Real-Time Syst (2013) 49:73–116
DOI 10.1007/s11241-012-9167-8

Schedulability analysis for Controller Area Network (CAN) with FIFO queues priority queues and gateways

Robert I. Davis · Steffen Kollmann · Victor Pollex · Frank Slomka

Published online: 10 November 2012

Abstract Controller Area Network (CAN) is widely used in automotive applications. Existing schedulability analysis for CAN is based on the assumption that the highest priority message ready for transmission at each node on the network will be entered into arbitration on the bus. However, in practice, some CAN device drivers implement FIFO rather than priority-based queues invalidating this assumption. In this paper, we introduce response time analysis and optimal priority assignment policies for CAN messages in networks where some nodes use FIFO queues while other nodes use priority queues. We show, via a case study and experimental evaluation, the detrimental impact that FIFO queues have on the real-time performance of CAN. Further, we show that in gateway applications, if it is not possible to implement a priority queue, then it is preferable to use multiple FIFO queues each allocated a small number of messages with similar transmission deadlines.

Keywords Controller Area Network (CAN) · Real-time scheduling · Schedulability analysis · FIFO · Priority assignment

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

S. Kollmann · V. Pollex · F. Slomka
Institute of Embedded Systems/Real-Time Systems, Ulm University, Albert-Einstein-Allee 11, 89081 Ulm, Germany

S. Kollmann
e-mail: steffen.kollmann@uni-ulm.de

V. Pollex
e-mail: victor.pollex@uni-ulm.de

F. Slomka
e-mail: frank.slomka@uni-ulm.de

1 Extended version

This paper forms an extended version of "*Controller Area Network (CAN) Schedulability Analysis with FIFO queues*" by Davis et al. (2011) published in ECRTS. The analysis given in that paper has been extended via the inclusion of the following new material:

- In Sect. 2.2 we have added examples of CAN devices that provide hardware support for FIFO queues.
- Sect. 5.6 has been added, providing formal proofs that the schedulability tests given in Sects. 5.1, 5.2 and 5.3 are sufficient (Theorems 2 and 3) and self-sustainable (Theorems 4 and 5). This section also shows how more precise analysis can be achieved when the priorities of messages in a FIFO queue span those of messages in a priority queue or another FIFO queue, which is often the case in practice.
- In Sect. 6.2, we have added a formal proof that transmission deadline monotonic priority ordering is optimal when all messages have the same maximum transmission time (Theorem 7).
- In Sect. 8, we have extended the experimental evaluation to show how the performance degradation due to FIFO queues depends on the number of messages in each queue.
- Sects. 7.1 and 8.1 have been added, exploring the effects of implementing one or more FIFO queues in gateway nodes that are responsible for transferring messages from one network to another.

2 Introduction

Controller Area Network (CAN) (Bosch 1991; ISO 11898-1 1993) was designed as a simple, efficient, and robust, broadcast communications bus for in-vehicle networks. Today, typical mainstream family cars contain 25–35 Electronic Control Units (ECUs), many of which communicate using CAN. As a result of this wholesale adoption of CAN by the automotive industry, annual sales of CAN nodes (8, 16 and 32-bit micro-controllers with on-chip CAN controllers) have grown from under 50 million in 1999 to around 750 million in 2010.[1]

In automotive applications, CAN is typically used to provide high speed networks (500 Kbits/s) connecting chassis and power-train components, for example engine management and transmission control. It is also used for low speed networks (100 or 125 Kbits/s) connecting body and comfort electronics. Data required by nodes on different networks is typically transferred between the different CAN buses by a *gateway* node connected to both.

CAN is an asynchronous multi-master serial data bus that uses Carrier Sense Multiple Access/Collision Resolution (CSMA/CR) to determine access to the bus. The CAN protocol requires that nodes wait for a bus idle period before attempting to transmit. If two or more nodes attempt to transmit messages at the same time, then

[1] Figures from the CAN in Automation (CiA) website www.can-cia.org.

the node with the message with the lowest numeric CAN Identifier will win arbitration and continue to send its message. The other nodes will cease transmitting and must wait until the bus becomes idle again before attempting to re-transmit their messages. (Full details of the CAN physical layer protocol are given by Bosch (1991), with a summary given by Davis et al. (2007). In effect CAN messages are sent according to fixed priority non-pre-emptive scheduling, with the identifier (ID) of each message acting as its priority.

2.1 Related work

Tindell and Burns (1994) showed how research into fixed priority scheduling for single processor systems could be adapted and applied to the scheduling of messages on CAN. The analysis of Tindell et al. provided a method of calculating the maximum queuing delay and hence the worst-case response time of each message on the network. Tindell and Burns (1994), Tindell et al. (1994, 1995) also recognised that with fixed priority scheduling, an appropriate priority assignment policy is key to obtaining effective real-time performance. Tindell et al. suggested that messages should be assigned priorities in 'Deadline minus Jitter' monotonic priority order (Zuhily and Burns 2007).

The seminal work of Tindell et al. lead to a large body of research into scheduling theory for CAN (Rufino et al. 1998; Broster et al. 2002, 2005; Broster and Burns 2003; Broster 2003; Ferreira et al. 2004; Hansson et al. 2002; Nolte et al. 2002, 2003; Nolte 2006), and was used as the basis for commercial CAN schedulability analysis tools (Casparsson et al. 1998).

Davis et al. (2007) found and corrected significant flaws in the schedulability analysis given by Tindell and Burns (1994) and Tindell et al. (1994, 1995). These flaws could potentially result in the original analysis providing guarantees for messages that could in fact miss their deadlines during network operation. Further, Davis et al. (2007) showed that the 'Deadline minus Jitter' monotonic priority ordering, claimed by Tindell et al. to be optimal for CAN, is not in fact optimal; and that Audsley's Optimal Priority Assignment (OPA) algorithm (Audsley 1991, 2001) is required in this case.

Prior to the advent of schedulability analysis and appropriate priority assignment policies for CAN, message IDs were typically assigned simply as a way of identifying the data and the sending node. This meant that only low levels of bus utilisation, typically around 30 %, could be obtained before deadlines were missed. Further, the only means of obtaining confidence that message deadlines would not be missed was via extensive testing. Using the systematic approach of schedulability analysis, combined with a suitable priority assignment policy, it became possible to engineer CAN based systems for timing correctness, providing guarantees that all messages would meet their deadlines, with bus utilisations of up to about 80 % (Davis and Burns 2009a; Casparsson et al. 1998).

2.2 Motivation

Engineers using schedulability analysis to analyse network/message configurations must ensure that all of the assumptions of the specified scheduling model hold for

their particular system. Specifically, when using the analysis given by Davis et al. (2007), it is important that each CAN controller and device driver is capable of ensuring that whenever message arbitration starts on the bus, the highest priority message queued at that node is entered into arbitration. This behaviour is essential if message transmission is to take place as if there were a single global priority queue and for the analysis to be correct.

As noted by Di Natale (2008), there are a number of potential issues that can lead to behaviour that does not match that required by the scheduling model given by Davis et al. (2007). For example, if a CAN node has fewer transmit message buffers than the number of messages that it transmits, then the following properties of the CAN controller hardware can prove problematic:

(i) internal message arbitration based on transmit buffer number rather than message ID (Fujitsu MB90385/90387, Fujitsu 90390, Intel 87C196 (82527), Infineon XC161CJ/167 (82C900));
(ii) non-abortable message transmission: Philips 82C200 (Di Natale 2006);
(iii) less than 3 transmit buffers: Philips 8xC592 (SJA1000), Philips 82C200 (Meschi et al. 1996).

CAN controllers which avoid these potential problems include, the Atmel AT89C51CC03/AT90CAN32/64 the Microchip MPC2515, and the Motorola MSCAN on-chip peripheral, all of which have at least 3 transmit buffers, internal message arbitration based on message ID rather than transmit buffer number, and abortable message transmission.

The CAN device driver/software protocol layer implementation also has the potential to result in behaviour which does not match that required by the standard scheduling model (Davis et al. 2007). Issues include, delays in refilling a transmit buffer (Khan et al. 2010), and FIFO queuing of messages in the device driver or CAN controller.

A number of CAN controller hardware implementations provide specific support for FIFO queues. These include:

- 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 (STMicroelectronics 2001).
- The XILINX CAN Controller Core (LogiCORE IP AXI Controller) which provides a transmit buffer FIFO of configurable depth (up to 64 messages) and a single additional high priority transmit buffer that takes precedence over the FIFO (XILINX 2010).
- The Microchip PIC32MX (Microchip Technology Inc. 2009) which has 32 FIFOs each of which can hold up to 32 messages. Arbitration between the individual FIFOs takes place on the basis of a priority assigned to each FIFO or the FIFO number in the case of ties, hence all of the messages in a high priority FIFO are sent before any of the messages in a lower priority FIFO. (We note that as there are 32 FIFOs, the PIC32MX can effectively provide priority-based queuing for up to 32 transmit messages, each utilising an individual FIFO.)
- The Avnet MC-ACT-XCANF which is a small FPGA footprint CAN Controller for use with Actel programmable logic devices (Avnet 2006). The MC-ACT-XCANF has a single transmit FIFO and a single receive FIFO.

- The Renesas R32C/160 (Renesas 2010) is a microcontroller from the M16C family, specific to vehicle network applications. The on-chip CAN peripheral has 32 message buffers/mailboxes and provides the option of a FIFO mailbox mode. In this mode, 4 mailboxes are configured as a 4-stage transmit FIFO and 4 mailboxes as a 4-stage receive FIFO. Otherwise the buffers may be configured for transmission based on either message priority or buffer number.

We note that the more sophisticated CAN controllers offer the option of hardware support for FIFO queues while also fully supporting priority queues, thus leaving the choice of which queuing policy to use up to the device driver/software protocol layer implementation.

Di Natale (2008) noted that using FIFO queues in CAN device drivers/software protocol layers can seem an attractive solution, "*because of its simplicity and the illusion that faster queue management improves the performance of the system*". This is unfortunate, because FIFO message queues undermine the priority-based bus arbitration used by CAN. They can introduce significant priority inversion and result in degraded real-time performance. Nevertheless, FIFO queues are a reality in some commercial CAN device drivers/software protocol layers.

One area in which the use of FIFO queues can have a particularly detrimental effect is in gateway applications. The number of messages transmitted onto a network by a gateway node can easily exceed the number of hardware transmit buffers available in the CAN controller it uses. A simple design solution to this problem is to use a single FIFO queue for all of these messages; however, such a choice can significantly degrade the real-time performance of the network.

As far as we are aware, there is no published research[2] integrating FIFO queues into response time analysis for CAN. This paper focuses on the issue of FIFO queues. We provide response time analysis and appropriate priority assignment policies for Controller Area Networks comprising some nodes that use FIFO queues and other nodes that use priority queues.

2.3 Organisation

The remainder of this paper is organised as follows: In Sect. 3, we introduce the scheduling model, notation, and terminology used in the rest of the paper. In Sect. 4 we recap on the sufficient schedulability analysis for CAN given by Davis et al. (2007). Section 5 then extends this analysis to networks where some nodes implement priority-based queues while others implement FIFO queues. Section 6 discusses priority assignment for mixed sets of FIFO-queued and priority-queued messages. Section 7 presents the results of a case study exploring the impact of FIFO queues on message response times and network schedulability. Section 8 evaluates the effect of priority assignment and FIFO queues on the maximum achievable network utilisation. Finally, Sect. 9 concludes with a summary and recommendations.

[2]The commercial tool NETCAR-Analyzer (www.realtimeatwork.com) addresses the case of FIFO queues.

3 System model, notation and terminology

In this section we describe a system model and notation that can be used to analyse the worst-case response times of CAN messages. This model is based on that used by Davis et al. (2007) with extensions to describe FIFO queues. A summary of the notation used is given in Table 1 for easy reference. Here we give only a high level description necessary to understand the message scheduling behaviour of CAN. Readers interested in the underlying lower level CAN protocol and its terminology are directed to Sect. 2.1 of Davis et al. (2007).

The system is assumed to comprise a number of nodes (microprocessors) connected to a single CAN bus. Nodes are classified according to the type of message queue used in their device driver. Thus *FQ-nodes* implement a FIFO message queue, whereas *PQ-nodes* implement a priority queue. PQ-nodes are assumed to be capable of ensuring that, at any given time when bus arbitration starts, the highest priority message queued at the node is entered into arbitration. 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 system is assumed to contain a static set of hard real-time messages, each statically assigned to a single node on the network. Each message m has a distinct fixed Identifier (ID) and hence a unique priority. As priority uniquely identifies each message, in the remainder of the paper we will overload m to mean either message m or priority m as appropriate. We use $hp(m)$ to denote the set of messages with

Table 1 Notation

Symbol	Meaning
B_m	Blocking factor at priority m.
C_m	Longest transmission time of message m.
C_m^{MAX}	Max. transmission time of a message in $M(m)$.
C_m^{MIN}	Min. transmission time of a message in $M(m)$.
C_m^{SUM}	Sum of transmission times of messages in $M(m)$.
D_m	Deadline of message m.
E_m	Transmission deadline of message m.
E_m^{MIN}	Min. transmission deadline of any message in $M(m)$
f_m	Buffering delay for message m.
$hp(m)$	Set of messages with higher priority than message m.
J_m	Release jitter of message m.
$lp(m)$	Set of messages with lower priority than message m.
L_m	Lowest priority of any message in $M(m)$.
m	Message m (also its priority).
$M(m)$	The set of messages sharing a FIFO queue with message m.
R_m	Worst case response time for message m.
τ_{bit}	Transmission time for one bit.
T_m	Minimum inter-arrival time or period of message m.
w_m	Queuing delay for message m.

priorities higher than m, and similarly, $lp(m)$ to denote the set of messages with priorities lower than m.

Each message m has a maximum transmission time of C_m (see Davis et al. 2007 for details of how to compute the maximum transmission time of messages on CAN, taking into account the number of data bytes and bit-stuffing).

The event that triggers queuing of message m is assumed to occur with a minimum inter-arrival time of T_m, referred to as the message *period*. Each message m has a hard *deadline* D_m, corresponding to the maximum permitted time from occurrence of the initiating event to the end of successful transmission of the message, at which time the message data is assumed to be available on the receiving nodes that require it. Tasks on the receiving nodes may place different timing requirements on the data, however in such cases we assume that D_m is the shortest such time constraint. We assume that the deadline of each message is less than or equal to its period ($D_m \leq T_m$). Each message m is assumed to be queued by a software task, process or interrupt handler executing on the sending node. This task is either invoked by, or polls for, the event that initiates the message, and takes a bounded amount of time, between 0 and J_m, before the message is in the device driver queue available for transmission. J_m is referred to as the *queuing jitter* of the message and is inherited from the overall response time of the task, including any polling delay.[3] The *transmission deadline* E_m of message m is given by $E_m = D_m - J_m$, and represents the maximum permitted time from the message being queued at the sending node to it being received at other nodes on the bus.

The maximum queuing delay w_m, corresponds to the longest time that message m can remain in the device driver queue or CAN controller transmit buffers, before commencing successful transmission on the bus.

In this paper, we define the *worst-case response time* R_m of a message m as the maximum possible transmission delay from the message being queued until it is received at the receiving nodes.[4] Hence:

$$R_m = w_m + C_m \tag{1}$$

As noted by Broster (2003), receiving nodes can access message m following the end of (message) frame marker and before the 3-bit inter-frame space. The analysis given in the remainder of this paper is therefore slightly pessimistic in that it includes the 3-bit inter-frame space in the computed worst-case response times. To remove this small degree of pessimism, it is valid to simply subtract $3\tau_{bit}$ from the computed response time values, where τ_{bit} is the transmission time for a single bit on the bus.

A message is said to be *schedulable* if its worst-case response time is less than or equal to its *transmission deadline* ($R_m \leq E_m$). A system is said to be schedulable if all of the messages in the system are schedulable.

[3] In the best case, the task could arrive the instant the event occurs and queue the message immediately, whereas in the worst-case, there could be a delay of up to the task's period before it arrives and then a further delay of up to the task's worst-case response time before it queues the message.

[4] Note this is a different way of defining response time to that used by Davis et al. (2007) which includes queuing jitter. To compensate for not including queuing jitter in the response time, in this paper we compare response times with transmission deadlines to determine schedulability.

The following additional notation is used to describe the properties of a set of messages that are transmitted by the same FQ-node and so share a FIFO queue. The FIFO group $M(m)$ is the set of messages that are transmitted by the FQ-node that transmits message m. The lowest priority of any message in the FIFO group $M(m)$ is denoted by L_m. C_m^{MAX} and C_m^{MIN} are the transmission times of the longest and shortest messages in the FIFO group, while C_m^{SUM} is the sum of the transmission times of all of the messages in the group. E_m^{MIN} is the shortest transmission deadline of any message in the group.

We use f_m to denote the maximum *buffering time* from message m being queued until it is able to take part in priority-based arbitration. For a FIFO-queued message f_m equates to the time from the message being entered into the FIFO queue to it becoming the oldest message in that queue. For a priority-queued message $f_m = 0$.

As well as determining message schedulability given a particular priority ordering, we are also interested in effective priority assignment policies.

Definition 1 (Optimal priority assignment policy) A priority assignment policy P is referred to as *optimal* with respect to a schedulability test S and a given network model, if and only if there is no set of messages that are compliant with the model that are deemed schedulable by test S using another priority assignment policy, that are not also deemed schedulable according to test S using policy P.

We note that the above definition is applicable to both sufficient schedulability tests such as those given in Sects. 4 and 5, as well as exact schedulability tests.

A scheduling algorithm is said to be *sustainable* (Baruah and Burns 2006) with respect to a system model, if and only if schedulability of any set of messages compliant with the model implies schedulability of the same set of messages modified by: (i) decreasing transmission times, (ii) increasing periods or inter-arrival times, and (iii) increasing deadlines. Similarly, a schedulability test is referred to as *sustainable* if these changes cannot result in a set of messages that was previously deemed schedulable by the test becoming unschedulable. We note that the modified set of messages may not necessarily be deemed schedulable by the test. A schedulability test is referred to as *self-sustainable* (Baker and Baruah 2009) if such a modified set of messages is always deemed schedulable by the test.

4 Schedulability analysis with priority queues

In this section, we recapitulate the simple sufficient schedulability analysis given by Davis et al. (2007). For networks of PQ-nodes, complying with the scheduling model given in Sect. 3, CAN effectively implements fixed priority non-pre-emptive scheduling. In this case, Davis et al. (2007) showed that an upper bound on the response time R_m of each message m can be found by computing the maximum queuing delay w_m using the following fixed-point iteration:

$$w_m^{n+1} = \max(B_m, C_m) + \sum_{\forall k \in hp(m)} \left\lceil \frac{w_m^n + J_k + \tau_{bit}}{T_k} \right\rceil C_k \tag{2}$$

where τ_{bit} is the transmission time for a single bit, and B_m is the blocking factor described below. Iteration starts with a suitable initial value such as $w_m^0 = C_m$, and continues until either $w_m^{n+1} + C_m > E_m$ in which case the message is not schedulable, or $w_m^{n+1} = w_m^n$ in which case the message is schedulable and its worst-case response time is given by:

$$R_m = w_m^{n+1} + C_m \tag{3}$$

As CAN message transmission is non-pre-emptable, the transmission of a single lower priority message can cause a delay of up to B_m (referred to as *direct blocking*) between message m being queued and the first time that message m could be entered into arbitration on the bus. B_m represents the maximum blocking time due to lower priority messages:

$$B_m = \max_{\forall k \in lp(m)} (C_k) \tag{4}$$

Alternatively, in some cases, the transmission of the previous instance of message m could delay transmission of a higher priority message causing a similar delay (referred to as *push-through blocking*[5]) of up to C_m. Both direct and push-through blocking are accounted for by the 1st term on the RHS of (2). The 2nd term represents *interference* from higher priority messages that can win arbitration over message m and so delay its transmission. Note that once message m starts successful transmission it cannot be pre-empted, so the message's overall response time is simply the queuing delay plus its transmission time (given by (3)).

Using (2) and (3), engineers can determine upper bounds[6] on worst-case response times and hence the schedulability of all messages on a network comprising solely PQ-nodes. Although the analysis embodied in (2) and (3) is pseudo-polynomial in complexity in practice it is tractable on a desktop PC for complex systems with hundreds of messages. (A number of techniques are also available for increasing the efficiency of such fixed point iterations (Davis et al. 2008)).

5 Schedulability analysis with FIFO queues

In this section, we derive sufficient schedulability analysis for messages on networks with both PQ-nodes and FQ-nodes. The analysis we introduce is *FIFO-symmetric*, by this we mean that the same worst-case response time is attributed to all of the messages in a FIFO group. We note that FIFO-symmetric analysis incurs some pessimism in terms of the worst-case response time attributed to the higher priority messages in a FIFO group; however, in practice this pessimism is likely to be small. This is because the order in which messages are placed in a FIFO queue is undefined, and so in the worst case, the highest priority message in a FIFO group has to wait for an instance of each lower priority message in the group to be transmitted.

[5] See Davis et al. (2007) for an explanation of why push-through blocking is important.

[6] Equation (2) is sufficient rather than exact due to the fact that push through blocking may not necessarily be possible.

5.1 Priority-queued messages

We now derive an upper bound on the worst-case queuing delay for a priority-queued message m, in a system with both PQ-nodes and FQ-nodes.

In the case of systems with only PQ-nodes, Davis et al. (2007) showed that the worst-case queuing delay for a priority-queued message m occurs for an instance of that message queued at the beginning of a priority level-m busy period[7] that starts immediately after the longest lower priority message begins transmission. Further, this maximal busy period begins with a so-called *critical instant* where message m is queued simultaneously with all higher priority messages and then each of these higher priority messages is subsequently queued again after the shortest possible time interval. Equation (2) provides a sufficient upper bound on this worst-case queuing delay.

The analysis embodied in (2) assumes that higher priority messages are able to compete for access to the bus (i.e. enter bus arbitration) as soon as they are queued; however, this assumption does not hold for FIFO-queued messages. Instead a FIFO-queued message k may have to wait for up to a maximum time f_k before it becomes the oldest message in its FIFO queue, and can enter priority-based arbitration. A FIFO-queued message k can therefore be thought of as becoming priority queued after an additional delay of f_k. Stated otherwise, in terms of its interference on lower priority messages, a FIFO-queued message k can be viewed as if it were a priority-queued message with its jitter increased by f_k. (Note, we will return to how f_k is calculated for FIFO-queued messages later.) An upper bound on the queuing delay for a priority-queued message m can therefore be calculated via the fixed point iteration given by (5).

$$w_m^{n+1} = \max(B_m, C_m) + \sum_{\forall k \in hp(m)} \left\lceil \frac{w_m^n + J_k + f_k + \tau_{bit}}{T_k} \right\rceil C_k \tag{5}$$

As with (2), iteration starts with a suitable initial value such as $w_m^0 = C_m$, and continues until either $w_m^{n+1} + C_m > E_m$ in which case the message is not schedulable, or $w_m^{n+1} = w_m^n$ in which case its response time is given by:

$$R_m = w_m^{n+1} + C_m \tag{6}$$

Note that the queuing delay and response time are only valid with respect to the values of f_k used. We return to this point later.

5.2 FIFO-queued messages

We now derive an upper bound on the worst-case queuing delay for a FIFO-queued message m, in a system with both PQ-nodes and FQ-nodes.

As our analysis is *FIFO-symmetric*, we will attribute the same upper bound response time to all of the messages sent by the same FQ-node. Our analysis derives this

[7] A priority level-m busy period is a contiguous interval of time during which there is always at least one message of priority m that has not yet completed transmission.

sufficient response time by considering an arbitrary message from the FIFO group $M(m)$. For the sake of simplicity, we will still refer to this message as message m; however our analysis will be independent of the exact choice of message from the FIFO group. At each stage in our analysis we will make worst-case assumptions, ensuring that the derived response time is a correct upper bound. For example, we will frame our calculation of the queuing delay w_m by assuming the lowest priority L_m of any message in the FIFO group.

As every message j in $M(m)$ has $D_j \leq T_j$ then in a schedulable system, when any arbitrary message from $M(m)$ is queued, there can be at most one instance of each of the other messages in $M(m)$ ahead of it in the FIFO queue. The maximum transmission time of these messages, and hence the maximum interference on an arbitrary message m, due to messages sent by the same FQ-node, is therefore upper bounded by:

$$C_m^{SUM} - C_m^{MIN} \tag{7}$$

Indirect blocking could also occur due to the non-pre-emptive transmission of a previous instance of any one of the messages in $M(m)$. This indirect blocking is upper bounded by C_m^{MAX}. As an alternative, direct blocking could occur due to transmission of any of the messages of lower priority than L_m sent by other nodes. Finally, in terms of interference from higher priority messages sent by other FQ-nodes and PQ-nodes, the argument about increased jitter made in the previous section applies, and so the interference term from (5) can again be used.

Considering all of the above, an upper bound on the queuing delay for an arbitrary message m belonging to the FIFO group $M(m)$ is given by the solution to the following fixed point iteration:

$$w_m^{n+1} = \max\left(B_{L_m}, C_m^{MAX}\right) + \left(C_m^{SUM} - C_m^{MIN}\right) + \sum_{\forall k \in hp(L_m) \wedge k \notin M(m)} \left\lceil \frac{w_m^n + J_k + f_k + \tau_{bit}}{T_k} \right\rceil C_k \tag{8}$$

Iteration starts with a value of $w_m^0 = \max(B_{L_m}, C_m^{MAX}) + (C_m^{SUM} - C_m^{MIN})$ and continues until either $w_m^{n+1} + C_m^{MIN} > E_m^{MIN}$ in which case the set of messages $M(m)$ is declared unschedulable, or $w_m^{n+1} = w_m^n$ in which case all of the messages in $M(m)$ are deemed to have a response time of:

$$R_m = w_m^{n+1} + C_m^{MIN} \tag{9}$$

Equations (8) and (9) make the worst-case assumption that interference from higher priority messages can occur up to a time C_m^{MIN} before transmission of message m completes. We note that this is a pessimistic assumption with respect to those messages belonging to the FIFO group that have transmission times[8] longer than C_m^{MIN}.

[8] In practice all messages sent on CAN often have the maximum length (8 data bytes) so as to minimise the relative overheads of the other fields in the message (ID, CRC etc.). In this case, no additional pessimism is introduced by this assumption.

5.3 Schedulability test with arbitrary priorities

We now derive a schedulability test from (5) and (6) and (8) and (9). The basic idea is to avoid having to consider the potentially complex interactions between the FIFO queues of different nodes. This is achieved by abstracting the FIFO behaviour of messages sent by other nodes as simply additional jitter f_k before each message k can enter priority based arbitration on the bus. When calculating the response time of a given message, we therefore need only consider the behaviour of the node that sends that message (PQ-node or FQ-node) and the buffering delays of messages sent by other nodes.[9]

An upper bound on the buffering time f_m of a FIFO-queued message m is:

$$f_m = R_m - C_m^{MIN} \tag{10}$$

When the priorities of messages in different FIFO groups are interleaved, this leads to a circular dependency in the response time calculations. For example, let m and k be the priorities of messages in two different FIFO groups with interleaved priorities (i.e. $k \in hp(L_m)$ and $m \in hp(L_k)$). The response time R_k of message k, and hence its buffering time f_k, depend on the buffering time f_m of message m as $m \in hp(L_k)$; however, the buffering time f_m of message m depends on its response time R_m which in turn depends on f_k as $k \in hp(L_m)$. This apparent problem can be solved by noting that the response times calculated via (5) and (6) and (8) and (9) are monotonically non-decreasing with respect to the buffering times, and that the buffering times given by (10) are monotonically non-decreasing with respect to the response times calculated via (6) and (9). Hence by using an outer loop iteration, and repeating response time calculations until the buffering times no longer increase, we can compute correct upper bound response times and hence schedulability for all messages, as shown in Algorithm 1. (Note, to speed up the schedulability test, for each message m, the value of w_m computed on one iteration of the while loop (lines 3 to 23) can be used as an initial value on the next iteration.)

Algorithm 1 provides a sufficient schedulability test for FIFO-queued and priority-queued messages in any arbitrary priority ordering.

5.4 Partial priority ordering within a FIFO group

In this section, we consider an appropriate priority ordering for messages within a FIFO group.

Definition 2 A *FIFO-adjacent priority ordering* is any priority ordering whereby all of the messages sharing a FIFO queue are assigned adjacent priorities.

Theorem 1 *If a priority ordering Q exists that is schedulable according to the FIFO-symmetric schedulability analysis of Algorithm 1 then a schedulable FIFO-adjacent priority ordering P also exists.*

[9] If the message belongs to a PQ-node, then the other messages sent by the same node have buffering delays of zero, if it belongs to an FQ-node, then the buffering delays for other messages sent by the same node are not needed in the calculations (8) and (9).

Algorithm 1 FIFO symmetric schedulability test

```
1  repeat = true
2  initialise all f_k = 0
3  while(repeat){
4     repeat = false
5     for each priority m, highest first{
6        if (m is FIFO-queued){
7           calc R_m according to Eqs. (8) and (9)
8           if (R_m > E_m^MIN) {
9              return unschedulable
10          }
11          if (f_m < w_m) {
12             f_m = w_m
13             repeat = true;
14          }
15       }
16       else {
17          calc R_m according to Eqs. (5) and (6)
18          if (R_m > E_m) {
19             return unschedulable
20          }
21       }
22    }
23 }
24 return schedulable
```

Proof Let m be a FIFO-queued message that is not the lowest priority message in its FIFO group. Now consider a priority transformation whereby message m is shifted down in priority so that it is at a priority level immediately above that of the lowest priority message in its FIFO group. We will refer to the old priority ordering as Q and the new priority ordering as Q'.

We observe from (5) and (8), that given the same fixed set of buffering times f_k, then (i) the response time computed for message m is the same for both priority orderings, and (ii) the response times computed for all other messages are no larger in priority ordering Q' than they are in priority ordering Q. Due to the mutual monotonically non-decreasing relationship between message buffering times and response times, and the fact that Algorithm 1 starts with all the buffering times set to zero, this means that on every iteration of Algorithm 1, the response times and buffering times computed for each message under priority ordering Q' are no larger than those computed on the same iteration for priority ordering Q. Hence if priority ordering Q is schedulable, then so is priority ordering Q'.

Applying the priority transformation described above to every FIFO-queued message that is not the lowest priority message in its FIFO group transforms any schedulable priority ordering Q into a FIFO-adjacent priority ordering P, without any loss of schedulability □

Theorem 1 tells us that regardless of the priority assignment applied to priority-queued messages, we should ensure that all of the messages that share a single FIFO queue have adjacent priorities. In terms of CAN message IDs we note that this does not require that consecutive values are used for the IDs, only that there is no interleaving with respect to the priorities of other messages. In practice message IDs can be chosen to meet these requirements, while also providing appropriate bit patterns for message filtering.

5.5 Schedulability test for FIFO-adjacent priorities

In this section, we derive an improved schedulability test that is valid for FIFO-adjacent priority orderings.

Recall that Davis et al. (2007) showed that the worst-case queuing delay for a priority-queued message m occurs within the priority level-m busy period that starts with a *critical instant*. Provided that a FIFO-adjacent priority ordering is used, then the same situation also represents the worst-case scenario when higher priority messages are sent by either PQ-nodes or FQ-nodes. This can be seen by considering the interference on a priority-queued message m from a higher priority FIFO-queued message k. As message k is of higher priority than message m, then so are *all* of the other messages in the same FIFO group (i.e. $M(k)$). Thus any message in $M(k)$ that is queued prior to the start of transmission of message m will be sent on the bus before message m, irrespective of the order in which the messages in $M(k)$ are placed in the FIFO queue. In effect all of the additional jitter on message k is already accounted for by interference on message m from other messages in the same FIFO group ($M(k)$). In this case, there is no additional jitter on message k caused by messages of lower priority than m. Hence for each FIFO message k, we can set $f_k = 0$, and use (5) and (6) to calculate the queuing delay and worst-case response time of each message m. The same argument applies when we consider the schedulability of a FIFO-queued message m. In this case we can use (8) and (9) to calculate the queuing delay and worst-case response time, with all buffering times $f_k = 0$. Further, as the buffering times are all fixed at zero, a single pass over the priority levels is all that is needed to determine schedulability. In other words, lines 11–14 of Algorithm 1 can be omitted when considering FIFO-adjacent priority orderings. This revised schedulability test therefore dominates the test given in Sect. 5.3 (i.e. Algorithm 1 with lines 11–14 present).

The simplified analysis given in this section is similar to that provided for FP/FIFO scheduling of flows by Martin et al. (2007) and for OSEK/VDX tasks by Bimbard and George (2006) and Hladik et al. (2007).

5.6 Sufficiency and sustainability of the FIFO-symmetric schedulability tests

In this section, we prove that treating all of the messages in a FIFO queue as having the lowest priority L_m of any message in that queue, leads to a worst-case response time that is no smaller than the actual worst-case response time of each message. Thus, we show that the FIFO-symmetric schedulability test given in Sect. 5.2, by (8) and (9) (i.e. Algorithm 1), is sufficient in the case of a general priority ordering (Theorem 2), and also in the case of a FIFO-adjacent priority ordering when the buffering

delays are set to zero (Theorem 3). We also show that the FIFO-symmetric schedulability test is self-*sustainable* (Baker and Baruah 2009) in these two cases (Theorems 4 and 5).

Recall from Sect. 3 that the property of *self-sustainability* implies *sustainability* of the schedulability test. Sustainability is an important property as it means that any set of messages that are deemed schedulable by the test remain schedulable if their transmission times are reduced, for example by bit-stuffing which is less than the worst-case assumed by the analysis, or their periods or deadlines are increased.

Lemma 1 *Consider a system G comprising a set of nodes connected via a CAN bus, with a static set of hard real-time messages sent on the bus. We assume that the node transmitting message m is an FQ-node, which transmits a FIFO-group of messages $M(m)$, and that messages from all other nodes are priority queued. Further, the priorities of the messages in the FIFO-group $M(m)$ are arbitrary, with a lowest priority of L_m. Let H be a system that is identical to system G, with the exception that all of the messages in the FIFO-group $M(m)$ have priority L_m. The worst-case response time of each message in $M(m)$ in system G is no greater than the worst-case response time of the equivalent message under system H.*

Proof We prove a stronger hypothesis: that for any valid sequence of message releases, the response time of every instance of every message in $M(m)$ is no greater in system G than it is in system H.

We observe that for any valid sequence of message releases, the duration of each priority level L_m busy period (during which there are ready messages of priority L_m or higher) is the same in both systems. This is the case because fixed priority non-preemptive scheduling is work-conserving, message release times are the same in both systems, and the only difference between them is the priority ordering of messages with priorities no lower than L_m. As a consequence, the times at which messages with priorities lower than L_m start to be transmitted are the same in both systems. Note the order in which messages of priority L_m and higher are sent in a priority level L_m busy period may be different in the two systems.

We prove the hypothesis by contradiction: For some arbitrary sequence of message releases, let x be the first instance of a message in $M(m)$ with a longer response time in system G than it has in system H. To compare the response times of message instance x in the two systems, we need only consider the priority level L_m busy period that contains transmission of x. Let t be the start of this busy period. The time s at which x starts to be transmitted in system H is given by:

$$s = t + B + CP(t) + \sum_{\forall k \in hp(L_m) \wedge k \notin M(m)} I_k(t) \tag{11}$$

where B is blocking due to a lower priority message (if any) that starts transmission at the start of the busy period, $CP(t)$ is the total transmission time for instances of messages in $M(m)$ released during the busy period prior to the release of x, and $I_k(t)$ is the total transmission time of instances of higher priority message k released during the busy period, prior to time s.

As (11) holds for system H, it must be the case in system G that x can start transmission no later than s, as its priority is no lower than L_m. This contradicts the

hypothesis that x is the first instance of a message in $M(m)$ with a longer response time in system G than it has in system H. Hence there can be no such instance x. □

Theorem 2 *The FIFO-symmetric schedulability test given in Sect.* 5.2, (8) *and* (9), *is sufficient.*

Proof Lemma 1 shows that the worst-case response times for a set of FIFO-queued messages $M(m)$ with arbitrary priorities, the lowest of which is L_m, are upper bounded by the worst-case response times of those same messages computed for a system that is equivalent except for the fact that all of the messages in $M(m)$ have priority L_m. Sufficient values for the worst-case response times of FIFO-queued messages may therefore be calculated according to these assumptions as described in Sect. 5.2. Note that the assumption that all messages sent by other nodes are priority queued is dealt with by (8) via modelling each message k sent by another FQ-node as a priority queued message with release jitter increased by the buffering delay f_k. □

Next we show that in the case of a FIFO-adjacent priority ordering, the FIFO-symmetric schedulability test given by Algorithm 1 is sufficient with the buffering delays set to zero. Further, we show that in systems where not all of the priorities of messages in FIFO-groups are adjacent, then some specific buffering delays can still be assumed to be zero, improving the precision of the analysis.

We use the concepts of *spanning* and *partitioning* to describe how the priorities of messages in different FIFO groups are interleaved. We say that a FIFO-group $M(m)$ *spans* a priority level j occupied by a priority queued message if there is at least one message in the group with a priority higher than j and at least one message in the group with a priority lower than j. Similarly, a FIFO-group $M(k)$ spans another FIFO group $M(j)$ if there is a message in $M(k)$ with a priority higher than L_j (the lowest priority of a message in $M(j)$) and another message in $M(k)$ with a priority lower than L_j. If no FIFO-groups span priority level j, then we say that priority level j, partitions the FIFO-groups. In this case, all messages in the same FIFO-group either have priorities that are higher than j or lower than j. Similarly, a FIFO-group $M(j)$ is said to partition the other FIFO groups if the lowest priority level L_j of the FIFO-group partitions the other FIFO-groups. In a FIFO-adjacent priority ordering each FIFO-group partitions all of the other FIFO-groups, and no FIFO group spans another FIFO-group or a priority queued message.

Lemma 2 *Let j be a priority queued message and $M(k)$ a FIFO group where all of the messages in $M(k)$ have higher priorities than j, i.e. $j \in lp(L_k)$. The worst-case response time of message j can be computed according to* (5) *and* (6) *with the interference from messages in $M(k)$ calculated with their buffering delays assumed to be zero.*

Proof As $j \in lp(L_k)$, then all of the messages in FIFO-group $M(k)$ have a higher priority than message j. Thus all of the ready messages in FIFO-group $M(k)$ must be sent prior to the start of transmission of message j. It follows that the worst-case interference from messages in $M(k)$ occurs when all of those messages are queued

simultaneously at the start of a priority level-j busy period and are queued again as soon as possible. Further, at any given time there can be no messages in $M(k)$ of priority higher than j that are ready but waiting in the FIFO queue behind a message of priority j or lower. Therefore the worst-case scenario for interference from messages in higher priority FIFO-group $M(k)$ is the same as the priority-queued case. (Note that buffering delays can still occur, but their only effect is to re-order the transmission of messages in $M(k)$, without changing the total interference on the message at priority level j.) □

Lemma 3 *Let $M(j)$ be a FIFO-group and $M(k)$ another FIFO-group such that all of the messages in $M(k)$ have higher priorities than the lowest priority message in $M(j)$, i.e. $L_j \in lp(L_k)$. The worst-case response time for messages in $M(j)$ can be computed according to* (8) *and* (9) *with the interference from messages in $M(k)$ calculated with their buffering delays assumed to be zero.*

Proof (Follows the logic of the proof of Lemma 2) Lemma 1 tells us that we can upper bound the worst-case response times of messages in $M(j)$ by assuming that they are all transmitted at priority L_j. As $L_j \in lp(L_k)$, then all of the messages in FIFO-group $M(k)$ have a higher priority than L_j. Thus all of the ready messages in FIFO-group $M(k)$ must be sent prior to the start of transmission of any message at priority L_j. It follows that the worst-case interference from messages in $M(k)$ occurs when all of those messages are queued simultaneously at the start of a priority level-L_j busy period and are queued again as soon as possible. Further, at any given time there can be no messages in $M(k)$ of priority higher than L_j that are ready but waiting in the FIFO queue behind a message of priority L_j or lower. Therefore the worst-case scenario for interference from messages in FIFO-group $M(k)$ is the same as the priority-queued case. (Note that buffering delays can still occur, but their only effect is to re-order the transmission of messages in $M(k)$, without changing the total interference on messages at priority level L_j.) □

Theorem 3 *For any priority queued message* (*or FIFO-group*) *that partitions the other FIFO groups, the worst-case response time can be computed according to* (5) *and* (6) (*or* (8) *and* (9)) *with the buffering delays of messages in higher priority FIFO-groups assumed to be zero. Further, for any FIFO-group $M(j)$ that does not partition the other FIFO-groups, then the worst-case response time of messages in $M(j)$ can be computed with non-zero buffering delays used only for those messages in FIFO-groups that span priority level L_j. Similarly, for a priority queued message j that does not partition the FIFO-groups, then the worst-case response time of message j can be computed using* (5) *and* (6) *with non-zero buffering delays used only for those messages in FIFO groups that span priority level j.*

Proof Follows from the proofs of Lemmas 1, 2, and 3, and analysis of the priority queued case. □

Corollary 1 *With a FIFO-adjacent priority ordering the FIFO-symmetric schedulability test given by Algorithm 1 is sufficient with all buffering delays assumed to be zero.* (*This corollary follows from Theorem 3.*)

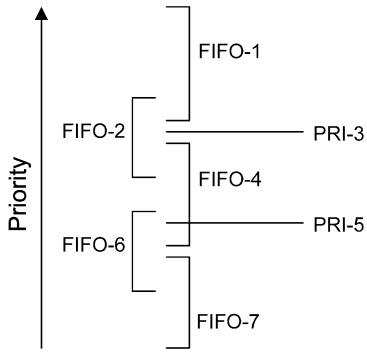

Fig. 1 FIFO groups spanning priorities

For sets of messages that are not all in FIFO-adjacent priority order, then the schedulability test given by Algorithm 1 must be used with lines 11–14 present. This is because at least one priority queued message or FIFO-group will need to have its worst-case response time computed using non-zero buffering delays, and hence repeated calculation is required to account for the circular dependency that this may imply.

Theorem 3 allows for more precise analysis of worst-case response times when there are constraints on message priorities that prevent the use of a FIFO-adjacent priority ordering. This is illustrated by Fig. 1 which shows, as brackets, the maximum and minimum priority of messages in the various FIFO-groups, as well as the priorities of two priority queued messages. Theorem 3 tells us that when computing the worst-case response time for messages in FIFO-4, then we may assume a buffering delay of zero for messages in FIFO-2 and FIFO-1, because all of the messages in those FIFO-groups have higher priorities than the lowest priority message in FIFO-4. However, we must compute and use non-zero buffering delays for messages in FIFO-6 as this group spans the lowest priority level of group FIFO-4. Further, when computing the worst-case response time for messages in FIFO-6, we can assume the buffering delays for messages in FIFO-1, FIFO-2 and FIFO-4 are zero, but must use the computed non-zero values for the buffering delays for messages in FIFO-7. When computing the worst-case response time of priority queued message PRI-5, we can assume the buffering delays for messages in FIFO-1 and FIFO-2 are zero, but must use the computed non-zero buffering delays for messages in FIFO-4 and FIFO-6. Finally, when computing the worst-case response time for messages in FIFO-7, we may assume a buffering delay of zero for all messages in FIFO-1, FIFO-2, FIFO-4, and FIFO-6.

Note that whether we can assume buffering delays of zero or not for messages in a particular FIFO-group (e.g. FIFO-6) depends on which messages we are computing the worst-case response time for (e.g. FIFO-4, PRI-5, and FIFO-7).

Theorem 4 *With a FIFO-adjacent priority ordering, the FIFO-symmetric schedulability test of Sect.* 5.5 (*Algorithm* 1 *with lines* 11–14 *omitted*) *is self-sustainable* (*Baker and Baruah* 2009). *Meaning that any set of messages deemed schedulable by the test will also be deemed schedulable by the test if those messages are modified by*:

(i) *decreasing transmission times*, (ii) *increasing periods or inter-arrival times*, *and* (iii) *increasing deadlines*.

Proof The proof is in three parts:

(i) Decreasing transmission times. Proof of this aspect of self-sustainability follows from the fact that the queuing delay w_m of message m, given by (5) and (8), is non-increasing with respect to any decrease in the transmission time of message m or any other messages. Note that in (8), the transmission time C_m^{MIN} of the shortest message in the FIFO queue cannot become smaller by an amount x without at least an equivalent reduction in the sum of the message transmission times C_m^{SUM}, hence the value of $C_m^{SUM} - C_m^{MIN}$ cannot increase. From (6) and (9), it follows that the message response time R_m is also non-increasing with respect to a decrease in message transmission times, and so the test is self-sustainable with respect to reductions in message transmission times.

(ii) Increasing periods or inter-arrival times. As message periods appear only in the denominator of the ceiling functions in (5) and (8), increases in these values cannot result in an increase in message queuing delays or response times. Hence the test is self-sustainable with respect to increases in message periods.

(iii) Increasing deadlines. As increases in message deadlines can only increase the transmission deadlines E_m and E_m^{MIN}, such increases cannot result in a message that was previously schedulable according to the test becoming unschedulable according to the test. Hence the test is self-sustainable with respect to increases in message deadlines. □

Theorem 5 *The FIFO-symmetric schedulability test of Sect.* 5.5 (*Algorithm* 1 *with lines* 11–14 *present*) *is self-sustainable* (*Baker and Baruah* 2009) *in the general case with an arbitrary priority ordering*.

Proof Follows from the proof of Theorem 4, noting that the buffering delay f_m is equal to the queuing delay w_m (see (9) and (10)), and so f_m is also non-increasing with respect to any decrease in message transmission times, or increase in message periods. □

Note that the proof of Theorem 5 does not require any changes to the priority order of the messages. They will still remain schedulable according to the FIFO-symmetric schedulability test with the original priority ordering.

5.7 Buffer sizes

Assuming that all messages have constrained deadlines, and that the network is schedulable, then irrespective of the queuing policy, two instances of the same message cannot be present in the queue at the same time; otherwise the first instance would have missed its deadline. The worst-case buffer usage is therefore equal to the number of messages that use that queue, and this occurs when all of the messages are queued at the same time.

6 Priority assignment policies

The schedulability test presented in Sect. 5.5 is applicable irrespective of the overall priority ordering, provided that messages sharing the same FIFO queue are assigned adjacent priorities. Choosing an appropriate priority ordering among the priority-queued messages and the FIFO groups is however an important aspect of achieving overall schedulability and hence effective real-time performance.

In this section, we consider the assignment of messages to *priority bands*, where a priority band comprises either a single priority level containing one priority-queued message, or a number of adjacent priority levels containing a FIFO group of messages. We derive priority assignment policies that are optimal with respect to the schedulability analysis given in Sect. 5.5.

6.1 Optimal priority assignment

Davis et al. (2007), showed that, assuming solely priority queuing, Audsley's Optimal Priority Assignment (OPA) algorithm (Audsley 1991, 2001) provides the optimal priority assignment for CAN messages. We now show that with an appropriate modification to handle FIFO groups, Audsley's algorithm is also optimal with respect to the schedulability test given in Sect. 5.5. The pseudo code for this OPA-FP/FIFO algorithm is given in Algorithm 2. Note that only one message from each FIFO group is considered in the initial list, as once this message is assigned to a priority band, then so are the other messages in the same FIFO group.

Davis and Burns (2009b, 2011) showed that Audsley's OPA algorithm is optimal with respect to any schedulability test that meets three specific conditions. According

Algorithm 2 Optimal priority assignment (OPA-FP/FIFO)

```
for each priority band k, lowest first
{
  for each message msg in the initial list {
    if msg is schedulable in priority band k according to schedulability
    test S with all unassigned priority-queued messages/other FIFO
    groups assumed to be in higher priority bands {
      assign msg to priority band k
      if msg is part of a FIFO group {
        assign all other messages in the FIFO group to adjacent
        priorities within priority band k
      }
      break (continue outer loop)
    }
  }
  return unschedulable
}
return schedulable
```

to Theorem 1, we need only consider the priority bands assigned to each priority-queued message, and each FIFO group (as all messages in a FIFO group have adjacent priorities in an optimal priority ordering). We therefore re-state these three conditions in the context of priority-queued messages and FIFO groups.

The three conditions refer to properties or attributes of the messages. Message properties are referred to as *independent* if they have no dependency on the priority assigned to the message. For example the longest transmission time, deadline, and minimum inter-arrival time of a message are all independent properties, while the worst-case response time typically depends on the message's priority and so is a *dependent* property.

Condition 1 The schedulability of a message/FIFO group identified by m, may, according to test S, depend on any independent properties of other messages/FIFO groups in higher priority bands than m, but not on any properties of those messages/FIFO groups that depend on their relative priority ordering.

Condition 2 The schedulability of a message/FIFO group identified by m may, according to test S, depend on any independent properties of the messages/FIFO groups in lower priority bands than m, but not on any properties of those messages/FIFO groups that depend on their relative priority ordering.

Condition 3 When the priorities of any two adjacent priority bands are swapped, then the message/FIFO group being assigned the higher priority band cannot become unschedulable according to test S, if it was previously schedulable in the lower priority band. (As a corollary, the message/FIFO group being assigned the lower priority band cannot become schedulable according to test S, if it was previously unschedulable in the higher priority band.)

Theorem 6 *The OPA-FP/FIFO algorithm is an optimal priority assignment algorithm with respect to the FIFO-symmetric schedulability test of Sect.* 5.5 (*Algorithm* 1 *with lines* 11–14 *omitted*).

Proof It suffices to show that conditions 1–3 hold with respect to the schedulability test given by Algorithm 1 with lines 11–14 omitted.

Condition 1: Inspection of (5) & (6) and (8) & (9), assuming all f_k are fixed at zero, shows that the response time of each message m is dependent on the set of messages in higher priority bands, but not on their relative priority ordering.

Condition 2: Inspection of (5) & (6) and (8) & (9), shows that the response time of each message m is dependent on the set of messages in lower priority bands via the direct blocking term, but not on their relative priority ordering.

Condition 3: Inspection of (5) & (6) and (8) & (9), assuming all f_k are fixed at zero, shows that increasing the priority band of message m cannot result in a longer response time. This is because although the direct blocking term can get larger with increasing priority this is always counteracted by a decrease in interference that is at least as large; hence the length of the queuing delay cannot increase with increasing priority, and so neither can the response time. □

For N priority-queued messages/FIFO groups, the OPA-FP/FIFO algorithm performs at most $N(N-1)/2$ schedulability tests and is guaranteed to find a schedulable priority assignment if one exists. It does not however specify an order in which messages should be tried in each priority band. This order heavily influences the priority assignment chosen if there is more than one ordering that is schedulable. In fact, a poor choice of initial ordering can result in a priority assignment that leaves the system only just schedulable. We suggest that, as a useful heuristic, priority-queued messages and FIFO groups are tried at each priority level in order of transmission deadline (i.e. E_m or E_m^{MIN}), largest value first. This will result in a priority ordering reflecting transmission deadlines if such an ordering is schedulable. Alternatively, approaches which result in a robust priority assignment can be developed from the techniques described by Davis and Burns (2009a).

6.2 TDMO-FP/FIFO priority assignment

In industrial practice, CAN configurations are sometimes designed such that all of the messages are of the same maximum length (8 data bytes). This is done to ameliorate the effects of the large overhead of the other fields (arbitration, CRC etc.) in each message.

Definition 3 Transmission deadline monotonic priority ordering for FP/FIFO (TDMPO-FP/FIFO) is a priority assignment policy that assigns priority bands to priority queued messages and FIFO groups according to their transmission deadlines; with a shorter transmission deadline implying a higher priority. (Recall that the transmission deadline of a FIFO group is given by the shortest transmission deadline of any message in that group.)

Figure 2 illustrates the TDMPO-FP/FIFO priority assignment policy. In this section, we show that the TDMPO-FP/FIFO priority assignment policy is optimal, with

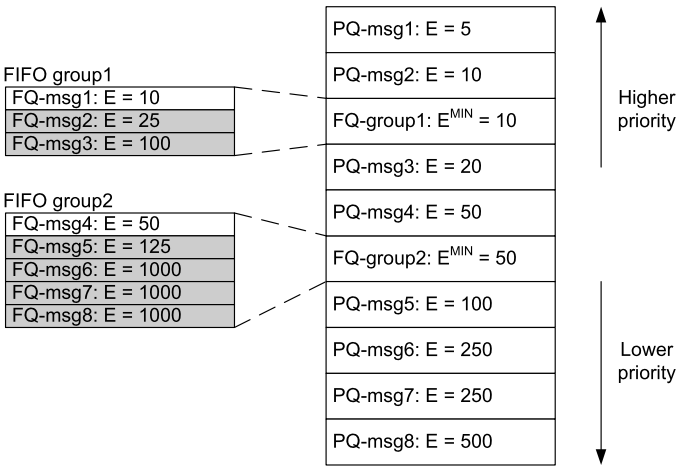

Fig. 2 TDMPO-FP/FIFO priority ordering

respect to the sufficient schedulability test given in Sect. 5.5 (i.e. Algorithm 1 with lines 11–14 omitted) when all messages have the same worst-case transmission time (C).

Corollary 2 *For networks where all of the message transmission times are the same, then the blocking factor, used in both the sufficient schedulability test given by Davis et al. (2007) (recapitulated in Sect. 4) and the sufficient schedulability tests given in Sect. 5 of this paper, is the same for every message, and is equal to the worst-case message transmission time* (C).

Lemma 4 *Let i and j be the indices of two adjacent priority bands in a priority ordering that is schedulable according to the sufficient schedulability test given in Sect. 5.5 (i.e. Algorithm 1 with lines 11–14 omitted). Assume that i is of higher priority than j, and that the transmission deadline E_X of the priority-queued message/FIFO group (X) initially in priority band i is longer than the transmission deadline E_Y of priority-queued message/FIFO group (Y) initially in priority band j. If the priorities of X and Y are swapped, so that X is in the lower priority band j, and Y is in the higher priority band i, then X remains schedulable, provided that the set of messages all have the same worst-case transmission time* (C).

Proof Let $R_{Y,j}$ be the response time of Y in priority band j (with X in the higher priority band i). Similarly, let $R_{X,j}$ be the response time of X in priority band j, (with Y in the higher priority band i). As Y is schedulable when it is in the lower priority band, then, $R_{Y,j} \leq E_Y$, thus as $E_Y < E_X$, it follows that to prove the lemma, we need only show that $R_{X,j} \leq R_{Y,j}$. Further, as all messages have the same worst-case transmission time (C), and so the response times are equal to the queuing delays plus C, we need only compare the two queuing delays, referred to for convenience as $w_{X,j}$ and $w_{Y,j}$. Below we give formulae for $w_{X,j}$ and $w_{Y,j}$ based on (5) & (6) and (8) & (9). We have separated out the interference terms for X and Y. Further, we use $B(j)$ to represent the blocking factor, and $I(i, w)$ to represent the interference from messages in higher priority bands.

$$B(j) = \max(B_j, C) = C$$
$$I(i, w) = \sum_{\forall k \in hp(i)} \left\lceil \frac{w + J_k + \tau_{bit}}{T_k} \right\rceil C$$

(i) Queuing delay $w_{X,j}$ (simplified by cancelling out the blocking factor C and the $-C$ from $(C_X^{SUM} - C)$) is given by:

$$w_{X,j}^{n+1} = C_X^{SUM} + \sum_{\forall k \in Y} \left\lceil \frac{w_{X,j}^n + J_k + \tau_{bit}}{T_k} \right\rceil C + I\left(i, w_{X,j}^n\right) \tag{12}$$

Note, in (12), if X is a priority-queued message, then $C_X^{SUM} = C$, also, if Y is a priority-queued message, then there is only one message $k \in Y$ present in the summation term; similarly for (13) below.

(ii) Queuing delay $w_{Y,j}$:

$$w_{Y,j}^{n+1} = C_Y^{SUM} + \sum_{\forall k \in X} \left\lceil \frac{w_{Y,j}^n + J_k + \tau_{bit}}{T_k} \right\rceil C + I\left(i, w_{Y,j}^n\right) \tag{13}$$

We can simplify (13) by noting that as Y is schedulable according to the assumption given in the lemma, then it must be the case that:

$$w_{Y,j} + C = R_{Y,j} \leq E_Y < E_X = \min_{\forall k \in X}(D_k - J_k) \leq \min_{\forall k \in X}(T_k - J_k)$$

As $C > \tau_{bit}$, we have $\forall k \in X : w_{Y,j} + J_k + \tau_{bit} < T_k$, and so the ceiling function in (13) evaluates to one in each case; indicating that only one instance of each message in X can contribute to the interference term. Hence (13) simplifies to:

$$w_{Y,j}^{n+1} = C_Y^{SUM} + C_X^{SUM} + I\left(i, w_{Y,j}^n\right) \tag{14}$$

Now let us consider a simplification of (12) that is valid for values of $w_{X,j} \leq E_Y - C$.

As $E_Y = \min_{\forall k \in Y}(D_k - J_k) \leq \min_{\forall k \in Y}(T_k - J_k)$ and $C > \tau_{bit}$ then we have $\forall k \in Y : w_{X,j} + J_k + \tau_{bit} < T_k$ and so the ceiling function in (12) evaluates to one in each case; indicating that only one instance of each message in Y can contribute to the interference term. Hence, provided that $w_{X,j} \leq E_Y - C$, then (12) reduces to:

$$w_{X,j}^{n+1} = C_X^{SUM} + C_Y^{SUM} + I\left(i, w_{X,j}^n\right) \tag{15}$$

Equations (14) and (15) are equivalent. As we know that (14) converges on a value $w_{Y,j} = R_{Y,j} - C \leq E_Y - C$, then (15) and hence (12) must also converge on the same value, thus $w_{X,j} = w_{Y,j}$, and $R_{X,j} = R_{Y,j}$. □

Theorem 7 *TDMPO-FP/FIFO is an optimal policy for assigning priority-queued messages and FIFO groups to priority bands, with respect to the sufficient schedulability test given in Sect.* 5.5 (*Algorithm* 1 *with lines* 11–14 *omitted*), *provided that all messages have the same worst-case transmission time.*

Proof We prove the theorem by showing that any ordering Q of priority bands that is schedulable according to the sufficient schedulability test given in Sect. 5.5 can be transformed into a TDMPO-FP/FIFO priority ordering without any loss of schedulability.

Let i and j be the indices of two adjacent priority bands in an ordering that is schedulable according to the sufficient schedulability test given in Sect. 5.5. Assume that i is of higher priority than j, and that the transmission deadline E_X of the priority-queued message/FIFO group (X) in priority band i is longer than the transmission deadline E_Y of the priority-queued message/FIFO group (Y) in priority band j.

We now consider what happens to the schedulability of all of the messages in the system when we swap the priorities of X and Y (i.e. when we place X in the lower priority band j, and Y in the higher priority band i) to create priority ordering Q'. There are four cases to consider:

1. *Priority bands with higher priority than i* $(h \in hp(i))$: Inspection of (5) & (6) and (8) & (9) shows that the response times of each of the messages in these bands is

the same in priority ordering Q' as it is in priority ordering Q. This is because the priority ordering of the messages with higher priorities than h is unchanged and the direct blocking factor due to the set of messages with lower priority than h depends only on the set of messages $lp(h)$ and not on their relative priority ordering, and is in any case equal to C for all priority bands. All of the messages in bands with priorities higher than j are therefore schedulable in priority ordering Q'.

2. *Priority band i*: Y was previously schedulable in the lower priority band j. Shifting Y up in priority above X results in no change to the blocking factor, but removes interference due to X, hence the worst-case response time for Y can be no greater than it was in priority ordering Q, Y is therefore schedulable in priority ordering Q'.
3. *Priority band j*: Lemma 4 proves that X is schedulable in priority band j.
4. *Priority bands with lower priority than j* ($l \in hp(j)$): Inspection of (5) & (6) and (8) & (9) shows that the response times of each of these messages is the same in priority ordering Q' as it is in priority ordering Q. This is because the set of messages in higher priority bands is the same in both orderings, and the interference due to higher priority messages does not depend on their relative priority ordering. Further, the blocking factor due to the set of messages with lower priority than l depends only on the set of messages $lp(l)$ and not on their relative priority ordering, and is in any case equal to C for all priority bands. All of the messages in bands with priorities lower than j are therefore schedulable in priority ordering Q'.

By repeatedly swapping the priorities of any two adjacent priority bands that are not in TDMPO-FP/FIFO priority order, any arbitrary schedulable priority ordering Q can be transformed into a TDMPO-FP/FIFO priority ordering without any loss of schedulability. □

Corollary 3 *For the case where all nodes use priority queues and all messages have the same worst-case transmission time, TDMPO-FP-FIFO reduces to transmission deadline monotonic priority ordering, which is therefore an optimal priority assignment policy with respect to the sufficient schedulability test given by Davis et al.* (2007) (*recapitulated in Sect.* 4).

Note that transmission deadline (i.e. Deadline minus Jitter) monotonic priority ordering has also been shown to be an effective heuristic policy in the general case with mixed length messages (Davis and Burns 2009a).

6.3 Priority inversion

All of the messages in a FIFO group need to have sufficiently high priorities that the message with the shortest transmission deadline in the group can still meet its deadline. We have shown that with the FIFO-symmetric schedulability analysis introduced in this paper, the most effective way to achieve this is to assign adjacent priorities to all of the messages in a FIFO group. Despite this, we note that the use of FIFO queues still typically results in *priority inversion* with respect to the priority assignment that would be used if all nodes implemented priority queues.

The problem of priority inversion can be seen by considering priority assignment according to the TDMPO-FP/FIFO policy, see Fig. 2. With only PQ-nodes, the priority assigned to each message would depend only on its transmission deadline, with a longer deadline implying lower priority. With FIFO queues, there are two forms of priority inversion: *internal* and *external*. Internal priority inversion takes place within a FIFO queue when messages with longer transmission deadlines enter the queue before, and so are transmitted ahead of, messages with shorter transmission deadlines. External priority inversion occurs because all of the messages in a FIFO group effectively obtain priorities based on the shortest transmission deadline of any message in that group. This has the effect of creating priority inversion with respect to messages sent by other nodes that have transmission deadlines between the maximum and minimum transmission deadlines of messages in the FIFO group. This is illustrated in Fig. 2, where messages causing external priority inversion are shaded in grey.

In Fig. 2, observe that the messages within each FIFO group have their priorities assigned according to transmission deadline monotonic priority assignment. We recommend this approach as although it does not alter the sufficient worst-case response times of the messages as calculated by our analysis, it could result in lower actual worst-case response times for those messages in the group that have shorter transmission deadlines.

7 Case study: automotive

To show that our priority assignment policies and schedulability analysis work with a real application we analysed a CAN bus architecture from the automotive domain, first presented by Kollmann et al. (2010). Figure 3 shows this architecture. The system consists of a CAN bus connecting 10 ECUs. There are a total of 85 messages sent on the bus. The number of messages sent by each ECU is given by the annotations in Fig. 3. All messages are sent strictly periodically and share a common release time. The intended bus speed for this network was 500 kBit/s. We assumed that the queuing jitter for each message was 1 % of its period.

We initially compared five different configurations of the system:

Expt. 1: All ECUs used priority queues.
Expt. 2: ECU3 and ECU6 used FIFO queues and the remaining ECUs used priority queues.
Expt. 3: All ECUs used FIFO queues.
Expt. 4: All ECUs used priority queues, but the priority ordering was that established by Expt 3.

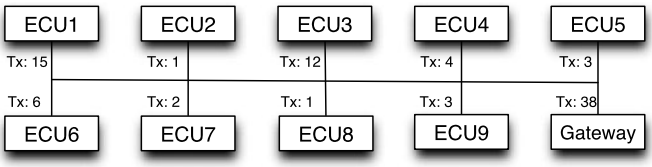

Fig. 3 CAN bus architecture

Expt. 5: All ECUs used priority queues, but the priority ordering used was random.

In each experiment we determined the lowest bus speed commensurate with a schedulable system. The minimum bus speed was found by a binary search with the message priorities assigned according to the OPA-FP/FIFO algorithm (Algorithm 2) using transmission deadline monotonic priority ordering as the reverse ordering for the initial list. (For each FIFO group, only the message with the shortest transmission deadline was included in the initial list.) We simulated the system assuming a bus running at the minimum bus speed, and using the priority ordering obtained during analysis. The simulated network operating time was 1 hour. We used the commercial simulator from Inchron (chronSIM http://www.inchron.com) to produce the simulation results.

There are three lines plotted on each of the graphs. The lines give the following information for each message:

(i) Transmission deadline;
(ii) Worst-case response time computed using the analysis given in Sect. 5.5, assuming the minimum schedulable bus speed for the configuration.
(iii) Maximum observed response time found by simulation, assuming the minimum schedulable bus speed found by analysis.

All of this data is plotted in *ms* on the y-axis using a logarithmic scale. The x-axis on the graphs represents the priority order of the messages. Hence data for the message assigned the highest priority in a particular configuration appears on the LHS of the graph, while data for the lowest priority message appears on the RHS. Note the priority order is different in each experiment.

Figure 4 depicts the results of Expt. 1, where all ECUs used priority queues. In this case, the minimum bus speed was 277 kBit/s, and the corresponding bus utilisa-

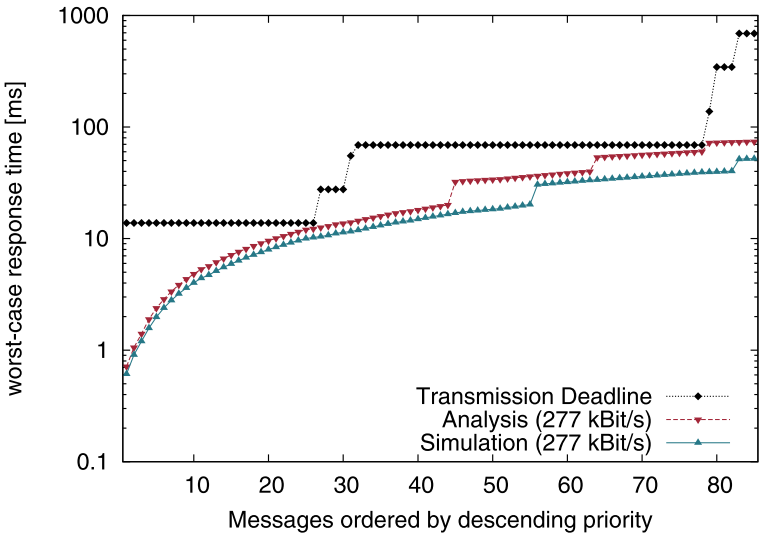

Fig. 4 Response times (PQ only)

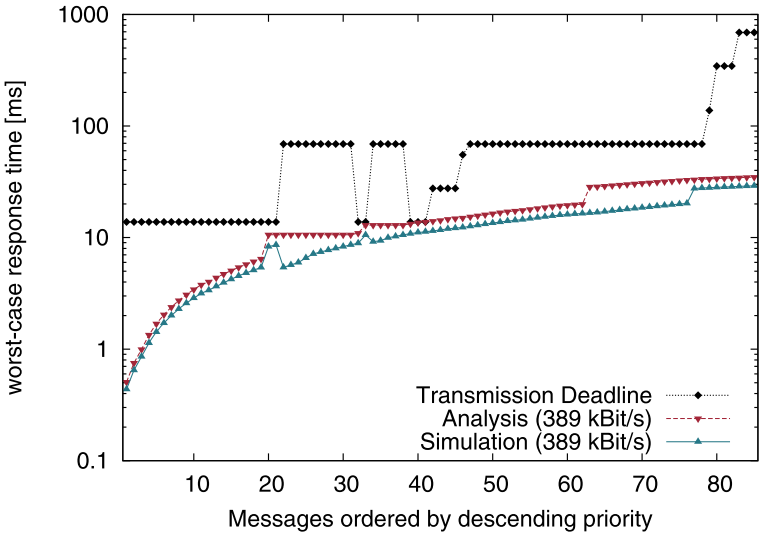

Fig. 5 Response times (FQ and PQ)

tion 84.5 %. We observe that with this bus speed, the 26th highest priority message only just meets its deadline. Further, the results of analysis and simulation are close together. This is because the messages share a common release time, and all of the ECUs used priority-based queues, hence there is very little pessimism in the analysis, and the simulation captures the worst-case scenario well.

Figure 5 depicts the results of Expt. 2, where ECU3 and ECU6 used FIFO queues and the other ECUs used priority queues. In this case, the minimum bus speed was 389 kBit/s, and the corresponding bus utilisation 60.1 %. Our analysis attributes the same worst-case response time to all of the messages in a FIFO queue; this results in the horizontal segments of the analysis lines in Fig. 5. The first FIFO queue is the 12 messages sent by ECU3, and the second, the 6 messages sent by ECU6. The minimum transmission deadline for both FIFO queues was 13.8 ms. Observe that in Fig. 5 the results of analysis and simulation are close together for the messages sent via priority queues, whereas for the messages sent via FIFO queue there are larger gaps. These gaps are predominantly due to the simulation not capturing the worst-case scenario for all of the FIFO-queued messages. This is evident from the variability of the maximum response times obtained via simulation for messages in the same FIFO group.

Figure 6 depicts the results of Expt. 3, where all ECUs used FIFO queues. In this case, the minimum bus speed was 654 kBit/s, and the corresponding bus utilisation only 35.8 %. In contrast to Expt. 1 and 2, this configuration is not schedulable at the intended bus speed for the network of 500 kBit/s. In Expt. 3 (Fig. 6), some of the maximum response times observed in the simulation are very low compared to the worst-case response times computed by the analysis. This is caused by differences in the order in which messages enter the FIFO queues in the simulation, compared to the assumptions made by the analysis.

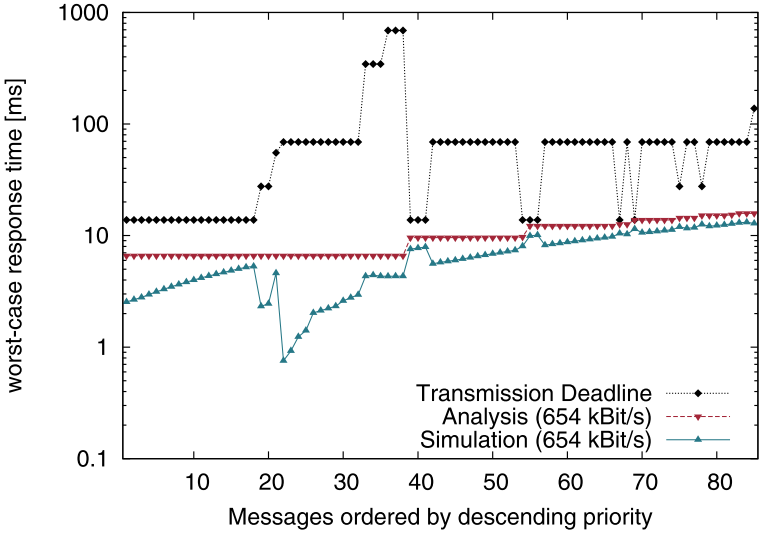

Fig. 6 Response times (FQ only)

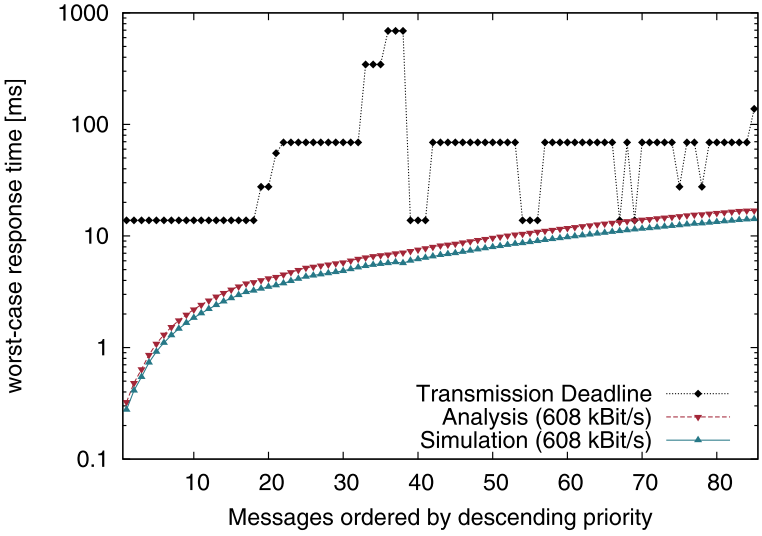

Fig. 7 Response times (PQ only, FQ priorities)

Figure 7 depicts the results of Expt. 4 which used the priority ordering obtained in Expt. 3, but assumed priority queues rather than FIFO queues. In this case, the minimum bus speed required was 608 kBit/s, and the corresponding bus utilisation 38.5 %. Comparison of these results with those from Expt. 1 and Expt. 3 shows that the majority of the performance degradation caused by using FIFO queues occurs as a result of unavoidable external priority inversion in the form of a disrupted priority

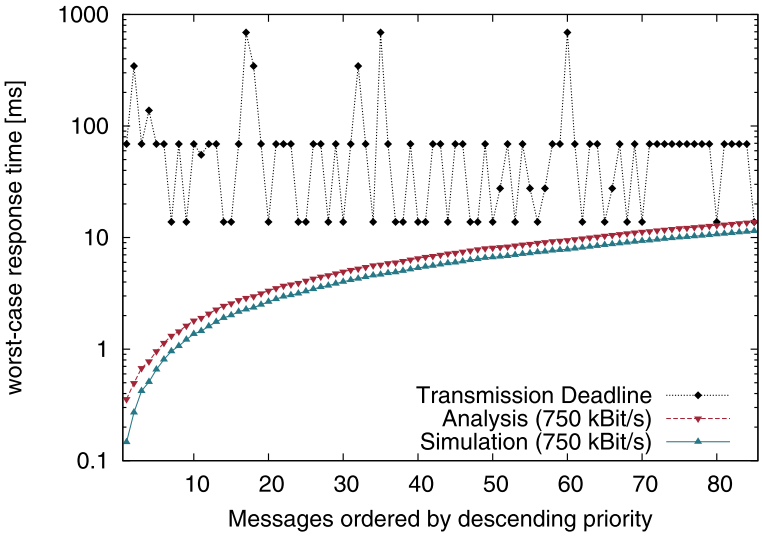

Fig. 8 Response times (PQ only, random priorities)

ordering, rather than as a consequence of internal priority inversion or pessimistic schedulability analysis for FIFO queues.

Finally, Expt. 5 examined 1000 random priority orderings with no correlation between message priority and transmission deadline. This experiment simulates assigning priorities to messages on the basis of the type of data or ECU, or indeed any other metric that has little or no correlation with message transmission deadlines. In this case, the mean value for the minimum bus speed required was 731 kBit/s (min 618 kBit/s, max 750 kBit/s), and the corresponding bus utilisation 32.0 % (max 37.8 %, min 31.2 %). Figure 8 depicts the results of Expt. 5 for the worst of the random priority orderings, which required a minimum bus speed of 750 kBit/s in order to be schedulable. It is clear from the graph, that it is the assignment of a low priority (80th highest priority) to a message with a short transmission deadline that results in the need for such a high bus speed. Expt. 5 is directly comparable with Expt. 1 and shows the importance of appropriate priority assignment. In this case, arbitrary priority assignment increased the minimum bus speed required by 163 % while reducing the maximum schedulable bus utilisation from 84.5 % to 32.0 % (figures for the average case).

The results of the experiments are summarised in Table 2.

7.1 Gateways and multiple FIFO queues

Our case study is typical of automotive applications in that it includes a *gateway* ECU, which is connected to two CAN buses and used to transfer data between them.

The gateway ECU has 38 messages to transmit, which is far more than the number of transmit buffers available in most CAN controllers. A seemingly attractive design solution for the gateway is to use a single FIFO queue; however, as we will see, such a choice can significantly degrade the real-time performance of the network, compared

Table 2 Case study: FIFO queues: summary of results

Expt.	Node type	Priority order	Min bus speed	Max bus util.
1	All PQ	OPA	277 Kbit/s	84.5 %
2	2 FQ, 8 PQ	OPA-FP/FIFO	389 Kbit/s	60.1 %
3	All FQ	OPA-FP/FIFO	654 Kbit/s	35.8 %
4	All PQ	Priority ordering from Expt. 3	608 Kbit/s	38.5 %
5	All PQ	Random[a]	731 Kbit/s	32.0 %

[a]Values are the average for 1000 random orderings

Table 3 Case study: gateway multiple FIFO queues: summary of results

Expt.	Gateway	Priority order	Min bus speed	Max bus util.
6	1-FQ	OPA-FP/FIFO	388 Kbit/s	60.3 %
7	2-FQ	OPA-FP/FIFO	285 Kbit/s	82.1 %
8	3-FQ	OPA-FP/FIFO	277 Kbit/s	84.5 %

to implementing a priority queue. If a priority queue implementation is not possible, then a viable alternative may be to implement multiple FIFO queues, each of which uses a separate hardware transmit buffer in the gateway's CAN controller to send its messages. We note that some CAN devices such as the PIC32MX provide specific hardware support for multiple FIFO queues in this way.

In this section, we report the results of three further experiments examining the use of FIFO queues in the gateway ECU. In each of these experiments, ECUs 1–9 all used priority queues; however, we varied the behaviour of the gateway ECU as follows:

Expt. 6: The gateway used a single FIFO queue.
Expt. 7: The gateway used two FIFO queues. The 18 messages with the same (shortest) transmission deadline of less than 20 ms shared the 1st FIFO queue and the rest of the messages sent by the gateway shared the 2nd FIFO queue.
Expt. 8: The gateway used three FIFO queues. The first 18 messages by transmission deadline shared the 1st FIFO queue, the next 14 messages the 2nd FIFO queue, and the remaining messages the 3rd FIFO queue.

Note the allocation of messages to FIFO queues was done on the basis of grouping messages with similar transmission deadlines together, as this minimises priority inversion.

Table 3 summarises the results of the three experiments.

The results for Expt. 1 where the gateway used a priority queue are shown in Fig. 4. Figure 9 shows that using a single FIFO queue for the gateway increased the minimum schedulable bus speed from 277 Kbit/s (in the case of a priority queue) to 388 Kbit/s, and reduced the maximum achievable bus utilisation from 84.5 % to 60.3 %. Using two FIFO queues made a significant improvement, reducing the priority inversion caused by the sub-set of gatewayed messages with transmission deadlines greater than 20 ms. This decreased the minimum schedulable bus speed to 285

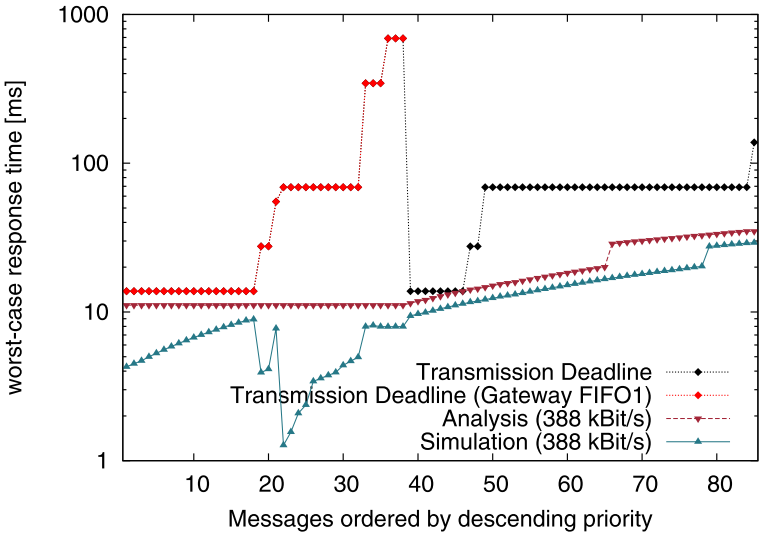

Fig. 9 Response times (gateway 1-FQ)

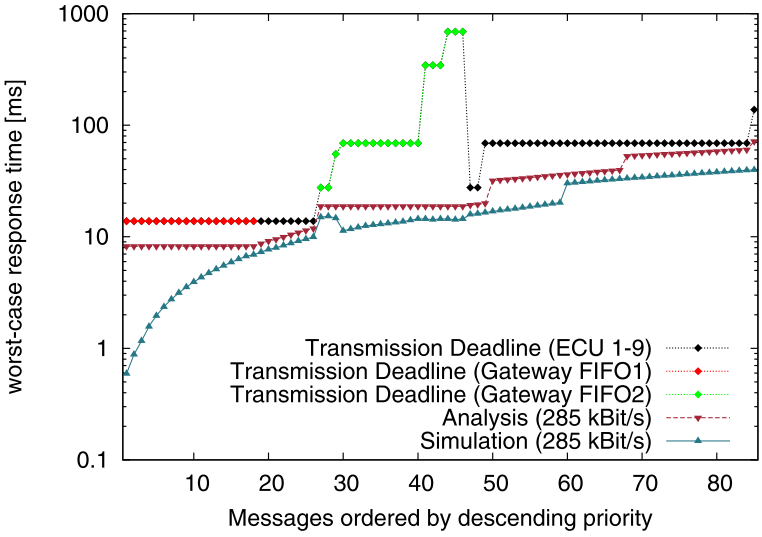

Fig. 10 Response times (gateway 2-FQ)

Kbit/s, and increased the maximum achievable bus utilisation to 82.1 %, as shown in Fig. 10. Finally, using three FIFO queues produced results that were equivalent in performance terms to using a priority queue, see Fig. 11.

Note, in Figs. 9, 10, and 11, the transmission deadlines of messages sent by the gateway are colour coded to show which FIFO they belong to.

These experiments show that, for the case-study, a configuration where the gateway uses three FIFO queues is far more effective than the default option of using just

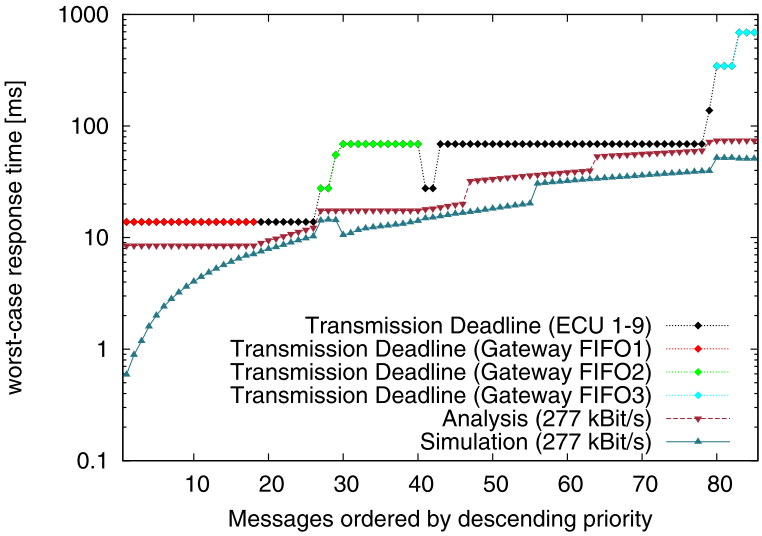

Fig. 11 Response times (gateway 3-FQ)

one FIFO queue. In this case, using multiple FIFO queues and grouping messages by transmission deadline greatly reduces the amount of priority inversion, and significantly improves the real-time performance of the network with respect to just using one FIFO queue.

8 Experimental evaluation

In this section we explore further the effects that FIFO queues and priority assignment policies have on the maximum bus utilisation. Our experimental evaluation examined a system with 8 nodes and 80 messages connected via a single CAN bus. We considered five different configurations of this network. In Config. #1, all of the nodes used priority queues. Configs. #2, #3, and #4 increased the number of nodes using FIFO queues from 2, to 4 to 8 (1/4, 1/2 and all nodes respectively). In Configs. #1–#4, message priorities were assigned according to the TDMPO-FP/FIFO policy as depicted in Fig. 2. (As all of the messages were of the same length, this priority ordering was optimal.) In contrast, in Config. #5, message priorities were assigned at random, and all nodes used priority queues.

To examine the performance of these five configurations, we randomly generated 10,000 sets of messages as follows:

- The period of each message was chosen according to a log-uniform distribution from the range 10–1000 ms; thus generating an equal number of messages in each time band (e.g. 10–100 ms, 100–1000 ms etc.).
- The deadline of each message was equal to its period.
- The jitter of each message was chosen according to a uniform random distribution in the range 2.5 ms to 5 ms.

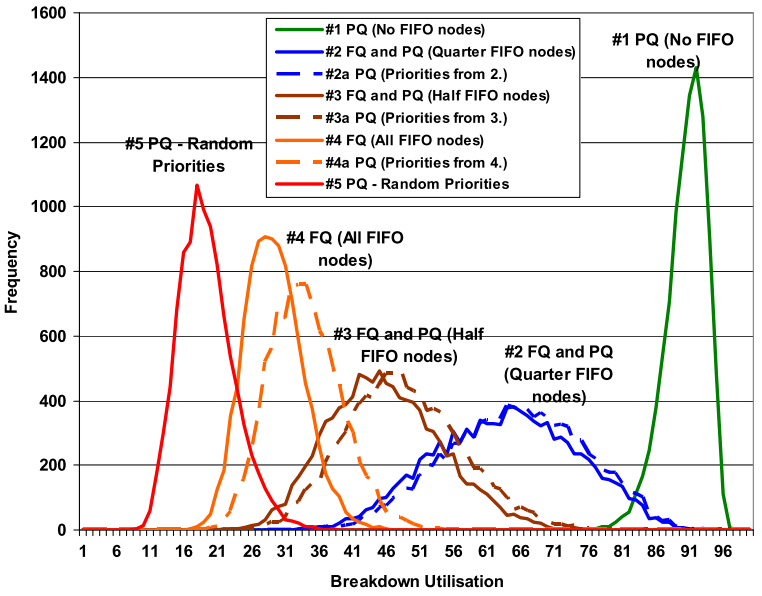

Fig. 12 Frequency distribution of max bus utilisation (8 nodes, 80 messages, 10,000 message sets)

- Each message contained 8 data bytes.
- Each message was randomly allocated to one of the 8 nodes on the network, thus on average, each node transmitted 10 messages.
- All messages were assumed to have 11-bit identifiers.

For each configuration, we computed the maximum bus utilisation for each message set. This was done via a binary search combined with the schedulability analysis given in Sects. 4 and 5. A bin size of 1 % was used in the frequency distribution plots, with message sets with maximum bus utilisations in the range 50.00 % to 50.99 % recorded in the 50 % bin.

The solid lines in Fig. 12 illustrate the frequency distribution of the maximum bus utilisation across the 10,000 message sets for each of the five configurations. From Fig. 12, it is clear that the use of FIFO queues significantly degrades the real-time performance of the network. With all eight nodes using priority queues (#1), the mean value of the maximum bus utilisation was 89.5 %. With a quarter of the nodes using FIFO queues (#2), this reduced to 62.7 %, and with half of the nodes using FIFO queues (#3) it further reduced to 44.9 %. Finally, with all eight nodes using FIFO queues (#4) the mean value of the maximum bus utilisation degraded to just 28.4 %. Worse still was random priority assignment (#5) with a mean value of just 18.4 %; despite using priority queues.

Figure 12 also shows results for the priority orderings obtained from Configs. #2, #3, and #4, assuming that all nodes use priority queues. These results are labelled #2a, #3a, and #4a respectively (dashed lines). The differences between Configs. #1, #2a, #3a, and #4a are indicative of the performance degradation caused by the FIFO queues due to external priority inversion (i.e. priority inversion with respect to mes-

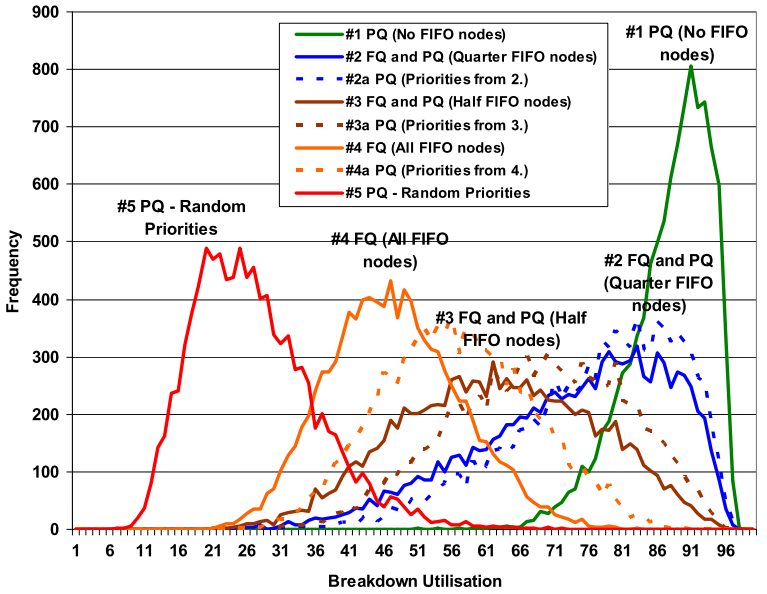

Fig. 13 Frequency distribution of max bus utilisation (8 nodes, 20 messages, 10,000 message sets)

sages sent by other nodes). By contrast, the difference between the pairs of Configs. #2–#2a, #3–#3a, and #4–#4a are indicative of the performance degradation caused by the FIFO queues due to internal priority inversion (i.e. priority inversion with respect to messages sent by the same node), and also potential pessimism in the schedulability analysis for FIFO queues. As expected, the degradation in performance due to external priority inversion is much larger than that due to internal priority inversion, which affects only a limited number of messages.

We repeated our experimental evaluation of an 8 node system for message sets of size 20 and 40. The form of the results and the broad conclusions that can be drawn from them remained the same as with message sets of size 80. However, with fewer messages to randomly allocate to each node, the performance degradation due to each FIFO queue became somewhat smaller. (This is expected as in the limit, with just one message per node, FIFO and priority queues are equivalent.) Results for 8 nodes and message sets of sizes 20, 40 and 80 are summarised in Table 4 and depicted for message sets of sizes 80 and 20 in Figs. 12 and 13.

We also repeated our experimental evaluation for 16 and 24 node systems with message sets of size 160 and 240 respectively. As the average number of messages per node was the same as the case with 8 nodes and 80 messages, the results were also similar. Results for message sets of sizes 80, 160 and 240 are summarised in Table 5 and depicted for message sets of sizes 80 and 240 in Figs. 12 and 14.

As the average number of message per node was constant in these experiments, the average of the maximum achievable bus utilisation varied only a small amount. However, with more messages the frequency distributions became sharper, and the maximum achievable bus utilisation increased slightly. The latter effect is due to the fact that with more messages, message transmission times are smaller with respect

Table 4 Evaluation: 8 nodes, varying the number of messages per node

Config.	Node types	Priority order	Mean of max bus util.		
			$n = 20$	$n = 40$	$n = 80$
1	All PQ	TDMPO	86.8 %	88.4 %	89.5 %
2	1/4 FQ, 3/4 PQ	TDMPO-FP/FIFO	72.7 %	68.1 %	62.7 %
3	1/2 FQ, 1/2 PQ	TDMPO-FP/FIFO	61.6 %	53.6 %	44.9 %
4	All FQ	TDMPO-FP/FIFO	46.5 %	36.9 %	28.4 %
5	All PQ	Random	26.1 %	21.5 %	18.4 %

Table 5 Evaluation: varying number of nodes and messages with the same average number of messages per node

Config.	Node types	Priority order	Mean of max bus util.		
			8 nodes $n = 80$	16 nodes $n = 160$	24 nodes $n = 240$
1	All PQ	TDMPO	89.5 %	90.3 %	90.7 %
2	1/4 FQ, 3/4 PQ	TDMPO-FP/FIFO	62.7 %	65.6 %	67.0 %
3	1/2 FQ, 1/2 PQ	TDMPO-FP/FIFO	44.9 %	47.2 %	48.3 %
4	All FQ	TDMPO-FP/FIFO	28.4 %	29.8 %	30.6 %
5	All PQ	Random	18.4 %	16.3 %	15.4 %

to overall response times, and so the effect of non-pre-emptive transmission becomes less pronounced, and so schedulability improves.

In the case of Config. #5, using a random priority order, the average achievable bus utilisation decreased as the number of nodes and messages increased, even though the average number of messages per node remained constant. This was due to the fact that with a larger number of messages, there is a smaller probability that none of the messages with short deadlines will be assigned low priorities (for example in the lowest 5 % of messages by priority), hence the frequency distribution is less spread out towards higher utilisation values, and has a lower mean.

Overall, our experimental evaluation shows that real-time network performance, measured in terms of the maximum achievable bus utilisation is sensitive to the following:

- the proportion of nodes on the network implementing FIFO queues;
- the number of messages sent by FQ-nodes, and
- the range of transmission deadlines of messages in each FIFO group compared to other messages sent on the network.

Increasing any/all of these factors increases priority inversion, to the detriment of network performance.

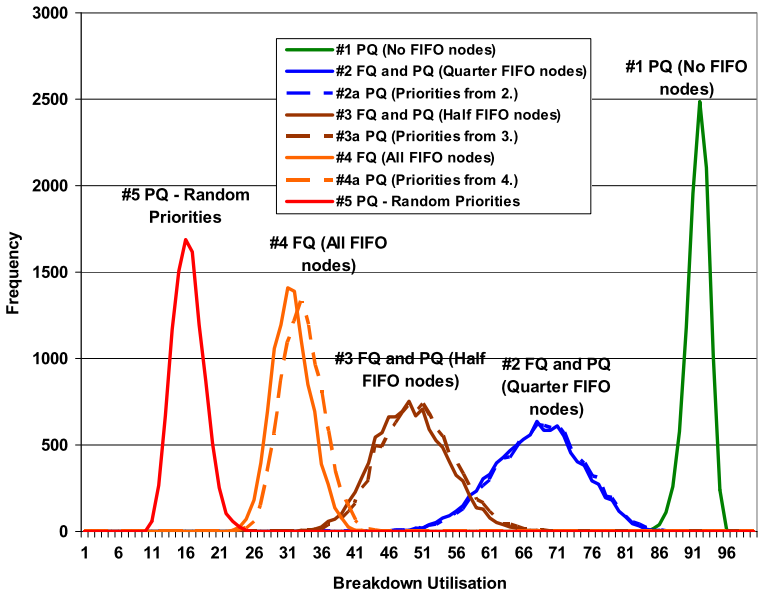

Fig. 14 Frequency distribution of max. bus utilisation (24 nodes, 240 messages, 10,000 message sets)

8.1 Gateways and multiple FIFO queues

We now explore further the effect that using FIFO queues in gateway applications has on the maximum achievable bus utilisation.

To investigate this, we evaluated a network with 120 messages in total, 48 of which were sent by a gateway node. All other messages were assumed to be sent by nodes implementing priority queues, hence the results hold independent of the number of non-gateway nodes on the network. The message parameters were generated as described previously, with 48 messages allocated to the gateway and the remainder to the other nodes.

We considered seven different configurations of the gateway node. In Config. #1, the gateway used a priority queue. In Configs. #2 to #6, the gateway implemented 16, 8, 4, 2, and 1 FIFO queues respectively, which were used to transmit its 48 messages. In Configs. #1 to #6, message priorities were assigned according to the TDMPO-FP/FIFO policy as depicted in Fig. 2. (As all the messages were of the same length, this priority ordering was optimal.) In contrast, in Config. #7, message priorities were assigned at random, and the gateway again used a priority queue.

When the gateway used more than one FIFO queue, then the messages sent by the gateway were sorted according to their transmission deadlines, and the n_G/N_{FIFO} messages with the shortest transmission deadlines were assigned to the first FIFO queue, where n_G is the number of messages sent by the gateway, and N_{FIFO} is the number of FIFO queues it uses. The next n_G/N_{FIFO} messages, ordered by transmission deadline, were assigned to the 2nd FIFO queue, and so on. This simple allocation heuristic ensured that all of the FIFO queues had a small number of messages, and that the messages in each FIFO queue had broadly similar transmission deadlines.

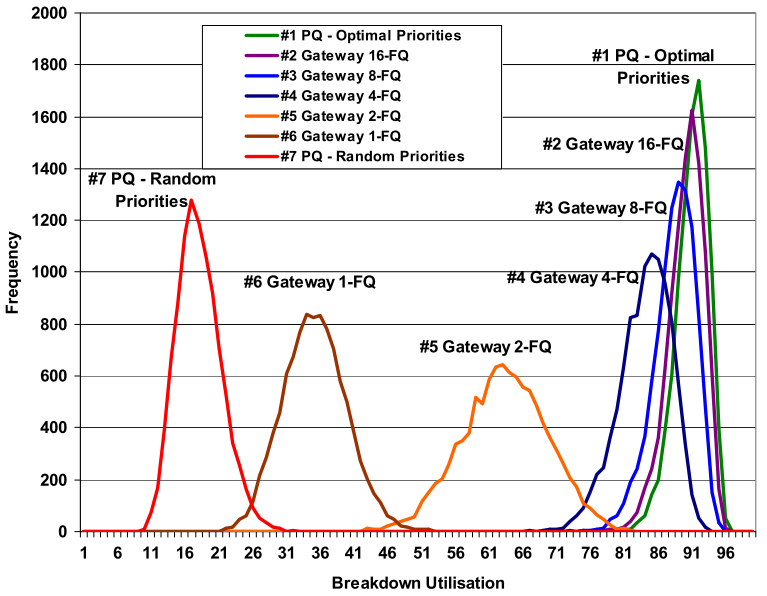

Fig. 15 Frequency distribution of max bus utilisation for different gateway configurations (120 messages, 48 sent by the gateway, 10,000 message sets)

Table 6 Evaluation: gateway configurations

Config.	Gateway	Priority order	Mean of max bus util.
#1	PQ	TDMPO	90.0 %
#2	16-FQ	TDMPO-FP/FIFO	89.1 %
#3	8-FQ	TDMPO-FP/FIFO	87.4 %
#4	4-FQ	TDMPO-FP/FIFO	83.1 %
#5	2-FQ	TDMPO-FP/FIFO	62.3 %
#6	1-FQ	TDMPO-FP/FIFO	34.2 %
#7	PQ	Random	17.1 %

Figure 15 shows the frequency distribution of the maximum achievable bus utilisation for the seven different configurations, and the 10,000 randomly generated sets of messages used. It is evident from Fig. 15 that, with the gateway sending a large number of messages with diverse transmission deadlines, using a single FIFO queue results in poor network performance. The average value for the maximum achievable bus utilisation in this case was just 34.2 %. However, performance was significantly improved by implementing multiple FIFO queues in the gateway. Using two FIFO queues improved the average value for the maximum achievable bus utilisation to 62.3 %. While utilising 4, 8 or 16 FIFO queues, resulted in performance approaching that of a priority queue. A summary of these results is given in Table 6.

We note that the results of our evaluation are based on the use of a simple heuristic for allocating messages to FIFO queues. We expect that in many cases, improved performance could be obtained with fewer FIFO queues by using a more sophisti-

cated message allocation policy. This is borne out by the results of the case study. Investigation of such policies is however beyond the scope of this paper.

9 Summary and conclusions

The major contribution of this paper is the derivation of sufficient response time analysis for CAN where some of the nodes on the network implement FIFO queues, while others implement priority queues. This analysis is *FIFO-symmetric* in that it attributes the same worst-case response time (measured from the time a message is queued in the sending node until it is received by other nodes on the bus) to all of the messages that share the same FIFO. For this schedulability analysis, we proved that it is optimal to assign adjacent priorities to messages that share the same FIFO. We modified Audsley's Optimal Priority Assignment algorithm to provide an overall priority assignment policy (OPA-FP/FIFO) that is optimal with respect to our analysis for both priority-queued messages and groups of messages that share a FIFO. Further, we showed that a simple policy based on transmission deadlines (TDMPO-FP/FIFO), depicted in Fig. 2, is optimal with respect to our analysis for the specific case when all messages are of the same length.

Although this paper provides schedulability analysis for CAN assuming FIFO queues, we cannot recommend the use of such queues. By comparison with priority queues, FIFO queues inevitably cause priority inversion which is detrimental to real-time performance.

Using appropriate optimal priority assignment policies in both cases, we were able to make a like-for-like comparison between the use of priority queues and FIFO queues, thus determining the specific penalty incurred by the latter in terms of network performance. We found that the use of FIFO queues significantly increases the minimum bus speed necessary to ensure that all deadlines are met. This was illustrated in our case study where allowing just two ECUs (sending 18 out of the 85 messages) to use FIFO queues increased the minimum bus speed required from 277 kBit/s with priority queues to 389 kBit/s, a 40 % increase. With all ECUs using FIFO queues, the minimum bus speed required increased to 654 kBit/s; an increase of over 130 %. Using FIFO queues reduces the maximum bus utilisation achievable before any deadlines are missed, thus limiting the scope for extending a system by adding further messages without having to increase bus speed. In our case study, the maximum bus utilisation with priority queues was 84.5 %, this reduced to 60.1 % when two ECUs used FIFO queues, and to just 35.8 % when all of the ECUs used FIFO queues. These figures were backed-up by our experimental evaluation of an eight node system with 80 messages. This evaluation of 10,000 randomly generated message sets showed a degradation in the mean value of the maximum bus utilisation from 89.5 % with all nodes using priority queues, to 62.7 % with two nodes using FIFO queues, to 44.9 % with four nodes using FIFO queues, to just 28.4 % with all eight nodes using FIFO queues. Such reductions in achievable utilisation not only increase the minimum bus speed required to obtain a schedulable network, but also decrease the robustness of the network to errors that result in message retransmission.

We recommend that CAN device drivers/software protocol layers implement priority-based queues, rather than FIFO queues whenever possible. FIFO queues are appealing because they are simpler to implement and make the device driver appear more efficient; however, this perceived local gain typically comes at the expense of undermining the priority-based message arbitration scheme used by CAN, and significantly degrading the overall real-time performance capability of the network.

We note that the degree of priority inversion caused and hence the degradation in performance due to using FIFO queues is lower when only a few messages use each FIFO queue or alternatively when the messages that use each FIFO queue have similar transmission deadlines. Under these circumstances, the use of FIFO queues along with appropriate priority assignment may result in a satisfactory solution. If on the other hand, FIFO queues are used for large numbers of messages with a wide range of transmission deadlines, then this can be expected to have a significant detrimental impact on network performance.

For ECUs that act as a gateway from one CAN bus to another and thus have a large number of messages to transmit, if a priority queue implementation is not possible, then system designers may wish to consider using multiple FIFO queues each utilising a separate hardware transmit buffer. An allocation of messages to these multiple FIFO queues can then aim to avoid assigning messages with widely differing transmission deadlines to the same FIFO queue, while also keeping the number of messages in each FIFO queue relatively small. This approach can result in significantly higher network performance than the alternative of using a single FIFO queue. The schedulability analysis and priority assignment policies given in this paper provide the tools necessary to investigate such trade-offs. This was demonstrated in a further configuration of our case study (described in Sect. 8.1), where the minimum bus speed required reduced from 388 kBit/s when the gateway implementation used a single FIFO queue, to 285 kBit/s when it used two FIFO queues, to 277 kBit/s when it used three FIFO queues (assuming all other nodes used priority queues). This compares favourably with the minimum bus speed of 277 kBit/s required when the gateway used a priority queue. These figures equate to maximum achievable bus utilisations of 60.3 % with one FIFO queue, 82.1 % with two FIFO queues, 84.5 % with three FIFO queues, and the same 84.5 % with a priority queue. These figures were backed up via further empirical evaluation showing that reducing priority inversion via the use of multiple FIFO queues, rather than a single FIFO queue, within a gateway node is effective in reducing the minimum required bus speed and so increasing the maximum achievable bus utilisation.

Finally, both our case study and experimental evaluation confirmed that appropriate priority assignment is vital to obtaining effective real-time performance from Controller Area Networks. Using a random priority assignment policy, representative of priority assignment based on the type of data and ECU, or indeed any other metric that has little or no correlation with transmission deadlines, increased the minimum bus speed required from 277 kBit/s to 731 kBit/s, and reduced the maximum bus utilisation from 84.5 % to just 32.0 % in the case study, as compared to an optimal priority assignment policy. This data was backed up by our evaluation of an eight node system with 80 messages. Here, for message sets of size 80, a random priority assignment policy resulted in values for the maximum bus utilisation, for 10,000 randomly generated message sets, in the range 8 % to 45 % with a mean of just 18.4 %,

compared to a range of 69 % to 96 % and a mean of 89.5 % when an optimal priority assignment policy was used. We therefore strongly recommend that in Controller Area Networks, message IDs are assigned using an optimal or near optimal priority ordering reflecting message transmission deadlines.

9.1 Recommendations and further research

The research presented in this paper serves two main purposes. Firstly, it highlights the detrimental effect that using FIFO queues can have on network performance. Here, our aim was to inform the design choices made by system integrators and designers, thus ensuring that newly developed systems implement priority queues whenever possible. Secondly, we recognise that due to other factors influencing or constraining design choices, some automotive networks will continue to be built using some ECUs that implement FIFO queues. In this case, the analysis presented in this paper can be used to determining network schedulability. Given that it may not always be possible to avoid using FIFO queues, our results on priority assignment and the use of multiple FIFOs show how to make the most effective use of them.

Further, we highlighted the detrimental effect that using a sub-optimal message ID allocation (priority assignment) can have. We acknowledge that there are sometimes design constraints on priority assignment, for example due to the inclusion of legacy components; however, we hope that our work in this area will motivate system integrators to fully consider priority assignment for messages on CAN, debunking the myth of CAN bus utilisation, that one cannot run CAN reliably at more than 35 % utilisation (Buttle 2012).

We note that the analysis described in this paper has been generalised by Davis and Navet (2012) to messages with arbitrary deadlines, and work-conserving queuing policies,[10] of which FIFO is an example. Davis and Navet (2012) showed that for messages with constrained deadlines, the analysis given in this paper holds not only for FIFO queues but also for work-conserving queuing policies in general.

Finally, we note that many automotive systems make use of offsets between message transmission times as a means of reducing the peak load on the network and hence improving message schedulability. The analysis of FIFO queues in this paper was derived for systems where all of the messages can potentially be queued simultaneously. As such, it provides upper bounds on the response times of messages with offsets; however, this is an area where further research would be useful in obtaining tighter upper bounds on message response times.

Acknowledgements The authors would like to thank Alan Burns for his comments on a previous draft of this paper. This work was partially funded by the UK EPSRC funded Tempo project (EP/G055548/1), the EU funded ArtistDesign Network of Excellence, the German Research Foundation, and the Carl Zeiss Foundation.

[10] A work-conserving queuing policy is one which ensures that whenever the node has any messages in its transmit queue and the bus becomes idle then there is a message ready to be transmitted.

References

Audsley NC (1991) Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical report YCS 164, Dept Computer Science, University of York, UK, Dec 1991

Audsley NC (2001) On priority assignment in fixed priority scheduling. Inf Process Lett 79(1):39–44

Avnet (2006) Avnet core datasheet version 1.0 July 2006

Baker TP, Baruah SK (2009) Sustainable multiprocessor scheduling of sporadic task systems. In: Proceedings of the EuroMicro conference on real-time systems, pp 141–150

Baruah SK, Burns A (2006) Sustainable scheduling analysis. In: Proceedings of the 27th real-time systems symposium, pp 159–168

Bosch (1991) CAN Specification version 2.0. Robert Bosch GmbH, Postfach 30 02 40, D-70442 Stuttgart

Bimbard F, George L (2006) FP/FIFO feasibility conditions with kernel overheads for periodic tasks on an event driven OSEK system. In: Proceeding of ISORC

Broster I (2003) Flexibility in dependable communication. PhD thesis, Department of Computer Science, University of York, UK, August 2003

Broster I, Burns A (2003) An analysable bus-guardian for event-triggered communication. In: Proceedings of the 24th real-time systems symposium, pp 410–419

Broster I, Burns A, Rodríguez-Navas G (2002) Probabilistic analysis of CAN with faults. In: Proceedings of the 23rd IEEE real-time systems symposium (RTSS'02), pp 269–278

Broster I, Burns A, Rodriguez-Navas G (2005) Timing analysis of real-time communication under electromagnetic interference. Real-Time Syst 30(1–2):55–81

Buttle D (2012) Real-time in the prime time. Keynote talk at the EuroMicro conference on real-time systems. Presentation available from http://ecrts.eit.uni-kl.de/index.php?id=69

Casparsson L, Rajnak A, Tindell K, Malmberg P (1998) Volcano—a revolution in on-board communications. Volvo technology report, 1998/1

Davis RI, Burns A (2009a) Robust priority assignment for messages on controller area network (CAN). Real-Time Syst 41(2):152–180

Davis RI, Burns A (2009b) Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. In: Proceedings real-time systems symposium (RTSS), pp 398–409

Davis RI, Burns A (2011) Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. Real-Time Syst 47(1):1–40

Davis RI, Navet N (2012) Controller area network (CAN) schedulability analysis for messages with arbitrary deadlines in FIFO and work-conserving queues. In: Proceedings 9th workshop on factory communication systems, pp 33–42

Davis RI, Burns A, Bril RJ, Lukkien JJ (2007) Controller area network (CAN) schedulability analysis: refuted, revisited and revised. Real-Time Syst 35(3):239–272

Davis RI, Zabos A, Burns A (2008) Efficient exact schedulability tests for fixed priority real-time systems. IEEE Trans Comput 57(9):1261–1276. IEEE Computer Society Digital Library. IEEE Computer Society

Davis RI, Kollmann S, Pollex V, Slomka F (2011) Controller area network (CAN) schedulability analysis with FIFO queues. In: Proceedings 23rd EuroMicro conference on real-time systems, pp 45–56

Di Natale M (2006) Evaluating message transmission times in controller area networks without buffer preemption. In: Proceedings 8th Brazilian workshop on real-time systems

Di Natale M (2008) Understanding and using the controller area network. inst.eecs.berkeley.edu/~ee249/fa08/Lectures/handout_canbus2.pdf

Ferreira J, Oliveira A, Fonseca P, Fonseca JA (2004) An experiment to assess bit error rate in CAN. In: Proceedings of 3rd international workshop of real-time networks, pp 15–18

Hansson H, Nolte T, Norstrom C, Punnekkat S (2002) Integrating reliability and timing analysis of CAN-based systems. IEEE Trans Ind Electron 49(6):1240–1250

Hladik P, Deplanche A, Faucou S, Trinquet Y (2007) Schedulability analysis of OSEKNVDX applications. In: Proceedings international conference on real-time and network systems

ISO 11898-1 (1993) Road vehicles—interchange of digital information—controller area network (CAN) for high-speed communication. ISO Standard-11898, International Standards Organisation (ISO)

Khan DA, Bril RJ, Navet N (2010) Integrating hardware limitations in CAN schedulability analysis. In: Proceedings workshop on factory communication systems, pp 207–210

Kollmann S, Pollex V, Kempf K, Slomka F, Traub M, Bone T, Becker J (2010) Comparative application of real-time verification methods to an automotive architecture. In: Proceedings international conference on real-time and network systems

Martin S, Minet P, George L (2007) Non pre-emptive fixed priority scheduling with FIFO arbitration: uniprocessor and distributed cases. Technical report no 5051, INRIA Rocquencourt

Meschi A, Di Natale M, Spuri M (1996) Priority inversion at the network adapter when scheduling messages with earliest deadline techniques. In: Proceedings EuroMicro conference on real-time systems

Microchip Technology Inc. (2009) PIC32MX family reference manual. DS-61155A

Nolte T (2006) Share-driven scheduling of embedded networks. PhD thesis, Malardalen University Press,

Nolte T, Hansson H, Norstrom C (2002) Minimizing CAN response-time analysis jitter by message manipulation. In: Proceedings 8th IEEE real-time and embedded technology and applications symposium, pp 197–206

Nolte T, Hansson H, Norstrom C (2003) Probabilistic worst-case response-time analysis for the controller area network. In: Proceedings of the 9th IEEE real-time and embedded technology and applications symposium, pp 200–207

Renesas (2010) R32C/160 group hardware manual Renesas MCU M16C Family/R32C/100 series. Rev 1.02. Feb 2010

Rufino J, Verissimo P, Arroz G, Almeida C, Rodrigues L (1998) Fault-tolerant broadcasts in CAN. In: Digest of papers, the 28th IEEE international symposium on fault-tolerant computing, pp 150–159

STMicroelectronics (2001) AN1077 application note. Overview of enhanced CAN controllers for the ST7 and ST9 MCUS. Available from www.st.com

Tindell KW, Burns A (1994) Guaranteeing message latencies on controller area network (CAN). In: Proceedings of 1st international CAN conference, pp 1–11

Tindell KW, Hansson H, Wellings AJ (1994) Analysing real-time communications: controller area network (CAN). In: Proceedings 15th real-time systems symposium, pp 259–263

Tindell KW, Burns A, Wellings AJ (1995) Calculating controller area network (CAN) message response times. Control Eng Pract 3(8):1163–1169

XILINX (2010) LogiCORE IP AXI controller area network (axi_can) (v1.01.a) product specification DS791

Zuhily A, Burns A (2007) Optimality of (D-J)-monotonic priority assignment. Inf Process Lett 103:247–250

Robert I. Davis is a Senior Research Fellow in the Real-Time Systems Research Group at the University of York, and a Director of Rapita Systems Ltd. He received his D.Phil. in Computer Science from the University of York in 1995. Since then he has founded three start-up companies, all of which have succeeded in transferring real-time systems research into commercial products. Robert's research interests include scheduling algorithms and schedulability analysis for real-time systems and networks.

Steffen Kollmann studied computer science with the focus on embedded systems and micro-robotics at the University of Oldenburg. In 2012 he received his doctoral degree at Ulm University. Currently he is working at BTC Embedded Systems AG. His scientific interests include model driven development of embedded systems and corresponding verification methods like schedulability analysis. He mainly applies his scientific results to the automotive domain.

Victor Pollex is a Ph.D. candidate at the Institute of Embedded Systems/Real-Time Systems at Ulm University. He received his Diploma in Computer Science from Ulm University in 2009. His research interests are schedulability analysis for real-time systems and networks including real-time calculus.

Frank Slomka holds a diploma degree (Dipl.-Ing.) in electrical engineering from the Technical University of Braunschweig. After three years developing real-time software for DECT telephones he was with the University of Erlangen-Nuremberg. In 2002 he receives the Ph.D. (Dr.-Ing.) with a thesis on design space exploration in telecommunication systems. From 2002 until 2007 he was an assistant professor for embedded systems at the University of Oldenburg. Since 2007 he is full professor of embedded systems/real-time systems at Ulm University. His research interests are real-time analysis and real-time calculus, design-space exploration, hardware-software co-design and design methodologies and metrics for distributed embedded real-time systems.