CAN Bus Scheduling

CPEN 432 Real-Time System Design

Arpan Gujarati
University of British Columbia

Homework and Programming Assignments

- Inform us if you plan to use the skip days!

Recap: Response-Time Analysis

- Fixed-priority scheduling (RM, DM, …) with preemptions
- Tasks $\tau = \{\tau_1, \tau_2, \ \ldots, \tau_n\}$
  - Each task τ_i has time period T_i, completion time C_i, relative deadline D_i
  - Tasks IDs are used as priorities
- Solve the following recurrence for each τ_i to obtain its worst-case response time R_i
  - $R_i = C_i + \sum_{a<i} \left(\left\lceil \frac{R_i}{T_a} \right\rceil \cdot C_a \right)$
- Verify that either
  - $\forall \tau_i \in \tau : R_i \leq D_i$ (the task set is schedulable), or
  - $\exists \tau_k \in \tau : R_k > D_k$ (the task set is not schedulable)

The Controller Area Network (CAN) Bus

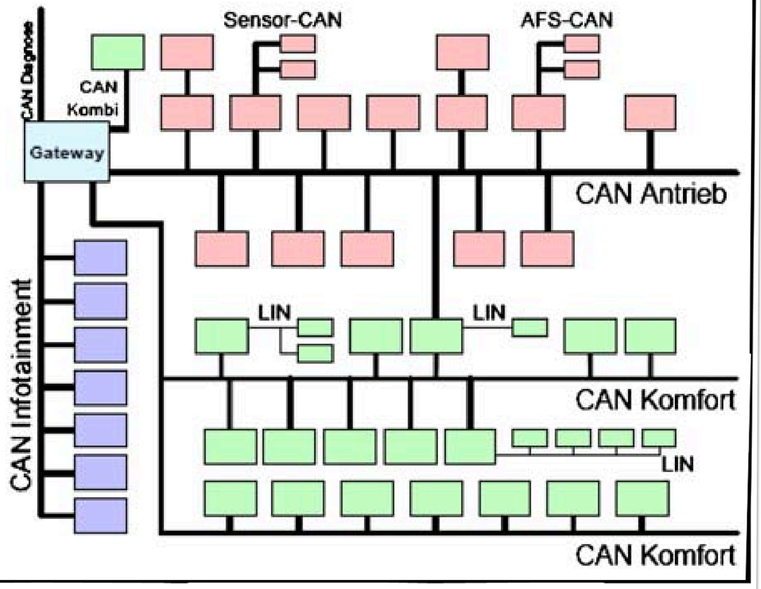
- CAN was designed (by Bosch 1983-1987) as a real-time bus to *reduce the number of cables* required in modern cars.
  ➞ Multiplex many signals on a few wires.
- Now used in many areas, including automotive, robotics, factory automation, etc.
- A comprehensive treatment of CAN is a course topic in itself.
- Focus here: how to carry out schedulability analysis.

CAN Messages have Real-Time Constraints

“An ECU reads the position of a switch attached to the brake pedal. This ECU must send a message over the CAN network, carrying information (a signal) that the brakes have been applied. The ECU responsible for the rear light clusters receives the message, recognises the change in the value of the signal, and switches the brake lights on. All within a few tens of milliseconds of the brake pedal being pressed.”

“Engine, transmission, and stability control systems typically place even tighter time constraints on signals, which may need to be sent as frequently as once every 5 milliseconds to meet their time constraints”

Fig. 2 VW Passat network architecture

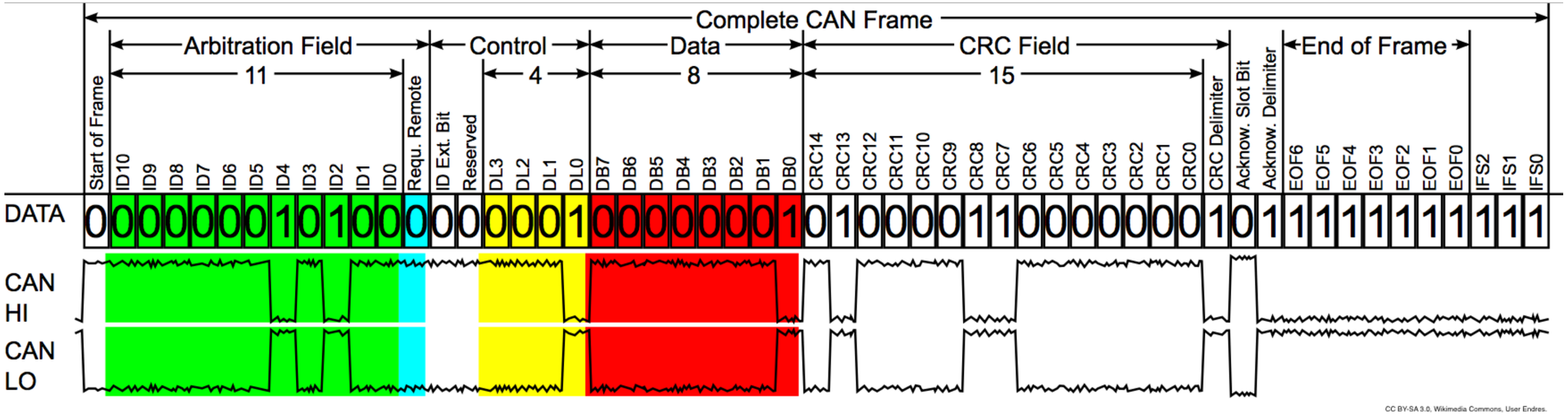


Real-Time Syst (2007) 35:239–272
DOI 10.1007/s11241-007-9012-7

Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised

Robert I. Davis · Alan Burns · Reinder J. Bril · Johan J. Lukkien

CAN Packets & Bus Arbitration



Carrier Sense Multiple Access / **Collision Resolution** (CSMA/CR)

Hosts start sending simultaneously ➔ lowest-ID message wins.

Non-Preemptive Fixed-Priority Scheduling

- Just like regular fixed-priority scheduling…
  ➞ Message types = tasks
  ➞ message = job
  ➞ WCET = message length / bit rate
  ➞ period, deadline, jitter, phase as before.

- …but lower-priority executing tasks / messages in transmission **cannot be preempted**.
  ➞ Need to account for *priority inversions*.

Classic Analysis (for Constrained Deadlines)

Intuition: It's just fixed-priority RTA with maximum priority inversion length ≈ maximum lower-priority message length.

$$R_i = C_i + B_i + \sum_{a<i} \left(\left\lceil \frac{R_i}{T_a} \right\rceil \cdot C_a \right)$$

where $B_i = \max_{b>i} C_b$.

New "blocking" term for non-preemptive scheduling

Approach in the previous lecture …

- Define recurrence function using C_i and B_i
  - $$R_i^{(n+1)} = C_i + B_i + \sum_{a<i} \left(\left\lceil \frac{R_i^{(n)}}{T_a} \right\rceil \cdot C_a \right)$$
- For non-preemptive scheduling, we can move C_i outside the recurrence
  - Define $R_i = C_i + W_i$ such that W_i denotes the maximum waiting time after release
  - W_i is obtained by solving the recurrence $W_i^{(n+1)} = B_i + \sum_{a<i} \left(\left\lceil \frac{W_i^{(n)}}{T_a} \right\rceil \cdot C_a \right)$

Example

The task set is schedulable!

- Messages with identical parameters can be successfully transmitted over CAN

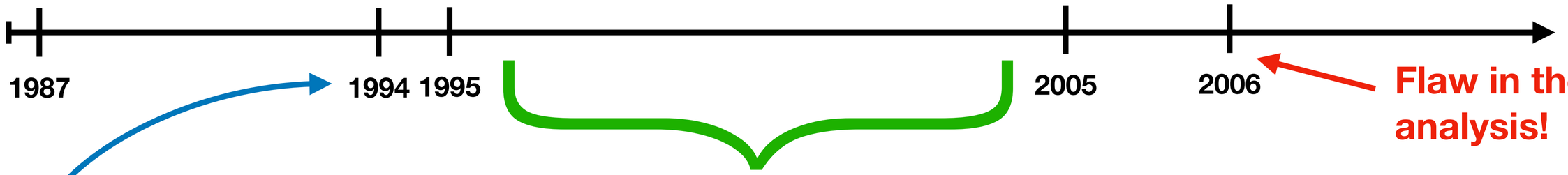
Task ID	T	D	C	Priority
1	2.5	2.5	1	1 (high)
2	3.5	3.25	1	2 (medium)
3	3.5	3.25	1	3 (low)

- $R_1 = C_1 + W_1 = C_1 + B_1 = ?$
  - $1ms + 1ms = 2ms?$
- $R_2 = C_2 + W_2$
  - $W_2^{(1)} = \max_{b>2}(C_b) + \sum_{a<2}\left(\left\lceil \frac{W_2^{(0)}}{T_a} \right\rceil \cdot C_a\right)$
  - $W_2^{(1)} = C_3 + \left\lceil \frac{W_2^{(0)}}{T_1} \right\rceil \cdot C_1$
  - Starting with $W_2^{(0)} = 1$ (why?)
    - $W_2^{(1)} = C_3 + \left\lceil \frac{W_2^{(0)}}{T_1} \right\rceil \cdot C_1 = 1 + \left\lceil \frac{1}{2.5} \right\rceil \cdot 1 = 2ms$
    - $W_2^{(2)} = C_3 + \left\lceil \frac{W_2^{(1)}}{T_1} \right\rceil \cdot C_1 = 1 + \left\lceil \frac{2}{2.5} \right\rceil \cdot 1 = 2ms$
  - We have a fixed point! $R_2 = C_2 + W_2 = 1 + 2 = 3ms?$
- $R_3 = C_3 + W_3$
  - $W_3^{(1)} = \max_{b>2}(C_b) + \sum_{a<2}\left(\left\lceil \frac{W_3^{(0)}}{T_a} \right\rceil \cdot C_a\right)$
  - $W_3^{(1)} = \left\lceil \frac{W_3^{(0)}}{T_1} \right\rceil \cdot C_1 + \left\lceil \frac{W_3^{(0)}}{T_2} \right\rceil \cdot C_2$
  - Starting with $W_3^{(0)} = 2$ (why?)
    - $W_3^{(1)} = \left\lceil \frac{2}{2.5} \right\rceil \cdot 1 + \left\lceil \frac{2}{3.5} \right\rceil \cdot 1 = 1 + 1 = 2ms$
  - We have a fixed point already! $R_3 = C_3 + W_3 = 1 + 2 = 3ms?$

Timeline

"In the early 1990s, a common misconception about CAN was that although the protocol was very good at transmitting the highest priority message with low latency, it was ***not possible to guarantee that less urgent signals, carried in lower priority messages, would meet their deadlines.****"* [emphasis added]

"In 1995, Tindell's research was ***recognised by Volvo Car Corporation*** *and successfully used in the configuration and analysis of the CAN buses for the forthcoming Volvo S80 (P23) (Casparsson et al., 1998). Following the success of this project, Volcano Communications Technologies AB3 used* ***Tindell's analysis as the basis of a commercial CAN schedulability analysis tool****, called Volcano Network Architect."* [emphasis added]



"Prior to Tindell's work, low levels of bus utilization, up to 30 or 40%, were typical in automotive applications, with extensive testing required to obtain confidence that CAN messages would meet their deadlines. With the advent of a systematic approach based on schedulability analysis, ***CAN bus utilization could be increased to around 80%*** *(DeMeis, 2005) whilst still guaranteeing that deadlines would be met."* [emphasis added]

"Tindell and Burns (1994) and Tindell et al. (1994b, 1995) ***showed how research into fixed priority pre-emptive scheduling for single processor systems could be adapted*** *and applied to the scheduling of messages on CAN. This analysis provided a method of calculating the worst-case response times of all CAN messages. Using this analysis it became possible to engineer CAN based systems for timing correctness,* ***providing guarantees that all messages****, and the signals that they carry, would meet their deadlines."* [emphasis added]

Classic Analysis (for Constrained Deadlines)

Intuition: It's just fixed-priority RTA with maximum priority inversion length ≈ maximum lower-priority message length.

$$R_i = C_i + B_i + \sum_{a<i} \left(\left\lceil \frac{R_i}{T_a} \right\rceil \cdot C_a \right)$$

where $B_i = \max_{b>i} C_b$. Reinder Bril[Bril06] showed this to be **wrong**...

[Bril06] R. Bril (2006). Existing worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred pre-emption is too optimistic. CS-Report 06-05, Technische Universiteit Eindhoven.

Example

The task set is schedulable!

- Messages with identical parameters can be successfully transmitted over CAN

Task ID	T	D	C	Priority
1	2.5	2.5	1	1 (high)
2	3.5	3.25	1	2 (medium)
3	3.5	3.25	1	3 (low)

Our analysis showed that $R_1 = 2ms, R_2 = 3ms, R_3 = 3ms$

Let's see why the analysis is wrong …

Revised Analysis

- Naive classic analysis does not reflect that higher-priority demand is “pushed through” due to non-preemptive execution
- Solution: Consider the response times of all messages after the critical instant, not just the first
- Read the following paper if you are interested!
- There may be a homework assignment question :-)

Real-Time Syst (2007) 35:239–272
DOI 10.1007/s11241-007-9012-7

Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised

Robert I. Davis · Alan Burns · Reinder J. Bril · Johan J. Lukkien

Published online: 30 January 2007

Abstract Controller Area Network (CAN) is used extensively in automotive applications, with in excess of 400 million CAN enabled microcontrollers manufactured each year. In 1994 schedulability analysis was developed for CAN, showing how worst-case response times of CAN messages could be calculated and hence guarantees provided that message response times would not exceed their deadlines. This seminal research has been cited in over 200 subsequent papers and transferred to industry in the form of commercial CAN schedulability analysis tools. These tools have been used by a large number of major automotive manufacturers in the design of in-vehicle networks for a wide range of cars, millions of which have been manufactured during the last decade.

This paper shows that the original schedulability analysis given for CAN messages is flawed. It may provide guarantees for messages that will in fact miss their deadlines in the worst-case. This paper provides revised analysis resolving the problems with the original approach. Further, it highlights that the priority assignment policy, previously claimed to be optimal for CAN, is not in fact optimal and cites a method of obtaining an optimal priority ordering that is applicable to CAN. The paper discusses the possible impact on commercial CAN systems designed and developed using flawed schedulability analysis and makes recommendations for the revision of CAN schedulability analysis tools.

R. I. Davis (✉) · A. Burns
Real-Time Systems Research Group, Department of Computer Science, University of York, YO10 5DD, York, UK
e-mail: rob.davis@cs.york.ac.uk

A. Burns
e-mail: alan.burns@cs.york.ac.uk

R. J. Bril · J. J. Lukkien
Technische Universiteit Eindhoven (TU/e), Den Dolech 2, 5600 AZ Eindhoven, The Netherlands
e-mail: r.j.bril@tue.nl

J. J. Lukkien
e-mail: j.j.lukkien@tue.nl

Resource Sharing

Task Coordination and Synchronization

So far we have assumed that tasks are *independent*. However, this often not the case in practice.

Two types of coordination:

1. **mutual exclusion constraints** — resource sharing with locks
   → `mutex_lock()`/`mutex_unlock()`
2. **precedence constraints** — when one job must wait for another
   → producer-consumer or `signal()`/`wait()` relationships
   → `pipe()` or `socket()` + blocking `read()`

Mutual Exclusion

*How to deal with delays due to **locking**?*

Need for Mutual Exclusion

Classic problem: prevent the interleaving of *critical sections* to ensure *atomicity* of multiple read/write accesses.

Examples of **shared resources**:

- accessing control registers of an I/O device
- shared OS services (e.g., timer facility, run queue)
- shared data structures (e.g., history of sensor data)
- message buffers (e.g., `pipe()` implementation)

How to Realize Mutual Exclusion

Three main options:

- use a **static schedule** that prevents interleaving of accesses
  ➞ good solution when possible
- use **locks** (or *binary semaphores*) to *block* interleaving of accsesses
  ➞ *need to analyze extra delays due to blocking!* (= blocking analysis)
- restrict access to a single sequential task (a **server task**) and invoke resource server via *inter-process communication* (**IPC**)
  ➞ *need to analyze delay due to message backlog!* (= blocking analysis)

The Key Locking Problem — Priority Inversion

Priority-based scheduling: at any time, the highest-priority incomplete job should be scheduled.
→ this assumption is the basis of all schedulability analysis!

Problem: what if the highest-priority job requires a lock?

Priority inversion: a job *should* be scheduled but *is* not.
→ On a uniprocessor: a **lower-priority** job is scheduled instead.
→ Only possible if some lower-priority job holds a **required lock**

Priority Inversion Example

- No useful response times with “unbounded” priority inversions!