
International Journal of Time-Critical Computing Systems, 20, 5–25, 2001

Improving the Accuracy of Scheduling Analysis Applied to Distributed Systems

Computing Minimal Response Times and Reducing Jitter

WILLIAM HENDERSON william.henderson@unn.ac.uk
School of Computing and Mathematics, University of Northumbria, Newcastle upon Tyne, UK

DAVID KENDALL david.kendall@unn.ac.uk
School of Computing and Mathematics, University of Northumbria, Newcastle upon Tyne, UK

ADRIAN ROBSON adrian.robson@unn.ac.uk
School of Computing and Mathematics, University of Northumbria, Newcastle upon Tyne, UK

Abstract. A well-established approach to the verification of end-to-end response times for distributed, hard real-time systems is an integrated scheduling analysis of both task processing and message communication. Hitherto, published analyses have been confined to the computation of worst-case bounds only and best-case response times have been ignored, assumed to be zero or treated approximately. However, there are compelling reasons for computing both upper and lower bounds on response times, not only to allow the verification of best-case performance but also to improve the accuracy of the overall analysis. This paper describes a precise best-case execution time analysis which reduces jitter and extends distributed scheduling analysis to yield more accurate upper and lower bounds on system response times. The analysis is combined with existing results for worst-case responses in a single scheduling algorithm to compute both upper and lower bounds on end-to-end response in distributed systems.

A design tool has been developed to automate the analysis and support the performance verification of diverse real-time systems composed of tasks executing on multiple processors which communicate using the Controller Area Network (CAN) fieldbus.

Keywords: real-time, scheduling, distributed systems, controller area network, best-case analysis, end-to-end responses

1. Introduction

For economic imperatives or because of physical constraints, many control systems are decentralised with actuators, sensors, control and other processors at widely different locations—these are sometimes described as distributed data communication and control systems (DDCCS). Such systems require data (process variables, control signals and other values) to be transmitted over networks (Ray, 1988). This type of control infrastructure is found in aircraft, manufacturing plant, increasingly in road vehicles and elsewhere. A communications network (as opposed to dedicated point-to-point wiring) introduces delays which may be variable if the channel is shared between several control loops. Different configurations are possible for decentralised control systems and network delays may be introduced once or a multiple number of times in a closed loop. Control algorithms are designed and implemented assuming a periodic behaviour. Periodicity is often important in achieving the required control performance (rise time, overshoot, etc.) and in maintaining

stability (Őzgűner, 1989). There are strict deadlines placed on closed loop response times which emerge from a control system stability analysis (Krtolica et al., 1994). In order to verify that a given control hardware and software system is capable of providing the required quality of control, a means of predicting end-to-end response times is required. Preferably, this information should be available at an early design stage.

An approach to the verification of end-to-end response times for distributed real-time software systems is the holistic scheduling analysis proposed by Klein (Klein et al., 1993) and Tindell (Tindell and Clark, 1994). The technique has been used to predict worst-case performance of periodic distributed systems composed of task sets scheduled on one or more processors and in which tasks synchronise and communicate by message passing supported by kernels and/or networks. The aim of such analysis is to compute time bounds on system responses and thereby verify that important performance deadlines can be met by an implementation. Essentially, the technique utilises scheduling models to predict worst-case task execution times on processors, and message queue and transmission times on networks. Periodic system transactions are defined as precedence constrained tasks and messages which are executed in sequence. Worst-case response times for transactions are computed by summing individual task and message response times. It is necessary in the overall analysis to account for the fact that portions of a response executed on processors other than the first are not necessarily initiated periodically but exhibit jitter in their release times (Klein et al., 1993).

Prior to recent work of Gutiérrez (Gutiérrez et al., 1998), published results on integrated scheduling analysis has been limited to the computation of worst-case response times for transactions. Thus, end-to-end responses have been computed assuming worst interference conditions between tasks on processors and messages on networks. Additionally, jitter, the variation of response time, has been pessimistically computed by assuming that the best-case response times are zero. This has lead to an overestimation of worst-case end-to-end response times in previous analysis because jitter in high priority transactions extends the response time of low-priority transactions. For many real-time systems, hard deadlines may be adequately expressed in terms of worst-case behaviour. However, there are situations where this is not the case and it is necessary also to predict best-case performance. In process control it is often required to perform certain operations periodically at exactly defined instants (Colnarič et al., 1998). Suppose that a control task were required not only to output its computed response prior to its next period, but must output it exactly 95 *ms* after the availability of the input value (perhaps within, say, ± 1 *ms*). Providing this level of assurance for control systems which involve scheduling tasks for such operations as *process sampling*, *regulation* and *output* within a distributed environment is challenging. Clearly, any analysis performed on such systems (to predict performance bounds) must be capable of delivering both upper and lower bounds on response times for critical transactions.

This paper extends integrated scheduling analysis to allow the computation of both worst- and best-case transaction response times. The evaluation of best-case performance facilitates a more accurate calculation of task and message jitter (variability in response time) than has hitherto been possible. This in turn allows computed bounds to be tightened on transaction responses which results in a less pessimistic evaluation of response times. To enable transaction bounds to be computed it is necessary to evaluate both worst- and best-

case task and message response times; expressions have been derived for the particular conditions of:

— Software task scheduling on processors using the fixed-priority preemptive policy

— Message transmission on Controller Area Networks (ISO, 1993) assuming fixed priority assignment and non-preemptive scheduling

The remainder of this paper is organised as follows. Section 2 introduces the computational model assumed for tasks, messages and their system compositions. We also make explicit the rôle played by jitter in distributed real-time systems and its treatment in our model. Section 3 reviews the established analysis of task and message scheduling for computing worst-case bounds on response times and §4 presents a complementary analysis of best-case bounds. Section 5 summarises the computational model and presents an algorithm to compute bounds on system transactions for diverse distributed real-time systems. Finally, §6 reviews the work, presents some conclusions and describes a programme of further work.

2. Computational Model and Assumptions

Distributed systems we consider are compositions of tasks scheduled on processors and messages scheduled on Controller Area Networks. Both tasks and messages are statically allocated to resources and, in order that they are amenable to a simple scheduling analysis, they are periodic or sporadic (have minimum inter-arrival times).

2.1. Tasks and Scheduling

The notation adopted to define task sets is as follows. Let $\mathcal{P}$ be a set of processors, and n be the total number of tasks in the system. Tasks are identified by an integer in the range $1 \ldots n$. The task set, $\mathcal{T}$, is defined as

$$\mathcal{T} = \left\{ (T_i, C_i^{\downarrow}, C_i^{\uparrow}, \pi_i, p_i) \mid 1 \leq i \leq n \right\}$$

where T_i is the period of task i, $C_i^{\downarrow}$ and $C_i^{\uparrow}$ are respectively the best- and worst-case computation times of the task, π_i is its priority and $p_i \in \mathcal{P}$ is the processor to which task i is allocated. The superscripts $^{\downarrow}$ and $^{\uparrow}$ denote respectively lower and upper values. The set $\mathcal{T}_p$ of tasks allocated to the processor $p \in \mathcal{P}$ denotes the set $\{\tau_i \in \mathcal{T} \mid p_i = p\}$

Tasks are scheduled according to the fixed-priority preemptive policy in which priorities are totally ordered. A task *arrives* infinitely often at the start of its period at which point it is logically ready to be scheduled for execution. However, a task may suffer a delay following arrival before it is *released* and can be entered into a priority ordered run queue (see Fidge (1998) for a comprehensive review of the semantics of task scheduling). This delay, called *release jitter*, is the maximal difference between arrival time and release time; it results from variations in response times of tasks and messages which precede the given

task in an end-to-end transaction (jitter is discussed in greater detail in section 2.4). Our model allows tasks to be triggered by clock, interrupt or by the arrival of a message handled by the real-time kernel:

Periodic release by clock — These tasks have periods which are integer multiples of the kernel clock tick. Thus, the arrival of a task will coincide with the servicing of a real-time clock at which point it will be entered into a priority ordered run queue and executed when it becomes the highest priority task. Such tasks are released on arrival at the beginning of their periods.

Periodic interrupts — An interrupt may be raised by an external device or perhaps the arrival of a network message. The processing of interrupts is assumed to be hardware prioritised and all such tasks have higher priorities than periodic clock tasks. The interrupt task is released immediately since it must possess the highest priority but it may be preempted by the arrival of higher priority interrupt task(s).

Periodic messages handled by the kernel — A task waiting on a kernel message (sent by another task on the same processor) arrives at the beginning of its period and is released some time later following the jitter inherited from predecessor tasks/messages.

This assumed scheduling and interrupt behaviour matches that supported by popular real-time kernels and processors (e.g., VxWorks (Wind River Systems, 1993) and 68000 CPU) and provides timely execution of sporadic, high priority, interrupt-driven tasks and restricted periodic behaviour for other tasks.

We exclude the possibility that the execution of multiple instances of a task can be underway, i.e., tasks must complete before their next arrival. When a task is preempted by the arrival of a higher priority task, the current task is switched for the higher priority task; the real-time kernel performs a context switch within the interval $[C_{cs}^{\downarrow}, C_{cs}^{\uparrow}]$. Thus, the context switching overhead for a single preemption by a higher priority task is $2\,C_{cs}^{\uparrow}$ in the worst case; i.e., one context switch to start the higher priority task and another on its completion. The restricted semantics of task execution adopted here is as follows:

— Tasks progress through the following phases each time they execute:

1. Initial communication (reception),
2. Computation period (no communication),
3. Final communication (transmission).

— A task arrives (and subsequently may be scheduled for execution) following the reception of its initial communication or is released by a clock if it is the initial task of a transaction.

— The last task in a transaction has no final communication phase.

2.2. Inter-Task Communication

Two mechanisms for inter-task communication are considered - tasks on the same processor communicate with the support of kernel services and tasks on different processors communicate via one or more Controller Area Networks (CAN) (ISO, 1993). Let $\mathcal{N}$ be a set of networks, and m be the total number of messages in the system. Messages are identified by an integer in the range $1 \ldots m$. The message set, $\mathcal{M}$, is defined as

$$\mathcal{M} = \{(T_i, S_i, \pi_i, q_i) \mid 1 \leq i \leq m\}$$

where T_i is the period of message i, S_i is its length, π_i is its priority and $q_i \in \mathcal{N}$ is the network to which message i is allocated. The set $\mathcal{M}_q$ of messages allocated to the network $q \in \mathcal{N}$ denotes the set $\{\mu_i \in \mathcal{M} \mid q_i = q\}$. The period of a message is inherited from the task which transmits it. Tasks may communicate with each other in a restricted way by asynchronous message passing. Broadcast communication semantics are assumed: tasks cannot be blocked when they transmit a message and more than one task may receive the same message.

2.3. Precedence Constraints and Graphs

If tasks do communicate they are said to be *precedence constrained* since a task is blocked until it receives a message from a task which precedes it. Currently, a task can have only one direct predecessor task (or none if it begins an end-to-end system response) and may have one or more successor tasks (or none if it terminates an end-to-end system response). Future extension of these semantics will admit the possibility of multiple direct predecessor tasks with AND or OR conditions.

A system of precedence constrained tasks is conveniently expressed in the form of a directed graph. The *Precedence Graph* is an acyclic directed graph $G = (V, \mathcal{E})$ where the set of vertices $V = \mathcal{T} \cup \mathcal{M}$ is composed of tasks and messages, and the set of edges, $\mathcal{E}$, giving the precedence constraints between them, is defined:

$$\mathcal{E} = \{(v_1, v_2) \mid v_1 \in V, v_2 \in V\}$$

It is required that $\#\{v \mid (v, v') \in \mathcal{E}\} \leq 1$ for all $v' \in V$. Currently, tasks may have only one immediate predecessor and the form of the precedence graph is restricted to that of a *tree*. Each sub-graph is a continuous chain of precedence related tasks which represents an end-to-end transaction of the system. Vertices in sub-graphs may share resources (processors, networks) with vertices in other sub-graphs. Figure 1 illustrates a precedence graph for a small system comprising two processors, 7 tasks and two messages transmitted on a single network. The accompanying key defines the graphical notation adopted to represent tasks, messages, processors, networks, constraints and transaction terminals. The large rectangular regions represent processors/networks on which are scheduled the tasks/messages they enclose. The control system behaves as follows: periodically, a process variable is measured at a remote sensor (task *sensor*) and the value is transmitted over

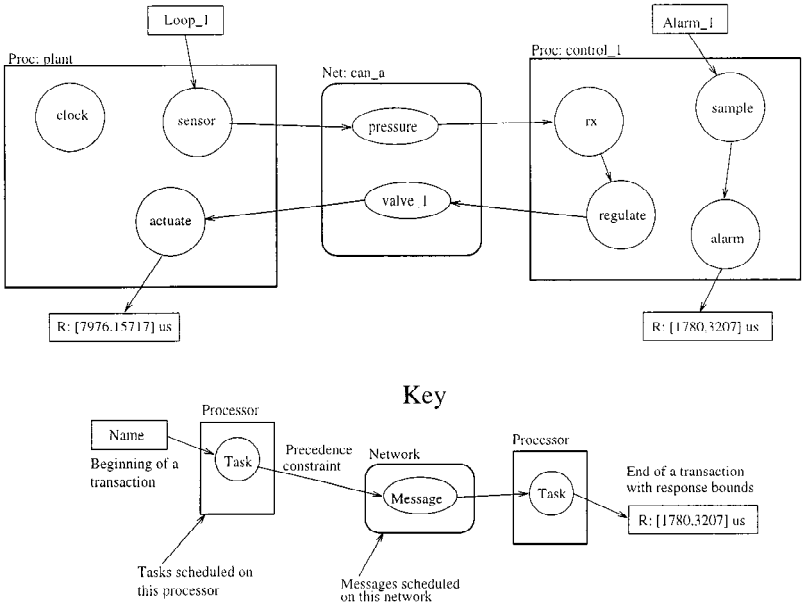

Figure 1. A precedence graph.

a network to a controller (task *regulate*). The controller computes a new control signal which is transmitted to an actuator (task *actuate*) to maintain desired behaviour of a process plant. The control processor also executes the task *sample* which polls an emergency signal and the task *alarm* which implements an emergency procedure. The messages *valve_1* and *pressure* are transmitted on the network *can_a* to communicate sensor and control data.

Tasks and messages constituting an end-to-end transaction have equally long periods but are shifted relative to one another. We associate hard deadlines with each end-to-end response; transactions must complete within their deadlines each time they execute. Let $G = (V, \mathcal{E})$ be a precedence graph. A full path $p = v_0, v_1, \ldots, v_n$ is a sequence of vertices such that v_0 is an initial vertex and v_n is a terminal vertex and $(v_i, v_{i+1}) \in \mathcal{E}$ for $0 \leq i < n$. A transaction $t = (p, D^{\downarrow}, D^{\uparrow})$ is a tuple where p is a full path and $D^{\downarrow}$, $D^{\uparrow}$ give the bounds on the completion time. The vertex $v \in V$ is an initial vertex if $\#\{v' \in V \mid (v', v) \in \mathcal{E}\} = 0$, and $v \in V$ is a terminal vertex if $\#\{v' \in V \mid (v, v') \in \mathcal{E}\} = 0$. A deadline is measured from the beginning of the period of the initial task (its arrival) to the completion of the computation of the last task. There are two end-to-end transactions in the example system of Figure 1, namely:

Loop_1 — The control loop—sample point to actuator change
Alarm_1 — The alarm system—alarm signal to alarm activation

2.4. Response Times and Jitter

As a result of interference, the response times of tasks will vary from one period to the next as will the response time of messages sharing a single network. In periodic distributed systems, transactions composed of precedence constrained tasks and messages will begin periodically. However, messages and tasks other than the first in a transaction will suffer variation in release time since they will inherit variations in response time (jitter) from predecessor tasks and messages. This range in response times will be bounded by the accumulated maximum interference suffered by tasks or messages on each resource (Fidge, 1998). Since the presence of jitter lengthens the worst-case response times of lower priority tasks and messages, it is desirable that jitter is not computed pessimistically. Three approaches to distributed real-time systems scheduling have emerged which differ in their treatment of jitter, namely:

Full Period Per Resource: — A full period may be reserved for the execution of each portion of a transaction on each resource (Sha and Sathaye, 1993). This technique solves the problem of accumulating jitter by simply delaying responses at each resource until the beginning of a new period. Data must be buffered on arrival at a resource and processed at the beginning of the next period. This allows processor and network scheduling to be analysed independently but at the expense of what may be a considerable increase in end-to-end transaction response time. The variation in completion time (completion jitter) of each transaction will be reduced by this method since jitter of a transaction will be introduced only on the last resource.

Task Offsets: — The execution of a transaction at each processor by a sequence of precedence constrained tasks may be achieved by introducing time offsets for each task after the first (Audsley and Burns,1998; Bate and Burns, 1997). Thus, the n^{th} instance of a task i arrives at a time $n\,T_i + O_i$ where O_i is the offset for that task. This avoids the accumulation of jitter from one resource to the next for distributed transactions if task offset values are chosen such that they are greater than worst-case response times of the event triggering the task. The use of offsets can increase utilisation of resources since task arrival times can be managed to reduce interference. However, the analysis is complex and tasks with co-prime periods cannot benefit from the approach (Bate and Burns, 1998). Like the "Full Period Per Resource" technique, kernel support and a global clock are required to implement offsets.

Inherited Jitter: — Perhaps the simplest approach that may be adopted is to schedule a task or message to arrive as soon as its predecessor task/message has completed. The major disadvantage of this technique is that jitter accumulates from one resource to the next in the course of executing a transaction and the completion time of a transaction may exhibit intolerable jitter. However, the approach will guarantee the shortest possible transaction times since transactions are not delayed at resources and is the easiest to implement in practice - it requires no special real-time kernel features or global clock. The analysis described in this paper assumes the "Inherited Jitter" model; this is now described in further detail.

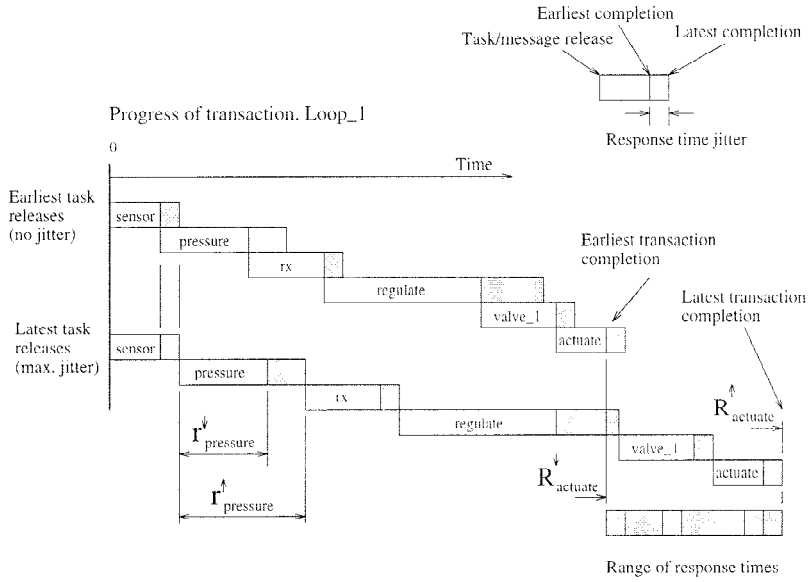

Figure 2. Jitter and precedence constrained tasks.

Figure 2 illustrates the progress of the transaction *Loop_1* illustrated in Figure 1. Each task or message of a transaction will have a variable response time since it may suffer preemption by higher priority tasks/messages. This variable response time of each stage in a transaction contributes an element of jitter towards the jitter in the complete transaction. We shall distinguish between *local* and *global* task/message response times. Local response times (denoted by r_i for task or message i) are simply the response times of tasks or messages measured from the local arrival times at each resource (at the beginning of a period). The global response time R_i of a task or message i in a transaction is measured from the beginning of the complete transaction. Thus, the global best- and worst-case response times at any point are:

$$R_i^{\downarrow} = R_{i-1}^{\downarrow} + r_i^{\downarrow} \quad \text{and} \quad R_i^{\uparrow} = R_{i-1}^{\uparrow} + r_i^{\uparrow}$$

where $r_i^{\downarrow}$ and $r_i^{\uparrow}$ are respectively the best- and worst-case *local* response times of the task or message, R_{i-1} is the global response time of the predecessor task or message and $R_0 = 0$.

2.5. Computation of Jitter

It is usually assumed in the analysis of distributed responses (Klein, Lohoczky, and Rajkumar, 1994; Larsson, 1996; Tindell and Clark, 1994; Tindell and Hanson, 1995) that the release jitter of a task or message is the worst-case response time of the task or message which preceded it, i.e., that the best-case response time of a predecessor task or message

is assumed to be arbitrarily small (zero). This simple evaluation of jitter may therefore be expressed:

$$J_i = R_{i-1}^{\uparrow} \quad \text{or} \quad J_i = \sum_{j \in \{1 \ldots i-1\}} r_j^{\uparrow}$$

Such an assumption is unlikely to be the case for any practical system because the minimum response time of a message is at least its data transfer time and the minimum response time of a task is at least its minimum computation time. The approximate treatment of jitter can lead to significant error in the calculation of lower priority transaction bounds since the jitter inherited by a task or message affects the computation of response times of lower priority tasks and messages. As will be demonstrated in §3, increasing jitter tends to widen the response time bounds of lower priority tasks, i.e., the best-case response is reduced and the worst-case response is increased. It is important that accurate estimates of jitter are used in computing the response times at each resource since jitter may increase rapidly with each additional precedence step in a transaction. This factor has been shown by simulation to be an important contributor to the pessimistic calculation of response times (Bate and Burns, 1998). The simple calculation of jitter will often result in the pessimistic evaluation of interference to lower priority tasks and therefore erroneous worst-case response times. The release jitter J_i of a task or message i (the maximum variation of its release time) is more accurately computed by:

$$J_i = R_{i-1}^{\uparrow} - R_{i-1}^{\downarrow} \tag{1}$$

or may be expressed as the sum of the differences between local best- and worst-case response times prior to that task/message:

$$J_i = \sum_{j \in \{1 \ldots i-1\}} (r_j^{\uparrow} - r_j^{\downarrow}) \tag{2}$$

where the index j ranges from the first task in the transaction to the task/message immediately preceding i. We compute both upper and lower bounds on response times and adopt the more accurate evaluation of jitter in the work reported here. The values of Jitter and global response times are *boundary conditions* imposed on the scheduling analysis on different processor/network resources. Systems which are likely to benefit from the more accurate calculation of jitter are those in which high priority tasks/messages suffer significant jitter—sufficient to contribute towards the preemption of lower priority tasks/messages. Since jitter in high priority tasks/messages potentially widens the response time intervals of lower priority tasks/messages, it can significantly increase jitter in the system as a whole.

3. Worst-Case Analysis

The analyses of worst-case fixed priority pre-emptive scheduling of tasks and worst-case message scheduling on a CAN are well known; we will simply state the principal results of previous analyses.

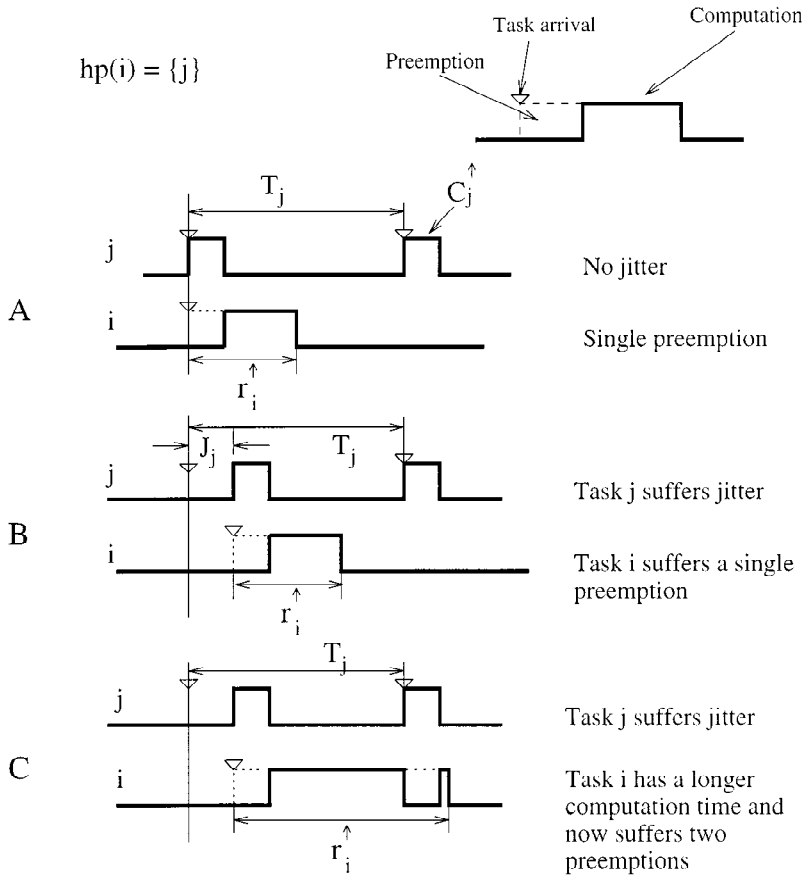

Figure 3. The calculation of worst-case task response time.

3.1. Computing Worst-Case Task Response Times

The scheduling analysis used to derive worst-case response times of fixed priority periodic tasks preemptively scheduled is described by Audsley (Audsley et al., 1993) and others, (Burns, 1991; Warren, 1991; Klein et al., 1993; Klein et al., 1994). The worst-case response time of a task is computed assuming that it is released at the same instant as all higher priority tasks—this is known as the "critical instant". Figure 3 illustrates the preemption suffered by a task i from a higher priority task j. If the worst-case computation time $C_i^{\uparrow}$ is short, task i may only suffer a single preemption from task j—as illustrated in Figure 3A. If task j suffers release jitter, depicted in Figure 3B, this reduces the time available for task i to complete without additional preemptions. Figure 3C shows the situation where the computation time for task i is longer and it suffers 2 preemptions as a direct result of jitter of task j. The local, worst-case, response time $r_i^{\uparrow}$ of a task i which is subject to interference

from higher priority tasks in the set $hp(i)$ may be expressed in the following iterative form:

$$r_i^{\uparrow *} = C_i^{\uparrow} + \sum_{j \in hp(i)} \left\lceil \frac{r_i^{\uparrow} + J_j}{T_j} \right\rceil C_j^{\uparrow}$$

where the newly computed response time, $r_i^{\uparrow *}$, replaces $r_i^{\uparrow}$ on each iteration, T_j and J_j are respectively the Period and Jitter of task j. Note that the expression must be iterated until convergence ($r_i^{\uparrow *} = r_i^{\uparrow}$) or the task is deemed unschedulable ($r_i^{\uparrow} \geq T_i$). An initial response time, $r_i^{\uparrow} = C_i$, may be assumed.

Finally, it is necessary to account for the time spent in context switching between the execution of different tasks. Let $C_{cs}^{\uparrow}$ be the worst-case time delay in performing a context switch. Each time a task is preempted by a higher priority task, two context switches may occur—the first to suspend the current task and a second to resume it following the preemption. The response time of a task also includes initial and final context switch times. Thus, the response time expression may be modified as follows to account for context switching (Burns and Wellings, 1995):

$$r_i^{\uparrow *} = C_i^{\uparrow} + 2\,C_{cs}^{\uparrow} + \sum_{j \in hp(i)} \left\lceil \frac{r_i^{\uparrow} + J_j}{T_j} \right\rceil (C_j^{\uparrow} + 2\,C_{cs}^{\uparrow}) \tag{3}$$

Some initial or final context switching delays may be omitted from equation 3. A task makes available a message it wishes to transmit to a communication controller just *before* the task terminates. Thus, the final context switch is not part of the response time of a task if it transmits a network message.

Since, task i is itself subject to jitter, the worst-case *global* response time is computed as the sum of the worst-case response time and the release jitter:

$$R_i^{\uparrow} = r_i^{\uparrow} + R_{i-1}^{\uparrow} + J_i \tag{4}$$

where $R_{i-1}^{\uparrow}$ is the worst-case global response time of the immediate predecessor task or message to task i on this transaction.

3.2. CAN Messages—Worst-Case Analysis

It is assumed that communication between tasks scheduled on separate processors is implemented using one or more Controller Area Networks. It has been shown (Tindell, Burns, and Wellings, 1995; Baba and Powner, 1995; Tindell and Burns, 1994) that the scheduling of periodic message sets on a CAN may be analysed to yield worst-case response times for all messages using much the same algorithm as for computing worst-case task response times. The analysis accounts for the time spent by a message waiting for access to the bus and its transmission time. The queueing time for bus access is composed of a), possible delay awaiting the transmission of the longest message already in progress (CAN messages are non-preemptible) and b), delay waiting for messages of higher priority to be transmitted. Using the standard CAN protocol (11-bit ID), the worst-case transmission time, $t_m^{\uparrow}$, for

message m may be computed as follows:

$$t_m^{\uparrow} = \left(8\,S_m + 47 + \left\lfloor \frac{34 + 8\,S_m}{5} \right\rfloor\right)\tau_{bit} \tag{5}$$

where S_m is the data length (0...8 bytes) and τ_{bit} is the time required to transmit a bit. CAN messages contain 47 bits of overhead. All data bits and 34 of the overhead bits are subject to bit stuffing with a 5-bit width. Thus, the term $\lfloor \frac{34+8\,S_m}{5} \rfloor$ expresses the maximal number of stuff bits inserted in a message. The worst-case queue delay time, $w_m^{\uparrow}$, a message m suffers waiting for higher priority messages to be transmitted may be expressed in the following iterative form:

$$w_m^{\uparrow *} = B_m + \sum_{j \in hp(m)} \left\lceil \frac{w_m^{\uparrow} + J_j + \tau_{bit}}{T_j} \right\rceil t_j^{\uparrow} \tag{6}$$

where B_m is the blocking time or time spent waiting for a lower priority message to complete its transmission and J_j and T_j are respectively the release jitter and period of message j. The second term in the equation is the worst-case delay suffered by message m waiting for all higher priority messages in the set $hp(m)$ to be transmitted assuming a common release instant. As with the evaluation of worst-case task execution times, the queueing time must be computed by iteration where the term $w_m^{\uparrow *}$ is a newly computed queueing time which replaces $w_m^{\uparrow}$ on each iteration. An initial queueing time of $w_m^{\uparrow} = B_m$ may be assumed and convergence is achieved when $w_m^{\uparrow *} = w_m^{\uparrow}$. The blocking time is simply the time required to complete the transmission of the longest lower priority message:

$$B_m = max_{k \in lp(m)}(t_k^{\uparrow})$$

where $lp(m)$ is the set of messages of lower priority than message m. The local worst-case response time of message m is the sum of the queueing and transmission times:

$$r_m^{\uparrow} = w_m^{\uparrow} + t_m^{\uparrow} \tag{7}$$

and the Global response time is expressed as the sum of the local response time, the response time of a predecessor task and any Jitter:

$$R_m^{\uparrow} = r_m^{\uparrow} + R_{m-1}^{\uparrow} + J_m \tag{8}$$

where $R_{m-1}^{\uparrow}$ is the worst-case global response time of the immediate predecessor task to message m on this transaction.

4. Best-Case Analysis

4.1. Computing Best-Case Response Times—Tasks

Minimal response times for tasks are associated with minimal (best-case) computation times, $C^{\downarrow}$. Consider the interactions between four independent tasks illustrated in Figure 4

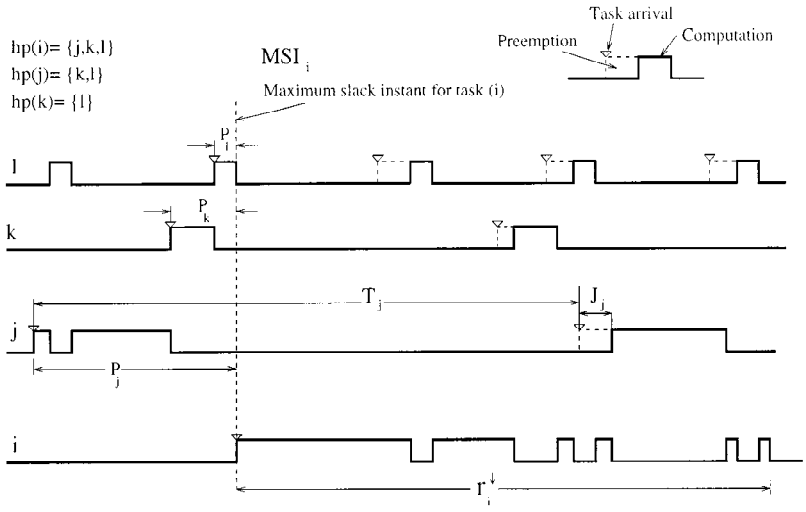

Figure 4. Finding the best-case response time of task i.

in which low priority task i suffers interference from higher priority tasks $\{j, k, l\}$, each of which is subject to release jitter. In computing the best-case response time of task i, we shall construct the situation which imposes the least interference from higher priority tasks; thus avoiding the "critical instant" (Audsley et al., 1993) or point of common release. This instant of minimal interference from higher priority tasks is termed the *Maximum Slack Instant*.

Maximum Slack Instant (MSI)

The *maximum slack instant* for task i, MSI_i, is a point in time at which *all* higher priority tasks begin their longest interval between the completion of one execution and the arrival of the next. The longest period of uninterrupted access is available to a task i following MSI_i. Clearly, to offer the longest uniterrrupted period for task i, each higher priority task should have suffered no jitter on its last release and should be delayed for its next execution by its maximum jitter.

An approach taken by Gutiérrez (Gutiérrez et al., 1998) in calculating minimal response times is simply to assume that all tasks of higher priority have just completed their execution; thus, their next release is $T_j + J_j - C_j$ later. This method has the advantage of simplicity but unfortunately results in an optimistic determination of interference since only one task can finish exactly at the MSI; other tasks will complete their computation at earlier times, as indicated for tasks $\{j, k\}$ in Figure 4. We are free to choose any task phasing which minimises interruptions to lower priority tasks during their response times since the periodic behaviour may start at any time; we shall assume that higher priority tasks finish latest before the MSI. In the example depicted in Figure 4, task l has the highest priority and finishes at MSI_i and tasks k and j finish at earlier times since they have lower priorities.

LEMMA 1 (MINIMAL PREEMPTED EXECUTION TIME PRIOR TO A MAXIMUM SLACK INSTANT) *Let P_j represent the interval between the arrival of a task j and a MSI during which task j is executed to completion. For a set of independent higher priority tasks (the set $hp(j)$), P_j may be computed iteratively by:*

$$P_j^* = C_j^{\downarrow} + \sum_{i \in hp(j)} \left\lceil \frac{P_j}{T_i} \right\rceil C_i^{\downarrow}$$

where P_j^* replaces P_j on each iteration.

Proof: Calculating the preemption a task suffers before a maximum slack instant is similar to the calculation of worst-case response times. We proceed using the approach suggested by Joseph and Pandya (Joseph and Pandya, 1986) who applied an interval arithmetic technique to compute worst-case task response times. Consider task j in Figure 4 which suffers interference from the two higher priority tasks k and l. The preemption time for task j, P_j, is minimised by summing the minimum computation times of higher priority tasks:

$$P_j = C_j^{\downarrow} + \sum_{i \in hp(j)} C_i^{\downarrow} \tag{9}$$

However, multiple preemptions from higher priority tasks during P are possible and this naïve approach is likely to be optimistic in practice. Consider task j which suffers interference prior to the MSI from both tasks k and l. Starting at the MSI and working in reverse, the maximal number of preemptions from task k is:

$$\left\lceil \frac{P_j}{T_k} \right\rceil$$

and the interference time from these preemptions is:

$$\left\lceil \frac{P_j}{T_k} \right\rceil C_k^{\downarrow}$$

More generally, the interference suffered prior to the MSI from all higher priority tasks is:

$$\sum_{i \in hp(j)} \left\lceil \frac{P_j}{T_i} \right\rceil C_i^{\downarrow}$$

P_j is the sum of the computation time of the task and its total interference, thus:

$$P_j = C_j^{\downarrow} + \sum_{i \in hp(j)} \left\lceil \frac{P_j}{T_i} \right\rceil C_i^{\downarrow}$$

It is necessary to solve for P_j iteratively in a way analogous to the worst-case analysis but "in reverse", i.e., P_j may be derived by iterating the formula:

$$P_j^* = C_j^{\downarrow} + \sum_{i \in hp(j)} \left\lceil \frac{P_j}{T_i} \right\rceil C_i^{\downarrow}$$

An initial value of $P_j = C_j^{\downarrow}$ may be assumed. Note that this expression degenerates to the approximate value of equation 9 if multiple preemptions by higher priority tasks are not possible. ■

Context switch times are included in this analysis in much the same way that they are for worst-case response times. Thus, the P interval including minimum context switch times is:

$$P_j^* = C_j^{\downarrow} + \sum_{i \in hp(j)} \left\lceil \frac{P_j}{T_i} \right\rceil (C_i^{\downarrow} + F_i \, C_{cs}^{\downarrow}) \tag{10}$$

where $C_{cs}^{\downarrow}$ is the lower bound on the context switch time and F_i is defined by:

$$F_i = \begin{cases} 2 & \text{when } hp(i) = \{\} \\ 0 & \text{when } hp(i) \neq \{\} \end{cases} \tag{11}$$

The final context switch for task j is excluded from equation 10 since it completes its computation as the next higher priority task is released. In the best-case, the initial context switch is also excluded because task j can be released at the instant a higher priority task completes its computation. The minimal number of context switches, as a result of preemptions of task j, occurs when the release times of all tasks (other than the highest priority task) coincide with completion times or occur during the computation of the highest priority task. Thus, the only preemption context switches included in equation 10 correspond to the highest priority task.

Remark. We have chosen an ordering of tasks which causes their completion prior to the MSI according to priority, with the highest priority tasks completing closest to the MSI. Other orderings are possible and could lead to less preemption of lower priority tasks. Thus, a particular ordering may be optimal for one value of C_i but not for another. Figure 5 illustrates the variation of the best-case response time with respect to computation time for the lowest priority task in an arbitrary system of three tasks. The two possible orderings of the two higher priority tasks prior to the MSI are considered. In the long term, cumulative interference of the two task is independent of starting order but there are short-term variations which mean that some orderings are optimal (produce the minimal best-case response times for lower priority tasks).

In computing $r_i^{\downarrow}$ for the example in Figure 4, there are only 6 possible orderings of the higher priority tasks $\{j, k, l\}$ but for larger sets of tasks the number of possible orderings could be numerous. For task i there are $!\#hp(i)$ orderings to consider. Potentially, a search for the task ordering which offers minimal blocking is computationally expensive. However, some task orderings are not permitted if tasks are precedence constrained. Clearly, the task/message invocation order prior to MSI must respect any precedence constraints between tasks and between tasks and messages.

THEOREM 1 (LEAST LOWER BOUND ON TASK RESPONSE TIMES) *The least lower bound on the response time of task i which suffers interference from a set of higher priority*

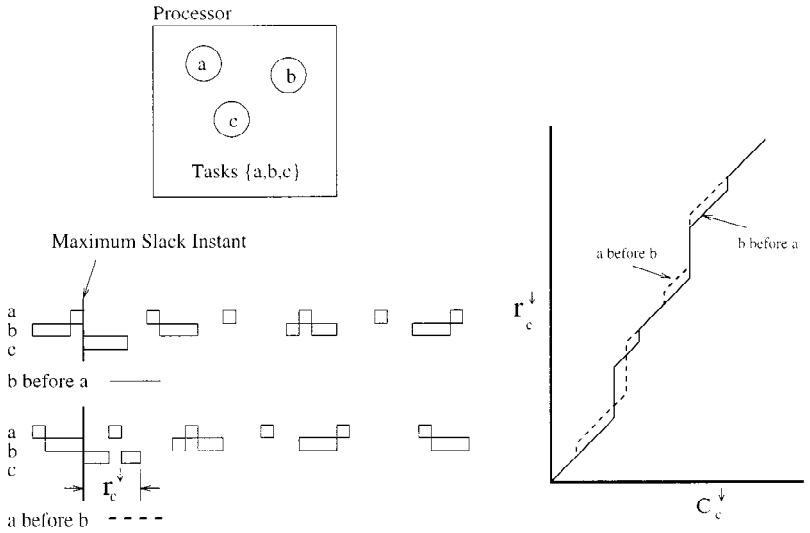

Figure 5. Response prior to MSI as a function of task ordering.

tasks $hp(i)$ is:

$$C_i^{\downarrow} + \sum_{j \in hp(i)} \left\lceil \frac{r_i^{\downarrow} - J_j + P_j}{T_j} - 1 \right\rceil C_j^{\downarrow}$$

where $r_i^{\downarrow}$ is the minimal response time of task i, T_j, J_j and $C_j^{\downarrow}$ are respectively the period, jitter and minimum computation times of the higher priority task j and P_i is the interval between the last release time of task i and the maximum slack instant, from Lemma 1.

Proof: Consider the situation illustrated in Figure 4 where the low priority task i suffers a single interference from the higher priority task j. To suffer no preemptions, the local response time of the lower priority task, r_i, is bounded by:

$$r_i^{\downarrow} = \left(0, T_j + J_j - P_j\right]$$

where $(t_1, t_2]$ is an open left interval from t_1 to t_2 excluding t_1 and including t_2. For a single preemption only:

$$r_i^{\downarrow} = \left(T_j + J_j - P_j, 2\,T_j + J_j - P_j\right]$$

In the general case, the response time interval for n preemptions only is:

$$r_i^{\downarrow} = \left(n\,T_j + J_j - P_j, (n+1)\,T_j + J_j - P_j\right]$$

This interference suffered by task i during its response is of course equal to $n\,C_j$ and the

response time $r_i^{\downarrow} = n\,C_j + C_i$. The minimal response time is bounded thus:

$$n\,T_j + J_j - P_j < r_i^{\downarrow} \leq (n+1)\,T_j + J_j - P_j$$

The inequality can be rearranged as follows

$$n - 1 < \frac{r_i^{\downarrow} - J_j + P_j}{T_j} - 1 \leq n$$

From the definition of the ceiling ($x \leq \lceil x \rceil < x + 1$) function, it can be shown that as $\lceil x \rceil = n$ then $n - 1 < \lceil x \rceil \leq n$. Thus, we can express the minimal number of preemptions suffered by task i as:

$$\left\lceil \frac{r_i^{\downarrow} - J_j + P_j}{T_j} - 1 \right\rceil$$

The total interference time associated with preemptions by all higher tasks in $hp(i)$ is:

$$\sum_{j \in hp(i)} \left\lceil \frac{r_i^{\downarrow} - J_j + P_j}{T_j} - 1 \right\rceil C_j^{\downarrow}$$

The minimal response time can thus be expressed in iterative form as the sum of the computation and interference times:

$$r_i^{\downarrow} = C_i^{\downarrow} + \sum_{j \in hp(i)} \left\lceil \frac{r_i^{\downarrow} - J_j + P_j}{T_j} - 1 \right\rceil C_j^{\downarrow}$$

■

The best-case response time is computed by iteration using the same technique as that adopted in the analysis of the worst-case:

$$r_i^{\downarrow *} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{r_i^{\downarrow} - J_j + P_j}{T_j} - 1 \right\rceil C_j^{\downarrow}$$

Where $r_i^{\downarrow *}$ replaces $r_i^{\downarrow}$ on each iteration assuming an initial value for $r_i^{\downarrow} = C_i^{\downarrow}$. This proceeds until convergence ($r_i^{\downarrow *} = r_i^{\downarrow}$) or the schedule is infeasible if $r_i^{\downarrow} > T_i$. Finally, the time penalties of context switching are included as follows:

$$r_i^{\downarrow *} = C_i^{\downarrow} + \sum_{j \in hp(i)} \left\lceil \frac{r_i^{\downarrow} - J_j + P_j}{T_j} - 1 \right\rceil (C_j^{\downarrow} + F_j\,C_{cs}^{\downarrow}) \tag{12}$$

where $C_{cs}^{\downarrow}$ is the best-case context switch time and F is defined by equation 11. The initial context switch to start task i is included in the computation of P_1 and the final context switch may, in the best-case, coincide with the release of a higher priority task; thus, both initial and final context switches are excluded from equation 12. As with the computation

of minimal preempted execution times (equation 10), only context switches resulting from the execution of the highest priority task are included in computing minimal response times.

Unlike worst-case response times, best-case response times reduce with increasing jitter. Thus, the range of response times exhibited by a given task is *widened* by the presence of jitter in higher priority tasks. In the best case, tasks will suffer no release jitter and their global responses time may be expressed by:

$$R_i^{\downarrow} = R_{i-1}^{\downarrow} + r_i^{\downarrow} \tag{13}$$

where $R_{i-1}^{\downarrow}$ is the best-case response time of the predecessor task to task i.

4.2. Computing Best-Case Response Times—CAN Messages

In computing minimal message transfer times, $t^{\downarrow}$, we assume that no additional synchronisation bits are inserted. Thus, the minimal transfer time of message m assuming the standard (11-bit ID) frame protocol is simply:

$$t_m^{\downarrow} = (8\ S_m + 47)\ \tau_{bit} \tag{14}$$

Since our communications scheduling regime is non-preemptive, a message may be transmitted with zero queueing time regardless of the relative priority of the message. In addition, it may be assumed that in the best-case, messages are not blocked because other transmissions are underway. Thus, the minimum local response time, $r_m^{\downarrow}$, of a message m is its minimum transfer time alone:

$$r_m^{\downarrow} = t_m^{\downarrow} \tag{15}$$

This is true only if message m is schedulable on the network, i.e., there are time intervals between the transmission of higher priority messages. In the best case, messages will suffer no release jitter and their global responses time may be expressed by:

$$R_m^{\downarrow} = R_{m-1}^{\downarrow} + r_m^{\downarrow} \tag{16}$$

where $R_{m-1}^{\downarrow}$ is the best-case response time of the predecessor task to message m.

5. An Integrated Scheduling Algorithm

Computing worst-case execution times for end-to-end transactions essentially involves the two stages:

1. Compute local response times on all resources
2. Compute global response times and jitter on all resources

which are repeated until convergence is achieved. The second stage requires the traversal of precedence graphs to sum the delays along each end-to-end transaction. Because not

all information is available when computing response times on each resource, the process has to proceed by iteration. On each iteration, the precedence constraints are applied in sequence. For constraints involving kernel message passing, this entails making the starting delay of the destination task equal to the response time of the source task (kernel delays in message passing are included in task computation times). For constraints involving network support, the starting delay of the message is made equal to the response time of the source task and the starting delay of the destination task is made equal to the response time of the message. The bounded response times of each of a set of end-to-end requirements is computed using the following algorithm:

begin { *Distributed Transaction Bounds* }
  Initialise global responses - $R_0^{\downarrow} = R_0^{\uparrow} = 0$;
  Initialise Jitter, $J_o = 0$;
  do
    for each $p \in \mathcal{P}$ **do**
      for each $\tau \in \mathcal{T}_p$ **do**
        compute blocking prior to MSI, P_τ; (**eqn 10**)
        compute local task responses, $r_\tau^{\uparrow}$, $r_\tau^{\downarrow}$; (**eqns 3,12**)
    for each $n \in \mathcal{N}$ **do**
      for each $m \in \mathcal{M}_n$ **do**
        compute message transmission times,$t_m^{\uparrow}$, $t_m^{\downarrow}$; (**eqns 5,14**)
        compute message queueing times,$w_m^{\uparrow}$; (**eqns 6**)
        compute local message responses, $r_m^{\uparrow}$, $r_m^{\downarrow}$; (**eqns 7,15**)
    for each full path, $p = (v_0, v_1 \ldots v_n)$ **do**
      for each $v \in p$ **do**
        compute Global response boundary conditions, $R_v^{\downarrow}$, $R_v^{\uparrow}$; (**eqns 4,13,8,16**)
        compute jitter boundary condition, J_v; (**eqn 1 or 2**)
    test for convergence;
  until convergence **or** not schedulable
end

The convergence test is satisfied when all response times are unchanged from one iteration to the next. For many small-medium sized systems, between 2 and 4 iterations appear sufficient to obtain a converged solution.

A software tool, $\mathcal{X}rma$ (Henderson, 1998), has been developed to undertake the distributed response time analysis. The tool accepts system descriptions, displays system data, checks data integrity, analyses systems and computes bounds on end-to-end transactions. Results of the analysis can be presented to show precedence graphs and end-to-end transaction as compositions of precedence constrained tasks and messages. System descriptions are prepared in text files which include the following elements:

— Processor declaration;

— Task assignment on processors including task period and computation time bounds;

— Network declaration and bit rate;

— Message assignment on networks including message period and length;

— Precedence constraints between tasks and messages;

— End-to-end transactions with deadline bounds.

The toolkit facilitates the rapid evaluation of different system configurations to check for schedulability and that transaction responses are within their deadline bounds.

6. Conclusions and Further Work

In this paper we have addressed the problem of guaranteeing bounds on the duration of end-to-end transactions in distributed real-time systems in which tasks communicate by asynchronous message passing on one or more Controller Area Networks. Precise performance data is required to assure the behaviour of distributed control systems where periodic control actions are required with precise timing. In contrast to the computation of worst-case performance of scheduled real-time systems, the evaluation of best-case response times has hitherto received little attention. We present a novel performance analysis based on a more accurate computation of release jitter which yields both lower and upper bounds on transactions using a single scheduling algorithm. The more accurate treatment of jitter may also lower the predicted upper bound on transaction times compared to previous, more pessimistic, analyses. Our expressions for response time include context switching overheads. Usually context switching is either ignored or the delays are subsumed into task computation times. Our treatment of context switching is approximate—it ignores the dependency of context switch delays on the level of the context switch and other factors (Burns and Wellings, 1995). This approximate treatment might usefully be refined to improve the overall accuracy of the analysis. However, any improvement in this respect will rely on the availability of detailed timing information on kernel behaviour.

Our programme of further work includes the generalisation of precedence constraints to permit multiple immediate predecessors to tasks; this will allow the modelling of a wider range of systems. Also of interest is the analysis of systems which use network protocols other than CAN, the derivation of optimal task ordering prior to a MSI to yield the longest period of non-preemption and empirical validation of analytical results.

Acknowledgments

The authors gratefully acknowledge the help of the reviewers whose insightful comments significantly contributed to the quality of this paper.

References

Audsley, N. C., and Burns, A. 1998. On fixed priority scheduling, offsets and co-prime task periods. *Information Processing Letters* 67(2): 65–69.

Audsley, N. C., Burns, A., Richardson, M., Tindell, K., and Wellings, A. J. 1993. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal* 8(5): 284–292.

Baba, M. D., and Powner, E. T. 1995. Scheduling performance in distributed real-time control systems. *2nd Int. CAN In Automation Conference*, pp. 7-2–7-11.

Bate, I., and Burns, A. 1997. Schedulability analysis of fixed priority real-time systems with offsets. *Proceedings of Ninth Euromicro Workshop on Real-Time Systems.* Toledo, Spain, pp. 153–160.

Bate, I., and Burns, A. 1998. Investigation of the pessimism in distributed systems timing analysis. *Proceedings of Tenth Euromicro Workshop on Real-Time Systems*. Berlin, pp. 107–114.

Burns, A. 1991. Scheduling hard real-time systems: A review. *Software Engineering Journal* 6(3): 116–128.

Burns, A., and Wellings, A. J. 1995. Engineering a hard real-time system: From theory to practice. *Software—Practice and Experience* 25(7): 705–726.

Colnarič, M., Verber, D., Gumzej, R., and Halang, W. A. 1998. Implementation of hard real-time embedded control systems. *Real-Time Systems* 14: 293–310.

Fidge, C. J. 1998. Real-time schedulability tests for preemptive multitasking. *Real-Time Systems* 14: 61–93.

Gutiérrez, J. P., García, J. G., and Harbour, M. G. 1998. Best-case analysis for improving the worst-case schedulability test for distributed hard real-time systems. *Proceedings of Tenth Euromicro Workshop on Real-Time Systems*. Berlin, pp. 35–44.

Henderson, W. D. 1998. The Xrma Toolkit. Technical Report NPC-TRS-98-1, University of Northumbria, School of Computing and Mathematics.

ISO. 1993. 11898—Road Vehicles—Interchange of Digital Information—Controller Area Network (CAN) for high-speed communication. ISO, 1st edition.

Joseph, M., and Pandya, P. 1986. Finding response times in a real-time system. *The Computer Journal* 29(5): 390–395.

Klein, M. H., Lohoczky, J. P., and Rajkumar, R. 1994. Rate-monotonic analysis for real-time industrial computing. *Computer* 27(1): 24–33.

Klein, M. H., Ralya, T., Pollak, B., Obenza, R., and Harbour, M. G. 1993. *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*. Boston: Kluwer.

Krtolica, R., Özgüner, U., Chan, H., Göktaş, H., Winkelman, J., and Liubakka, M. 1994. Stability of linear feedback systems with random communication delays. *International Journal of Control* 59(4): 925–953.

Larsson, J. 1996. ScheduLite—a fixed priority scheduling analysis tool. Master's thesis, Department of Computer Systems—ASTEC, Uppsala University.

Özgüner, U. 1989. Problems in implementing distributed control. *Proceedings of the American Control Conference*. Pittsburgh, pp. 274–279.

Ray, A. 1988. Distributed data communication networks for real-time process control. *Chemical Engineering Communications* 65: 139–154.

Sha, L., and Sathaye, S. 1993. A systematic approach to designing distributed real-time systems. *Computer* 26(9): 68–78.

Tindell, K., and Burns, A. 1994. Guaranteeing message latencies on controller area network (CAN). *Proceedings 1st International CAN Conference*, pp. 2–11.

Tindell, K., Burns, A., and Wellings, A. J. 1995. Analysis of hard real-time communications. *Real-Time Systems* 9: 147–171.

Tindell, K., and Clark, J. 1994. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and Microprogramming* 40(2–3): 117–134.

Tindell, K., and Hansson, H. 1995. Real-time systems and fixed priority scheduling. Technical Report Department of Computer Systems, Uppsala University.

Warren, C. 1991. Rate monotonic scheduling. *IEEE Micro* 11(3): 34–38.

Wind River Systems. 1993. 'VxWorks Programmer's Guide 5.1'. Wind River Systems Inc.