart contact sensor which informs the controller when the gripper makes contact with the object being grasped. For simulation, assume that the controller must receive this message within one-fourth of the drive cycle time.

For low-speed messages, we arbitrarily chose periodic messages to have periods of 20ms and deadlines of 8ms, and sporadic messages with deadlines of 5ms and MITs of 5s.

Table I summarizes the choice of parameters. The values marked with "$\star$" are default and will be varied during simulation.

We chose message lengths as 79 bits for periodic messages (including 32 data bits) and 47 bits for sporadic messages

TABLE I

Simulation workload.

High-speed messages				
Type	*Class*	*Period/MIT*	*Deadline*	*# of mssg.*
Fingers	Periodic	125.0μs (8 kHz)	50.0μs	4
Joints	Periodic	166.7μs (6 kHz)	66.6μs*	6*
Carriage	Periodic	250.0μs (4 kHz)	100.0μs	2
Drill	Periodic	500.0μs (2 kHz)	200.0μs	2
Sensors	Sporadic	2s	30.0μs*	2*

Low-speed messages			
Class	*Period* (ms)	*Deadline* (ms)	*MIT*
Periodic	20.0	8.0	—
Sporadic	—	5.0	5s

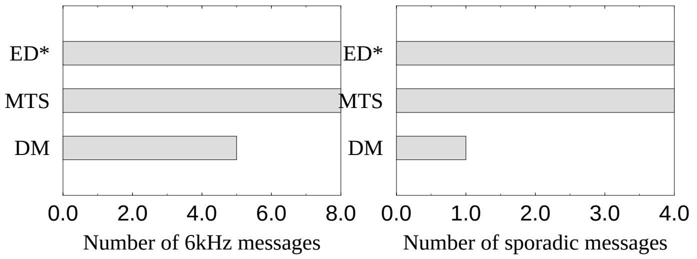

Fig. 4. Schedulability under DM, MTS, and ED* as number of messages are varied.

(0 data bits, since these messages are for notification only). Also, if we assume a 20 MIPS CPU, deadline updates requiring 1000 instructions, and allow 5% of CPU time for these updates, then $\ell = \frac{1000}{(0.05)(20\times10^6)} = 1$ms.

B. Simulation results

Current CAN chips are designed for a 1 Mb/s physical medium. This speed is insufficient for applications like high-speed servoing, so in our simulations, we assume a 10 Mb/s physical medium (which implies a time granularity of 0.1μs).

In all our simulations, we use the general drilling machine workload described in the previous subsection. Using the schedulability conditions of Section IV-C, we want to check schedulability of workloads under different scheduling policies when one workload parameter (such as the number of a certain type of message) is varied. To check schedulability under ED*, we use the schedulability check for non-preemptive ED in [15].

B.1 Varying number of high-speed periodics

We varied the number of 6 kHz periodic messages. The results are shown in Figure 4. DM can handle only five 6 kHz messages (total utilization of 58.5%), whereas both ED* and MTS can handle up to 8 each (72.7% utilization). At least for this workload, MTS performed better than DM and same as ED*.

B.2 Varying number of high-speed sporadics

Our simulations showed that DM can handle only one sporadic message in the workload, whereas both MTS and ED* handled up to four (Figure 4).

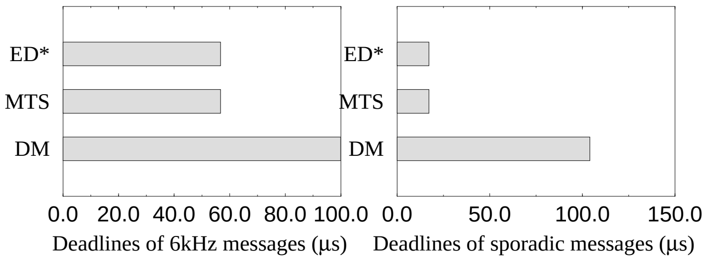

Fig. 5. Deadlines of messages for which workload is *unschedulable* under DM, MTS, and ED*.

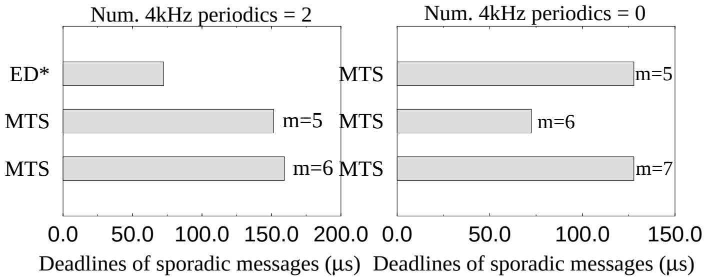

Fig. 6. Deadlines of sporadic messages for which workload is *unschedulable* under MTS and ED* (number of 6 kHz periodics is 10).

B.3 Varying deadlines of high-speed periodics

We varied the deadlines of 6 kHz periodic messages while keeping their number fixed at 6, giving a utilization of 63.2%. This workload was found unschedulable under DM at the deadlines of 99.9μs or less (Figure 5). On the other hand, the workload was schedulable under both MTS and ED* till the deadline of 56.8μs.

B.4 Varying deadlines of high-speed sporadics

In this simulation, we vary the deadlines of all high-speed sporadic messages. The workload becomes unschedulable under DM for sporadic message deadlines of 104.1μs or lower (Figure 5), but continues to be feasible under both MTS and ED* till the deadline of 17.3μs.

To better compare MTS against ED*, we tested the two under heavier loads. We increased the number of 6 kHz messages to 10 (82.2% utilization), then started varying sporadic message deadlines. This workload was unschedulable under MTS for deadlines of 151.5μs or less, whereas it became unschedulable under ED* only at deadlines of 72.5μs or less (Figure 6–left). So under heavy loads and tight deadlines, ED* outperforms MTS, as expected.

Earlier we stated intuitively that m should be (length of ID $-1-$ $\lceil \log_2(\text{num high-speed messages}) \rceil$) which is 5 for this workload. It turns out that the best value for m is really application-dependent. Figure 6 shows two workloads. For one workload (with two 4 kHz periodics), m=5 is the best value but not for the other. In the second workload (with no 4 kHz periodics), for $m = 6$, the uniqueness field is only 4 bits which forces us to treat two high-speed messages as low-speed. This decreases schedulability but is more than offset (in this particular case) by a longer deadline field. But when m is increased further to 7, too many high-speed messages have to be treated as low-speed, negating any benefit of a longer deadline field. In short, the intuitive formula for m should be used only as a starting point to find the optimal value of m through simulation for a given workload.

B.5 Low-speed messages

Under default loading conditions, high-speed messages use up 63.2% of the bandwidth. The remaining 36.8% can accommodate several hundred low-speed sporadic and periodic messages. Thus, the schedulability of low-speed messages is not a problem. Our simulations showed no real difference between DM, MTS, or ED* in scheduling low-speed messages for a fixed load of high-speed messages.

VI. Conclusion

The two most attractive features of CAN are a short worst-case bus access latency and a bus acquisition scheme based on the priority of messages, both of which give CAN the potential for high performance and fewer missed deadlines in distributed control systems. However, the bus arbitration mechanism must be used properly with careful design of the message ID; otherwise, CAN will give low utilization.

In this paper we presented the MTS scheduler which allowed three different types of messages — hard sporadic, hard periodic, and non-real-time aperiodic — to be carried on the same bus. Then, we designed the message ID which implements MTS on CAN. MTS not only gives high schedulability but is also easily implementable on CAN. Through simulations we compared MTS with DM and ED*. As expected, MTS performed much better than DM and only slightly worse than ED*, and in many cases, it matched the performance of ED*.

References

[1] Reza S. Raji, "Smart networks for control," *IEEE Spectrum*, vol. 31, no. 6, pp. 49–55, June 1994.

[2] Gerald W. Lenhart, "A field bus approach to local control networks," *Advances in Instrumentaion and Control*, vol. 48, no. 1, pp. 357–365, 1993.

[3] *Road vehicles — Interchange of digital information — Controller area network (CAN) for high-speed communication. ISO 11898*, 1st edition, 1993.

[4] Instrument Society of America, *Industrial Automation Systems — Systems Integration and Communication — Fieldbus (draft) (ISA/SP50-93)*, 1993.

[5] MAP/TOP Users Group, Dearborn, MI, *Manufacturing Automation Protocol (MAP) 3.0 Implementation Release*, 1987.

[6] Hermann Kopetz and Gunter Grunsteidl, "TTP – a protocol for fault-tolerant real-time systems," *IEEE Computer*, vol. 27, no. 1, pp. 14–23, January 1994.

[7] Holger Zeltwanger, "An inside look at the fundamentals of CAN," *Control Engineering*, vol. 42, no. 1, pp. 81–87, January 1995.

[8] K. W. Tindell, H. Hansson, and A. J. Wellings, "Analyzing real-time communications: Controller Area Network (CAN)," in *Proc. Real-Time Systems Symposium*, Dec. 1994, pp. 259–263.

[9] K. Tindell, A. Burns, and A. J. Wellings, "Calculating Controller Area Network (CAN) message response times," *Control Engineering Practice*, vol. 3, no. 8, pp. 1163–1169, 1995.

[10] Kang G. Shin, "Real-time communications in a computer-controlled workcell," *IEEE Trans. Robotics and Automation*, vol. 7, no. 1, pp. 105–113, February 1991.

[11] Joseph Y.-T. Leung and Jennifer Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Performance Evaluation*, vol. 2, no. 4, pp. 237–250, December 1982.

[12] Kevin Jeffay, Donald F. Stanat, and Charles U. Martel, "On non-preemptive scheduling of periodic and sporadic tasks," in *Proc. Real-Time Systems Symposium*, 1991, pp. 129–139.

[13] W. Zhao and K. Ramamritham, "Simple and integrated heuristic algorithms for scheduling tasks with time and resource constraints," *Jounal of Systems and Software*, vol. 7, pp. 195–205, 1987.

[14] Dilip D. Kandlur, Kang G. Shin, and Domenico Ferrari, "Real-time communication in multihop networks," *IEEE Trans. Parallel and Distributed Systems*, vol. 5, no. 10, pp. 1044–1056, October 1994.

[15] Qin Zheng and K. G. Shin, "On the ability of establishing real-time channels in point-to-point packet-switched networks," *IEEE Trans. Communications*, pp. 1096–1105, February/March/April 1994.