Controller Area Network (CAN) Schedulability Analysis with FIFO queues

Robert I. Davis
Real-Time Systems Research Group, Department of Computer Science, University of York, YO10 5DD, York, UK
rob.davis@cs.york.ac.uk

Steffen Kollmann, Victor Pollex, Frank Slomka
Institute of Embedded Systems / Real-Time Systems Ulm University, Albert-Einstein-Allee 11, 89081 Ulm, Germany
{steffen.kollmann, victor.pollex, frank.slomka} @uni-ulm.de

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.

Revision: This technical report was revised in April 2011 to include a section on experimental evaluation.

1. Introduction

Controller Area Network (CAN) [3], [21] 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]

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 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 in [3], with a summary in [11]). 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.

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

1.1. Related work

In 1994, Tindell et al. 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 et al. [30], [31], [32] 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 [33].

The seminal work of Tindell et al. lead to a large body of research into scheduling theory for CAN [5], [6], [7], [8], [17], [18], [25], [26], [27], [28], and was used as the basis for commercial CAN schedulability analysis tools [9].

In 2007, Davis et al. [11] found and corrected significant flaws in the schedulability analysis given by Tindell et al. [30], [31], [32]. 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. [11] 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 [1], [2] 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% [13], [9].

1.2. 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. in [11], 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 [15], there are a number of potential issues that can lead to behaviour that does not match that required by the scheduling model given in [11]. 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) [16];
(iii) less than 3 transmit buffers [24] (Philips 8xC592 (SJA1000), Philips 82C200).

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 [11]. Issues include, delays in refilling a transmit buffer [20], and FIFO queuing of messages in the device driver or CAN controller (The BXCAN and BECAN for the ST7 and ST9 Microcontrollers from STMicroelectronics include hardware support for both priority-queued and FIFO-queued message transmission [29]).

Di Natale [15] notes 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.

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] The commercial tool NETCAR-Analyzer (www.realtimeatwork.com) claims to address the case of FIFO queues.

1.3. Organisation

The remainder of this paper is organised as follows: In section 2, we introduce the scheduling model, notation, and terminology used in the rest of the paper. In section 3 we recap on the sufficient schedulability analysis for CAN given in [11]. Section 4 then extends this analysis to networks where some nodes implement priority-based queues while others implement FIFO queues. Section 5 discusses priority assignment for mixed sets of FIFO-queued and priority-queued messages. Section 6 presents the results of a case study exploring the impact of FIFO queues on message response times and network schedulability. Section 7 further evaluates the effect of priority assignment and FIFO queues on the maximum achievable network utilisation. Finally, section 8 concludes with a summary and recommendations.

2. 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 messages on CAN. This model is based on that used in [11] with extensions to describe FIFO queues.

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 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 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 [11] 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[4], 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. Hence:

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

As noted by Broster [7], 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.

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 sections 3 and 4, as well as exact schedulability tests.

3. Schedulability Analysis with Priority Queues

In this section, we recapitulate the simple sufficient schedulability analysis given in [11]. For networks of PQ-nodes, complying with the scheduling model given in section 2, CAN effectively implements fixed priority non-pre-emptive scheduling. In this case, Davis et al. [11] 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-

[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 in [11] 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.

[5] See [11] for an explanation of why push-through blocking is important.

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 [12]).

4. 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.

4.1. 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. [11] 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).

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

As with (3), 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:

$$R_m = w_m^{n+1} + C_m \quad (6)$$

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.

4.2. 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 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 $D_j \le T_j$ 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:

$$C_m^{SUM} - C_m^{MIN} \quad (7)$$

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

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

[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.

fixed point iteration:

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

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:

$$R_m = w_m^{n+1} + C_m^{MIN} \quad (9)$$

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}.

4.3. Schedulability test with arbitrary priorities

We now derive a schedulability test from (5) & (6) and (8) & (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:

$$f_m = R_m - C_m^{MIN} \quad (10)$$

When the priorities of messages in different FIFO groups are interleaved, this leads to an apparently 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) & (6) and (8) & (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 (8) & (9). Hence by using an outer loop iteration, and repeating response time calculations until the buffering times no longer change, 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).

```
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) & (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) & (6)
18           if( R_m > E_m^MIN ) {
19               return unschedulable
20           }
21       }
22   }
23 }
24 return schedulable
```

Algorithm 1: FIFO Symmetric Schedulability Test

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

4.4. 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.

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

[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.

[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) &(9).

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.

4.5. Schedulability test for FIFO-adjacent priorities

In this section, we derive an improved schedulability test that is only valid for FIFO-adjacent priority ordering*s*.

Recall that Davis et al. [11] 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) & (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) & (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 section 4.3 (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 in [23] and for OSEK/VDX tasks in [4], [19].

5. Priority Assignment Policies

The schedulability test presented in section 4.5 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 section 4.5.

5.1. Optimal priority assignment

Davis et al. [11], showed that, assuming solely priority queuing, Audsley's Optimal Priority Assignment (OPA) algorithm [1], [2] 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 section 4.5. 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.

```
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
```

Algorithm 2: Optimal Priority Assignment (OPA-FP/FIFO)

In [14] Davis and Burns showed that Audsley's OPA

algorithm is optimal with respect to any schedulability test that meets three specific conditions. According 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 2: The OPA-FP/FIFO algorithm is an optimal priority assignment algorithm with respect to the FIFO-symmetric schedulability test of section 4.5 (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 in [13].

5.2. TDMO-FP/FIFO priority assignment

In industrial practice, CAN configurations are often 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 1 illustrates the TDMPO-FP/FIFO priority assignment policy.

Theorem 3: 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 section 4.5 (Algorithm 1 with lines 11-14 omitted), provided that all messages have the same worst-case transmission time.

Proof: See Appendix A.

Corollary 1: 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. in [11] (recapitulated in section 3).

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 [13].

5.3. 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 Figure 1 below. 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 Figure 1, where messages causing external priority inversion are shaded in grey.

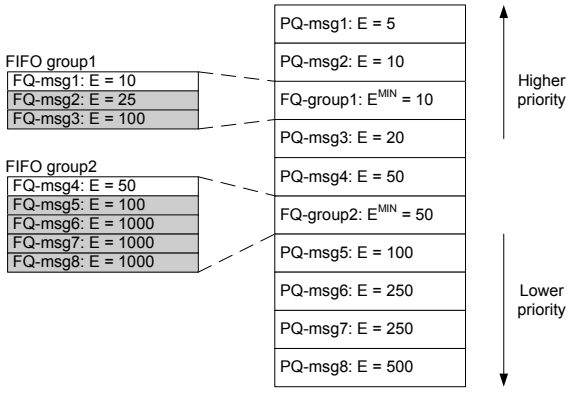

Figure 1: TDMPO-FP/FIFO priority ordering

In Figure 1, 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, in practice it could result in lower actual worst-case response times for those messages in the group that have shorter transmission deadlines.

6. 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 in [22]. Figure 2 shows this architecture. The system consists of a 500 kBit/s 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 Figure 2. All messages are sent strictly periodically and have no offsets with respect to each other. We assumed that the queuing jitter for each message was 1% of its period.

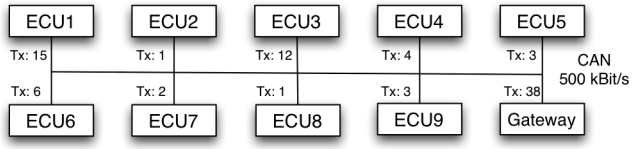

Figure 2: CAN bus architecture

We 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.

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). Based on the priority ordering obtained, we analysed and simulated the system assuming a 500 kBit/s bus. The simulated network operating time was 1 hour. We used the commercial simulator chronSIM from Inchron [10] to produce the simulation results.

There are four 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 section 4.5, assuming a 500 Kbit/s bus;

(iii) maximum observed response time found by simulation, assuming a 500 Kbit/s bus, and

(iv) worst-case response time computed using the analysis given in section 4.5, assuming the minimum schedulable bus speed for the configuration.

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 3 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 utilisation 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 for a 500 kBit/s bus are close together. This is because the messages have no offsets, 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 4 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 Figure 4. 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 Figure 4 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.

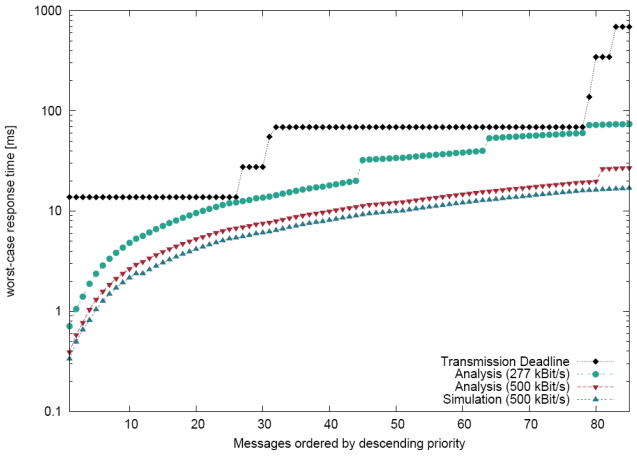

Figure 3: Response Times (PQ only)

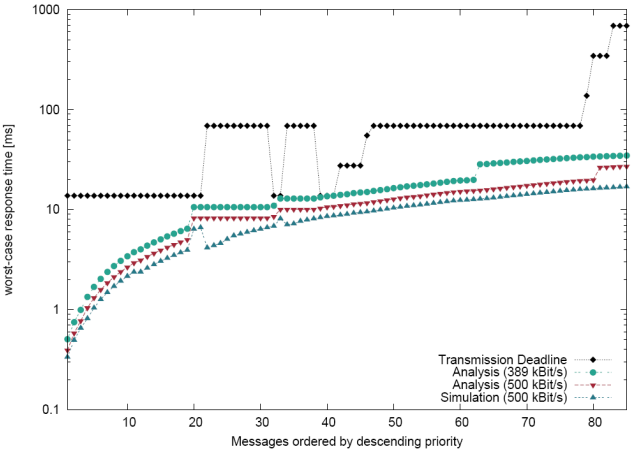

Figure 4: Response Times (FQ and PQ)

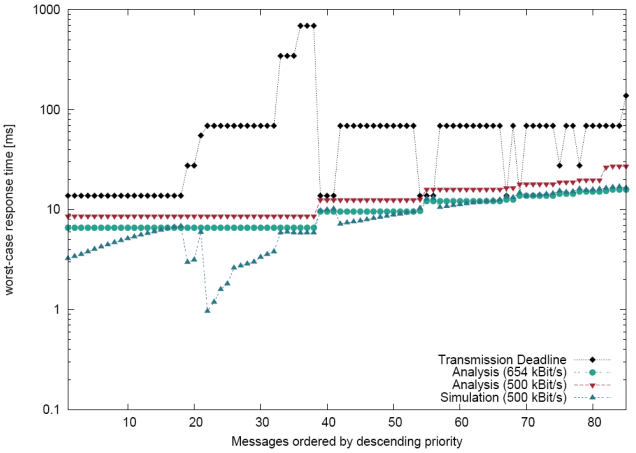

Figure 5: Response Times (FQ only)

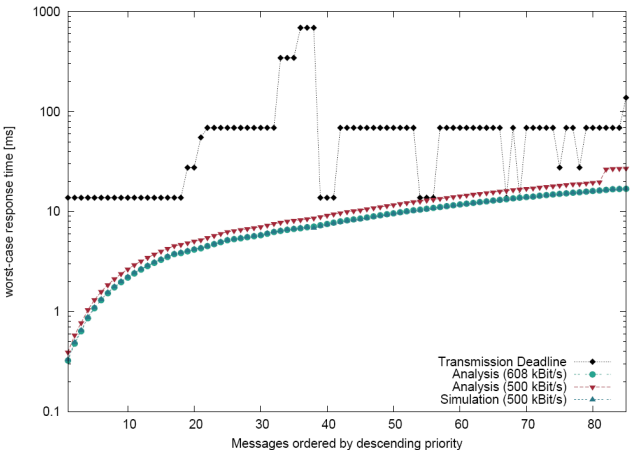

Figure 6: Response Times (PQ only, FQ priorities)

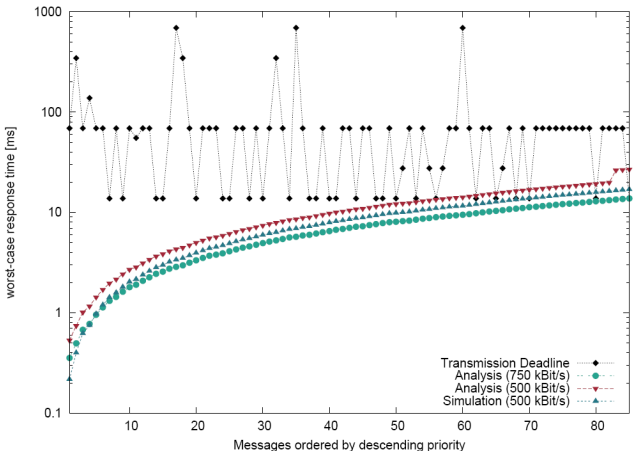

Figure 7: Response Times (PQ only, random priorities)

Figure 5 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 the Expt. 1 & 2, this configuration was not schedulable at a bus speed of 500 kBit/s. At 500 kBit/s, the 54 highest priority messages were found to be schedulable by the analysis. For the remaining lower priority messages, some appear to have worst-case response times that are less than their deadlines; however, this does not imply that such messages are schedulable. Once a single higher priority message is unschedulable, then the assumptions made by the analysis may be broken and the computed worst-case response times no longer valid. For example, the analysis assumes that due to constrained deadlines at most one instance of each of the other messages in the same FIFO group may be ahead of a particular message in the queue. If one of the messages in a FIFO group cannot meet its deadline then this assumption may no longer hold. In Expt. 3, 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.

Figure 6 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 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 7 depicts the results of Expt. 5 for the worst of the random priority orderings, which required a minimum bus speed of 750 kBit/s 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 1 below.

Table 1: Case Study: 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[10]	731 Kbit/s	32.0%

7. 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 configuration #1, all of the nodes used priority queues. Configurations #2, #3, and #4 increased the number of nodes using FIFO queues from 2, to 4 to 8 respectively. In configurations #1–#4, message priorities were assigned according to the TDMPO-FP/FIFO policy as depicted in Figure 1. (As all the messages were of the same length, this priority ordering was optimal). In contrast, in configuration #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 10-1000ms; thus generating an equal number of messages in each time band (e.g. 10-100ms, 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 2.5ms to 5ms.

[10] Values are the average for 1000 random orderings.

- 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 sections 3 and 4.

The solid lines in Figure 8 illustrate the frequency distribution of the maximum bus utilisation across the 10,000 message sets for each of the five configurations. From Figure 8, 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 two nodes using FIFO queues (#2), this reduced to 62.7%, and with four 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 8 also shows results for the priority orderings obtained from configurations #2, #3, and #4, but assuming that all nodes use priority queues. These results are labelled #2a, #3a, and #4a respectively (dashed lines). The difference between configurations #1, #2a, #3a, and #4a is indicative of the performance degradation caused by the FIFO queues due to external priority inversion (i.e. priority inversion with respect to messages sent by other nodes). By contrast, the difference between the pairs of configurations #2–#2a, #3–#3a, and #4–#4a is 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 message sets of sizes 20, 40 and 80 are summarised in Table 2 and depicted in Figure 8, Figure 9, and Figure 10 respectively.

Table 2: Evaluation: Message sets of size 20

Config	Node type	Priority order	Mean of Max. bus util.		
			n=20	n=40	n=80
1	All PQ	TDMPO	86.8%	88.4%	89.5%
2	2 FQ, 8 PQ	TDMPO-FP/FIFO	72.7%	68.1%	62.7%
3	4 FQ, 4 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%

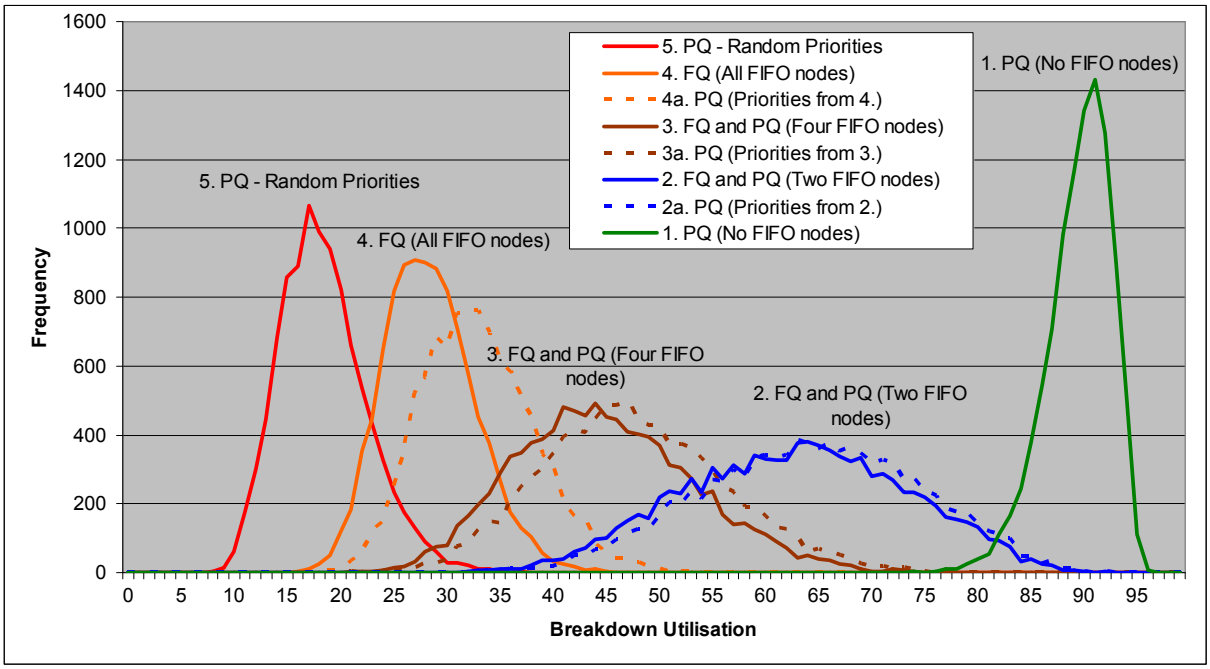

Figure 8: Frequency distribution of max. bus utilisation (8 nodes, 80 messages, 10,000 message sets)

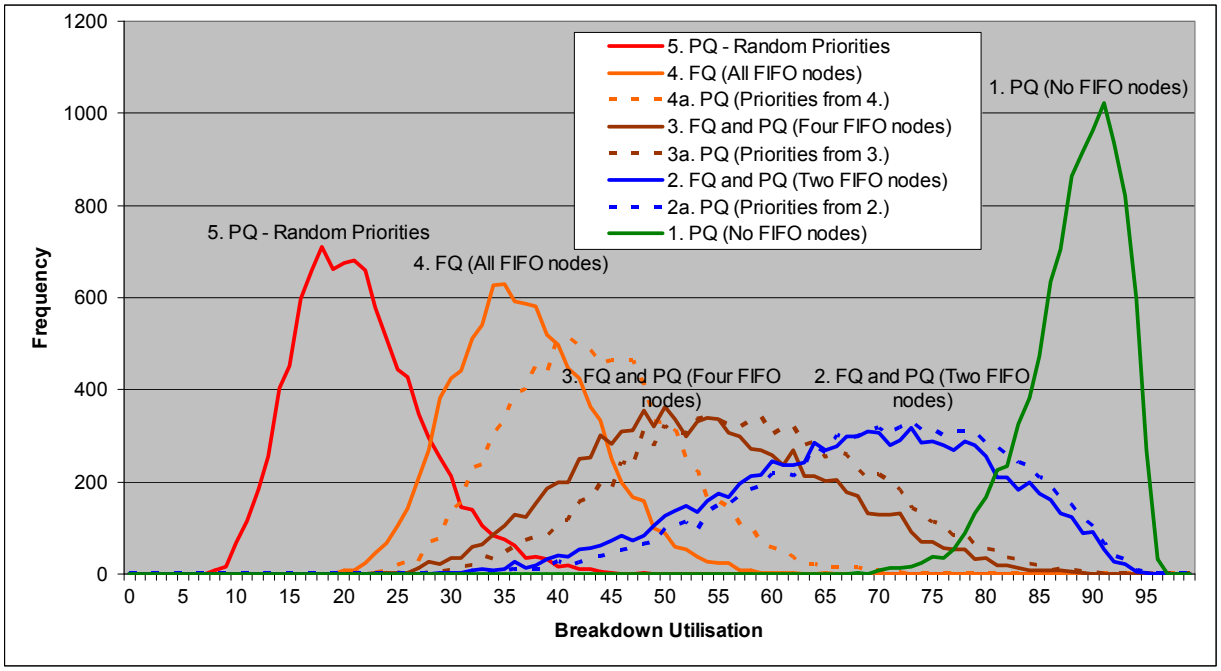

Figure 9: Frequency distribution of max. bus utilisation (8 nodes, 40 messages, 10,000 message sets)

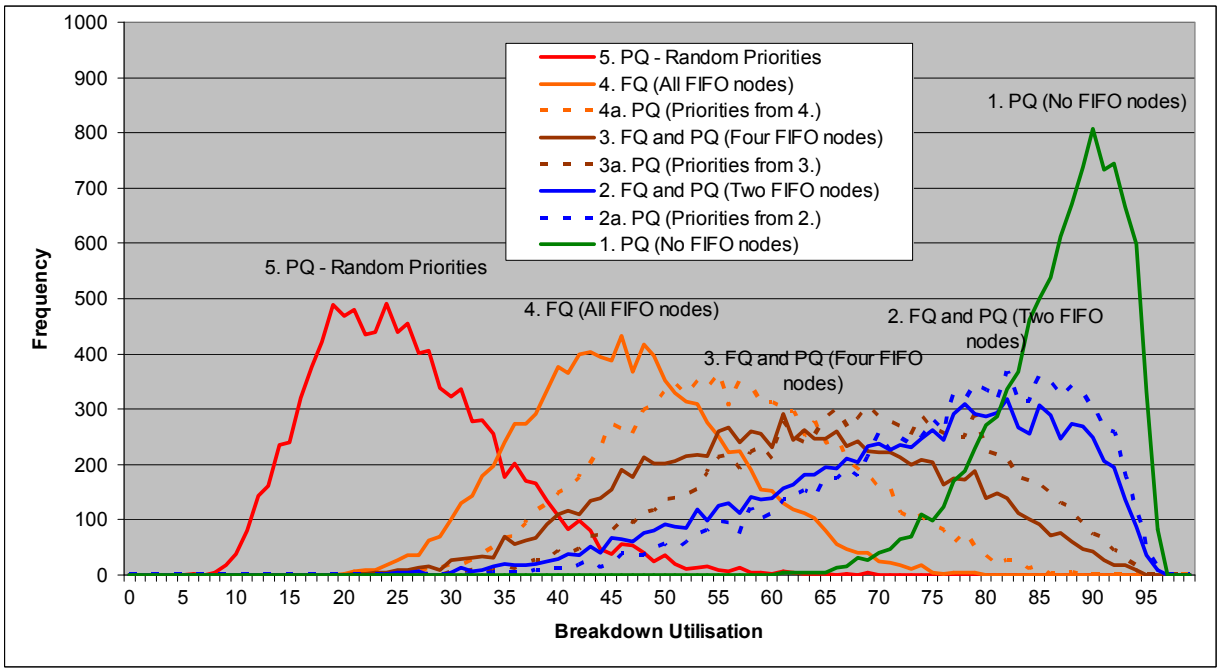

Figure 10: Frequency distribution of max. bus utilisation (8 nodes, 20 messages, 10,000 message sets)

8. 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 Figure 1, 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.

The use of FIFO queues 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 re-transmission.

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 tradeoffs.

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 experimental evaluation of an eight node system with 80 messages. Here, for message sets of size 80, such a 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.

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.

9. References

[1] N.C. Audsley, "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.

[2] N.C. Audsley, “On priority assignment in fixed priority scheduling”, Information Processing Letters, 79(1): 39-44, May 2001.

[3] Bosch. “CAN Specification version 2.0”. Robert Bosch GmbH, Postfach 30 02 40, D-70442 Stuttgart, 1991.

[4] F. Bimbard and L. George. “FP/FIFO feasibility conditions with kernel overheads for periodic tasks on an event driven OSEK system”. In Proceeding of ISORC, 2006.

[5] I. Broster, A. Burns , G. Rodríguez-Navas, “Probabilistic Analysis of CAN with Faults”, *In Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS'02)*, pp. 269-278, December, 2002.

[6] I. Broster and A. Burns. “An Analysable Bus-Guardian for Event-Triggered Communication”. In *Proceedings of the 24th Real-time Systems Symposium*, pp. 410-419, IEEE Computer Society Press, December 2003.

[7] I. Broster. “Flexibility in dependable communication”. *PhD Thesis*, Department of Computer Science, University of York, UK, August 2003.

[8] I. Broster, A. Burns and G. Rodriguez-Navas, “Timing analysis of real-time communication under electromagnetic interference”, *Real-Time Systems*, 30(1-2) pp. 55-81, May 2005.

[9] L. Casparsson, A. Rajnak, K. Tindell, and P. Malmberg. “Volcano - a revolution in on-board communications”. *Volvo Technology Report*, 1998/1.

[10] chronSIM. http://www.inchron.com

[11] R.I. Davis, A. Burns, R.J. Bril, and J.J. Lukkien. “Controller Area Network (CAN) Schedulability Analysis: Refuted, Revisited and Revised”. *Real-Time Systems*, Volume 35, Number 3, pp. 239-272, April 2007.

[12] R.I. Davis, A. Zabos, A. Burns, "Efficient Exact Schedulability Tests for Fixed Priority Real-Time Systems”. *IEEE Transactions on Computers* IEEE Computer Society Digital Library. IEEE Computer Society, September 2008 (Vol. 57, No. 9) pp. 1261-1276.

[13] R.I. Davis, A. Burns "Robust priority assignment for messages on Controller Area Network (CAN)”. *Real-Time Systems*, Volume 41, Issue 2, pages 152-180, February 2009.

[14] R.I. Davis and A. Burns, "Improved Priority Assignment for Global Fixed Priority Pre-emptive Scheduling in Multiprocessor Real-Time Systems”. Real-Time Systems, Volume 47, Issue 1, pages 1-40, 2010.

[15] M. Di Natale, “Understanding and using the Controller Area network” inst.eecs.berkeley.edu/~ee249/fa08/Lectures/ handout_canbus2.pdf.

[16] M. Di Natale, “Evaluating message transmission times in Controller Area Networks without buffer preemption”, In *8th Brazilian Workshop on Real-Time Systems*, 2006.

[17] J. Ferreira, A. Oliveira, P. Fonseca, J. A. Fonseca. “An Experiment to Assess Bit Error Rate in CAN”. In *Proceedings of 3rd International Workshop of Real-Time Networks (RTN2004),* pp. 15-18, Cantania, Italy. June 2004.

[18] H. Hansson, T. Nolte, C. Norstrom, and S. Punnekkat. “Integrating Reliability and Timing Analysis of CAN-based Systems”. *IEEE Transaction on Industrial Electronics,* 49(6): 1240-1250, December 2002.

[19] P. Hladik, A. Deplanche, S. Faucou, and Y. Trinquet, “Schedulability analysis of OSEKNVDX applications”. In Proceedings RTNS, 2007.

[20] D.A. Khan, R.J. Bril, N. Navet, "Integrating hardware limitations in CAN schedulability analysis," IEEE International Workshop on Factory Communication Systems (WFCS) pp.207-210, 18-21 May 2010. doi: 10.1109/WFCS.2010.5548604.

[21] ISO 11898-1. “Road Vehicles – interchange of digital information – controller area network (CAN) for high-speed communication”, *ISO Standard-11898*, International Standards Organisation (ISO), Nov. 1993.

[22] S. Kollmann, V. Pollex, K. Kempf, F. Slomka, M. Traub, T. Bone, J. Becker (2010). "Comparative Application of Real-Time Verification Methods to an Automotive Architecture, " In Proceedings of the 18th International Conference on Real-Time and Network Systems, Nov. 2010.

[23] S. Martin, P. Minet, L. George, “Non pre-emptive Fixed Priority scheduling with FIFO arbitration: uniprocessor and distributed cases”, Technical Report No. 5051, INRIA Rocquencourt, Dec. 2007.

[24] A. Meschi, M. DiNatale, and M. Spuri, “Priority inversion at the network adapter when scheduling messages with earliest deadline techniques,” in Proceedings of Euromicro Conference on Real-Time Systems, June 12-14 1996.

[25] T. Nolte. “Share-driven scheduling of embedded networks”, *PhD Thesis*, Malardalen University Press, May 2006.

[26] T. Nolte, H. Hansson, and C. Norstrom. “Minimizing CAN response-time analysis jitter by message manipulation”. In *Proceedings 8th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02)*, pp 197-206, September 2002.

[27] T. Nolte, H. Hansson, and C. Norstrom, "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 (RTAS'03)*, pp. 200-207, May 2003.

[28] J. Rufino, P. Verissimo, G. Arroz, C. Almeida, and L. Rodrigues. “Fault-tolerant broadcasts in CAN”. In *Digest of Papers, The 28th IEEE International Symposium on Fault-Tolerant Computing (FTCS'98)*. pp. 150-159, June 1998.

[29] STMicroelectronics, “AN1077 Application note. Overview of enhanced CAN controllers for the ST7 and ST9 MCUS” 2001 (available from www.st.com).

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

[31] K.W. Tindell, A. Burns, and A. J. Wellings. “Calculating Controller Area Network (CAN) message response times”. *Control Engineering Practice*, 3(8): 1163-1169, August 1995.

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

[33] A. Zuhily and A. Burns, “Optimality of (D-J)-Monotonic Priority Assignment”. Information Processing Letters, no. 103, pp. 247-250, Apr. 2007.

Appendix A: Transmission deadline monotonic priority assignment

In this appendix, we show that the TDMPO-FP/FIFO priority assignment policy is optimal, with respect to the sufficient schedulability test given in section 4.5 (i.e. Algorithm 1 with lines 11-14 omitted) when all messages have the same worst-case transmission time (C).

Corollary A.1: 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. in [11] (recapitulated in section 3) and the sufficient schedulability tests given in section 4 of this paper, is the same for every message, and is equal to the worst-case message transmission time (C).

Lemma A.1: For a set of messages that all have the same worst-case transmission time (C). 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 section 4.5 (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.

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, $R_{Y,j} \le E_Y$, thus as $E_Y < E_X$, it follows that to prove the Lemma, we need only show that $R_{X,j} \le R_{Y,j}$. 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.

$$B(j) = \max(B_j, C) = C$$

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

(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:

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

Note, in (A.1), 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 (A.2) below.

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

$$w_{Y,j}^{n+1} = C_Y^{SUM} + \sum_{\forall k \in X} \left\lceil \frac{w_{Y,j}^n + J_k + \tau_{bit}}{T_k} \right\rceil C + I(i,w) \quad \text{(A.2)}$$

We can simplify (A.2) by noting that as Y is schedulable according to the assumption given in the Lemma, then

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

Hence, at most one instance of each message in X can contribute to the interference term and so we have:

$$w_{Y,j}^{n+1} = C_Y^{SUM} + C_X^{SUM} + I(i,w) \quad \text{(A.3)}$$

Now let us consider the iterative solution to (A.1), for all values of

$$w_{X,j} \le E_Y - C < \min_{\forall k \in X}(T_k - J_k) - \tau_{bit},$$

only one instance of each message in Y can contribute to the interference term. Hence for $w_{X,j} \le E_Y - C$, (A.1) reduces to:

$$w_{X,j}^{n+1} = C_X^{SUM} + C_Y^{SUM} + I(i,w) \quad \text{(A.4)}$$

Equations (A.3) and (A.4) are the same, hence as we know that (A.3) converges on a value $w_{Y,j} \le E_Y - C$, then (A.4) must also converge on the same value, hence $w_{X,j} = w_{Y,j}$, and so $R_{X,j} = R_{Y,j}$ □

Theorem 3: 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 section 4.5 (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 section 4.5 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 section 4.5. 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 A.1 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 □.