\textbf{Xrma:} An holistic approach to performance prediction of distributed real-time CAN systems

William Henderson, David Kendall, Adrian Robson and Steven Bradley

Abstract

Rate Monotonic Analysis (RMA) is a standard technique for the analysis of task schedulability which has also successfully been applied to the schedulability analysis of CAN messages. The prediction of worst-case end-to-end response times in a distributed CAN application requires the integrated analysis of both tasks and messages - the so-called “Holistic” approach. The work discussed in this paper contributes to the practical application of an holistic approach to analysis by: a) providing a framework for describing distributed periodic systems as graphs of precedence constrained tasks and messages, b) developing a tool to automate the analysis and support systems design and, c) validating the approach by empirical means. We describe the analysis tool, \textit{Xrma}, and show how it supports the design and evaluation of distributed hard real-time systems. The analysis of an example distributed control system composed of multiple control/feedback loops is presented to illustrate how the tool is used to verify that critical end-to-end response times can be met by an implementation. To verify the analysis, timing measurements have been conducted on distributed control systems which use the VxWorks real-time kernel for task scheduling and CAN for inter-processor communication. The empirical results confirm that the analytical approach allows reliable bounds to be predicted for distributed responses in systems of practical complexity. However, it is shown that predicting tight bounds requires the elimination of pessimistic properties of distributed scheduling models.

1. Introduction

For economic or physical reasons many control systems are implemented as decentralised systems with control processors, actuators and sensors 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 [9]. This type of control infrastructure is found in aircraft, manufacturing plant, increasingly in automobiles 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 important in achieving the required control performance (rise time, overshoot, etc.) and in maintaining stability [8]. There are strict deadlines placed on closed loop response times which emerge from a control system stability analysis [7]. 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 so-called holistic scheduling analysis proposed by Tindell [11]. Essentially, the technique utilizes scheduling models to predict worst-case task execution times on processors and message queueing/transmission times on networks. Worst-case end-to-end system response times are computed by adding these individual delays on each resource. It is necessary in the overall analysis to account for the fact that the portions of a response that execute on processors or networks after the first are not necessarily initiated periodically, Klein [6].

Results of an holistic analysis may be pessimistic (possibly very pessimistic) - computed response times may be longer than any possible in a simulation or observed in empirical behaviour. The reason for this pessimism is that the direct application of scheduling analysis will assume that interference can occur between all tasks scheduled on each processor and between all messages scheduled on each net-
work - this may not be the case. If only one invocation of
an end-to-end response is underway at any one time, then
processors and networks are reused serially and some tasks
and messages cannot interfere.

The analytical inaccuracy resulting from pessimistic task
and message preemption is often tolerated because pre-
dicted “worst-case” responses are at least as long as any
exhibited by an implementation and might be regarded as
“safe” - at least the technique does not accept “bad” so-
lutions (system designs which cannot meet timing obliga-
tions). However, pessimistic performance predictions will
cause quite suitable implementations to be rejected and the
provision of needless CPU performance and network band-
width. Ideally, end-to-end responses should be computed
after taking into consideration pessimistic task and message
interference.

The availability of tools to support the analysis of sys-
tems is important since the numerical manipulation in-
volved in distributed scheduling is tedious and error prone.
Tools allow larger numbers of design alternatives to be in-
vestigated and provide additional benefits such as improved
management of the design process, data integrity checking
and automated presentation of results. A simple analytical
tool has been developed to support the analysis of single
processor and distributed real-time systems which is based
on the holistic scheduling approach.

In order to investigate the importance of pessimism in
scheduling models and improve our confidence in the anal-
ysis we have conducted measurements on distributed sys-
tems. We report here results of empirical work on sys-
tems implemented using the VxWorks real-time kernel for
task execution and CAN for inter-processor communica-
tion. The empirical results are compared with predictions
made using the scheduling analysis tool.

The remainder of this paper is organised as follows. In
§2 we present the semantics assumed for both task models
and inter-task message passing and in §3 describe the com-
position of end-to-end system responses as graphs of prece-
dence constrained tasks. §4 reviews the application of Rate
Monotonic Analysis to compute the performance of peri-
odic distributed systems and §5 describes the Armma tool
for automating the analysis. The approach is demonstrated
in §6 by deriving end-to-end response times for an exam-
ple distributed control system. The results of the analysis
are then compared with experimental measurements made
on the system. Finally, §7 presents the conclusions and de-
scribes further work.

2. The Semantics of System Models

The systems of inter-communicating tasks considered
here are constrained in their behaviour. In order to be
amenable to the Rate Monotonic approach, both tasks and
messages are periodic. Each task has associated with it a pe-
riod, a computation time, a jitter value and a blocking time.
The task period is a worst-case (minimum) inter-arrival in-
terval. The computation time is the time required to com-
plete the execution of a task assuming unrestricted access
to a processor. Jitter is a worst-case delay that a task may
suffer after the beginning of its period before it is available
for execution (placed in the run-queue) [1]. Blocking time
accounts for delays suffered by tasks when tasks of lower
priority access shared resources using the Priority Ceiling
Protocol [10].

Overheads due to preemption and implementation of the
scheduling strategy (context switching, etc) are evaluated
and included in the execution time of tasks. Since we are
considering distributed systems, tasks will communicate or
synchronise to meet end-to-end goals. Two mechanisms
for synchronous inter-task communication are considered
-tasks on the same processor communicate with the support
of kernel services and tasks on different processors commu-
nicate via a Controller Area Network (CAN) [5]. Messages
each have an associated period, a length and a jitter time.
The period of a message is inherited from the task which
transmits it. Thus, the late completion of a task will delay a
message it transmits. As with tasks, a message may inherit
a jitter value from hardware such as the network controller
or network software; this will represent the worst-case de-
lay suffered by the message. The blocking suffered by a
message is computed as part the scheduling analysis. The
inter-task communication semantics currently adopted are
as follows:

- Tasks exhibit the following phases each time they exe-
cute (unless they are end-to-end terminals):
  1. Initial communication (reception),
  2. Computation period (no communication),
  3. Final communication (transmission)
- A task is scheduled for execution following the recep-
tion of an initial communication or is released by a
clock if it begins an end-to-end response.
- A task can have only one direct predecessor task (or
  none if it begins an end-to-end system response).
- A task has one or more successor tasks (or none if it
  terminates an end-to-end system response).

Broadcast communication semantics are assumed. Thus,
tasks cannot be blocked when they transmit a message,
more than one task may receive the same message and tasks
block on waiting for an initial message reception. Future
extension of these semantics will admit the possibility of
multiple direct predecessor tasks with AND or OR condi-
tions. These restrictions may appear quite limiting. How-
ever, complex systems can be designed by decomposing
tasks, for example, intermediate communication may be modelled in this way.

3. Precedence Constraints and End-to-end responses

Tasks may communicate (synchronise) with each other in a restricted way by message passing. If tasks do communicate they are said to be precedence constrained in that a task is blocked until it receives a message from a task which precedes it. Timing obligations that a system must meet are expressed as end-to-end requirements; these are simply sequences of constraints between tasks.

$\mathcal{P}$ and $\mathcal{N}$ denote respectively sets of processors and networks in the system. $\mathcal{T}$ and $\mathcal{M}$ denote the sets of tasks and messages in the system. On each processor is allocated a set of tasks, $\mathcal{T}_p \subseteq \mathcal{T}$, for processor $p$ and on each network is allocated a set of messages, $\mathcal{M}_n \subseteq \mathcal{M}$, for network $n$. A system of precedence constrained tasks is conveniently expressed in the form of a directed graph. The **Precedence Graph** is an acyclic directed graph $G(\mathcal{T}, \mathcal{C})$ in which $\mathcal{T}$ is a set of tasks (vertices) and $\mathcal{C}$ a set of constraints (directed arcs). Currently, the form of the precedence graph is restricted to that of a tree; i.e., tasks may have only one immediate predecessor. **Figure 1** illustrates a simple precedence graph for a system comprising two processors, seven tasks and a single network. The following graphical notation has been used. Circular vertices represent tasks, rectangular vertices represent terminals of end-to-end responses, solid arcs represent precedence relationships between tasks and dotted arcs identify the start and end tasks of end-to-end responses. The large rectangular regions represent processors on which are scheduled the tasks they enclose. The graph is annotated with the names of processors, tasks, precedence constraints and end-to-end responses.

The control system behaves as follows: Periodically, a process variable is measured at a remote sensor and the value is transmitted over a network to a controller. The controller computes a new control signal which is transmitted to an actuator 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.

An end-to-end response represents a transaction performed by a system; it begins periodically or when the system is stimulated by the environment and ends when it makes a corresponding change in the environment. The end-to-end response is effected by the execution of a number of precedence-constrained tasks scheduled on one or more processors. We associate hard deadlines with each end-to-end response, i.e., responses must complete before their deadlines each time they are executed. There are two end-to-end responses in the example system, namely:

1. **Loop_1** - The control loop - sample point to actuator change
2. **Alarm_1** - The alarm system - alarm signal to alarm activation

4. Computing end-to-end response times

Computing worst-case execution times for end-to-end responses essentially involves the traversal of precedence graphs to sum the delays along each response. Tasks which follow the initial task of an end-to-end response are periodic in their average behaviour but will suffer variable delays in their starting times. This is because later tasks in a response will inherit the variable response times of earlier tasks. All inter-task messages will also exhibit this behaviour since responses cannot begin with a message and will inherit a variable delay from their transmitting task. Klein describes this process in his handbook [6] and draws attention to the influence deferred execution has on the scheduling of tasks and messages. For single processor systems, the task phasing which results in the worst-case scheduling point is where
all tasks are released together. However, if task execution is deferred because tasks are part of end-to-end responses, tasks of lower priority on the same processor may suffer additional delay.

Tindell[11] proposed a method of solving this problem by including inherited delay times in the computation of task and message response times. 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 (recall that 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 worst-case response times of each of a set of end-to-end requirements is computed using the following algorithm:

\begin{verbatim}
InitialiseForConvergenceTest;
InitialiseTaskAndMessagesDelays;
\emptyset Iterate to convergence
while (!converged && responses <= deadlines) {
  \emptyset Compute task and message response times
  for (proc= 1; proc <= n_procs; proc++) {
    ComputeTaskResponses(proc);
  }
  for (net= 1; net <= n_nets; net++) {
    ComputeMessageResponses(net);
  }
  \emptyset Inherit delays
  for (cons= 1; cons <= n_cons; cons++) {
    message_delay(cons.net,cons.mess) =
    task_response(cons.s_proc,cons.s_task)
    task_delay(cons.d_proc,cons.d_task) =
    message_response(cons.net,cons.mess)
  }
  check convergence of response times;
}
\end{verbatim}

The convergence test is satisfied when all response times are unchanged between iterations.

5. The $Xrma$ toolkit

A software tool, $Xrma$[3], has been developed to undertake the holistic analysis outlined above. The tool accepts system descriptions, displays system data, checks data integrity, analyses systems and computes end-to-end response times. Results of the analysis can be presented to show precedence graphs and end-to-end responses. Figure 2 provides a screen shot of the user interface. System descriptions are prepared in text files which include the following elements:

- Task assignment on processors including task period, computation, jitter and blocking times
- Message assignment on networks including message period and length
- Precedence constraints between tasks and messages
- End-to-end requirements

Systems are input to the tool using the simple format illustrated below.

\begin{verbatim}
PROC remote
TASK sample 100 10 50000 55 50000
TASK ...
PROC control
TASK rx 50 15 50000 47 0
TASK ...
NET net_a 500000
MESSAGE pressure 50000 2
MESSAGE ...
CON net1 remote sample net_a pressure control rx
CON ker1 control rx -- control regulator
CON ...
ETE Loop net1 ker1 10000
\end{verbatim}

The keyword PROC introduces the name of a processor and begins a block of task descriptions. TASK introduces a task including its identifier followed by the computation, jitter, period, blocking and deadline specified in $\mu$s. The deadline on the completion of a single task is usually not significant since it is part of a precedence constrained sequence requiring the execution of several tasks. NET introduces a CAN network name and speed (bits/s). This is followed by
a list of messages scheduled on that network. MESSAGE is followed by an identifier for that message, its period and length (bytes). Precedence constraints are introduced using CON followed by the identifier for that constraint and the source processor and task identifiers, network and message identifiers and lastly the destination processor and task identifiers. Thus, constraints are specified in terms of previously defined tasks and messages. For constraints between tasks on the same processor the network and message are both indicated by “-“. Finally, end-to-end responses are introduced using ETE and are expressed as sequences of constraints. End-to-end responses also have identifiers and, most importantly, deadlines.

6. An Example

This section illustrates the application of the Xrma tool to the analysis of an example control system to derive worst-case bounds on distributed responses. Results of the analysis are presented showing that bounds on worst-case response times can be narrowed by recognising that certain task and message interference is impossible and that scheduling models can be made less pessimistic. Finally, a complementary empirical study of the same control system is described. Empirical response times are compared with computed results in an attempt to validate the analytical approach.

6.1. System Description

The example in question is of a somewhat simplified distributed industrial control system comprising two control loops, as illustrated in Figure 3. The example system was chosen for its scalability - an increasing network load may be achieved simply by adding further control loops which share the network. Each control loop involves two processors which communicate using a shared CAN bus. Processors control_1 and control_2 execute the following tasks:

\[ T_{control_1} = T_{control_2} = \{rx, regulate, timer, sample, alarm, display\} \]

to support both a control behaviour and to implement a safety protection and display system. Input and output operations are performed remotely on the processors plant_1 and plant_2 which execute the tasks:

\[ T_{plant_1} = T_{plant_2} = \{timer, rx, sensor, actuate\} \]

A single network, can_a, transports four periodic messages:

\[ M_{can_a} = \{pressure, value_1, level, value_2\} \]

and operates at a speed of 72727 bits/s. The control loops execute periodically (every 50 ms). A pressure (or level) sensor is read by task sensor which transmits a value over the network in an 8-byte message. The message is received by network task rx making it available to the regulator task regulate which computes a new control value and then transmits it back to the plant processor. Task rx on the plant receives the message and makes the data available to task actuate which outputs it to the process plant. In addition to the control function, each control processor supports a user interface and alarm system. Task sample executes periodically (every 20 ms) and samples values from a command interface. Two further tasks, alarm and display, are subsequently executed.

This system is described for the purpose of input to the Xrma tool by the following file:

```plaintext
// Example system - 2 closed-loops
// Time units - microseconds

// Tasks on Processor - controller
PROC control_1
TASK clock 147 0 16666 0
TASK timer 362 0 20000 0
TASK sample 819 0 20000 0
TASK alarm 961 0 20000 0
TASK display 879 0 20000 0
TASK rx 157 0 50000 0
TASK regulate 1461 0 50000 0

PROC control_2
... similar to control_1

// Tasks on Processor - plant_1
PROC plant_1
TASK clock 147 0 16666 0
```

Figure 3. Layout of the Example System
TASK timer 362 0 50000 0
TASK rx 157 0 50000 0
TASK sensor 1181 0 50000 0
TASK actuate 1444 0 50000 0

PROC plant_2
... similar to plant_1

// The network
NET can_a 72727
MESSAGE pressure 50000 8
MESSAGE valve_1 50000 8
MESSAGE level 50000 8
MESSAGE valve_2 50000 8

// Constraints
CON C_n1_1 plant_1 sensor can_1 pressure
   control_1 rx
CON Rx_p_1 control_1 rx - - control_1 regulate
CON C_n2_1 control_1 regulator can_a valve
   plant_1 rx
CON Rx_v_1 plant_1 rx - - plant_1 actuator
CON C_a_1 control_1 sample - - control_1 alarm
CON C_d_1 control_1 sample - - control_1 display
... etc. for control loop 2 constraints

// End-to-end responses
ETE Loop_1 C_n1_1 Rx_p_1 C_n2_1 Rx_v_1 25000
ETE Alarm_1 C_a_1 5000
ETE Display_1 C_d_1 5000
... etc. for control system 2

All tasks other than the message reception tasks and the timers were implemented by time-consuming loops which did nothing useful. The computation times included in the model were determined experimentally. Measurements were made using a storage scope and simple software instrumentation under conditions in which each task was executed in isolation. The times include the overheads of implementing the scheduling strategy (context switching, etc) and inter-task message handling by the kernel. The small measurement errors introduced by software instrumentation were evaluated and execution times adjusted accordingly. The error associated with task execution time measurement is estimated to be ±3%. In principle, it would have been possible to determine task execution times by counting machine cycles. However, part of the “computation time” includes time lost in the kernel in scheduling and kernel message queueing, etc. We did not have access to this code to allow it to be timed by this method and so instead relied on empirical means.

6.2. The Analysis

The example system of two control loops was analysed first assuming that all tasks on a processor potentially could interfere and all messages potentially could interfere according to the normal priority preemptive scheduling strategy. The results for this type of analysis are presented in the column marked naïve in Figure 4. Evidently, the worst-case execution time of Loop_1 is about 20 ms - this is well within the period of the response (50 ms). Note that the responses Loop_1 and Loop_2 differ markedly; this is a result of CAN message priority assignment: the two messages involved in response Loop_1 have a higher priority than for Loop_2.

The progress of end-to-end response Loop_1 for the naïve analysis is illustrated in Figure 5 indicating the execution sequence of tasks and messages. For each task/message, the dark region indicates the execution/transmission time and the light region the worst-case delay before execution/transmission. The notation control[alarm] means task alarm executing on processor control. Note that the chart indicates that all tasks and messages comprising the response may be delayed before they execute.

Many tasks on each of the processors and some messages are serially constrained; this can be utilised in optimising the analysis. Four opportunities for optimising distributed scheduling models are listed in Figure 6. The transformations are applied by searching precedence graphs for processor or network reuse on each end-to-end response and testing for certain periodic constraints [2]. For example, the $\Gamma_{P1}$ transformation is applicable if the end-to-end response time is less than the period of the response, i.e., only one such response may be in progress at any time. Tasks which
are executed on the same processor at different points in an end-to-end response cannot interfere if this is the case, and the analysis of response times of tasks in the response can be optimised. The transformation is implemented using the analysis tool by defining “interference sets” for each task which reflect the interference possible on that resource; these are subsets of the complete task set. The system was analysed after applying these transformations cumulatively and in sequence, the results of which appear in columns 3,4,5 and 6 of Figure 4. The degree of pessimism inherent in the original analysis is apparent. For example, response Loop_2 is overestimated by 70% and Display_1 by 140% by the naïve analysis. The network scheduling optimisations, \( \Gamma_{N1} \) and \( \Gamma_{N2} \), do not benefit the responses Alarm_1, Display_1, Alarm_2 or Display_2 since these responses do not involve inter-processor message passing. The progress of

6.3. Empirical Study

A complementary empirical study was undertaken to validate the distributed scheduling analysis and to evaluate the importance of optimisation in the development of accurate scheduling models. The experimental systems were implemented using four M68306 micro-controllers and a single CAN network; hardware and software used are listed in Figure 8. The Motorola MC68306 microprocessor was selected because it possesses a simple architecture with no data/instruction cache and no vector pipeline - this facilitated the accurate and repeatable determination of code execution times. Inter-task synchronisation between tasks on the same processor was implemented using message queues. Tasks which received CAN messages were implemented as CAN controller interrupt handlers. The two end-to-end responses Loop_1 and Loop_2 were instantiated periodically on the plant processor by timer interrupt handlers (these are the tasks named timer).

The durations of end-to-end responses were determined using a 68230 Peripheral Interface and Timer. The local clock rate of the timer limited the accuracy of time measurement to \( \pm 4 \mu s \). It was found necessary for the duration of each experiment to prevent interrupt generation by the Ethernet interface. Without this precaution, the measured control loop response times exhibited large random variation. A further source of interrupts from outside the control system was that of the VxWorks system clock. The clock period was set to \( \frac{1}{1000} \) s for all experiments; no use was made of the system clock in determining time delays. The time lost in updating the system clock was measured to be \( 147 \mu s \pm 15 \mu s \) and the impact of this regular interrupt on the system was modelled in \( \lambda \text{rms} \) as a high-priority task, clock. If each of the control loop processors kept perfect time, the messages might never interfere (depending on the starting instances). In fact, the range of clock periods characteristic of crystal clocks means that the control loops did interfere occasionally. However, to improve the chance of interference (and the opportunity of recording worst-case response times) the period of one of the loops was increased.

<table>
<thead>
<tr>
<th>Trans</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma_{P1} )</td>
<td>Interference between tasks executed at different points in the same end-to-end response which also share the same processor.</td>
</tr>
<tr>
<td>( \Gamma_{P2} )</td>
<td>Interference suffered by tasks executed at different points on the same end-to-end response from other tasks executed on the same processor.</td>
</tr>
<tr>
<td>( \Gamma_{N1} )</td>
<td>Interference between messages which are transmitted at different points on the same end-to-end response and on the same network.</td>
</tr>
<tr>
<td>( \Gamma_{N2} )</td>
<td>Interference suffered by messages transmitted at different points in the same end-to-end response from other messages transmitted on the same network.</td>
</tr>
</tbody>
</table>

**Figure 6. Optimising transformations**

end-to-end response Loop_1 for the fully optimised analysis is illustrated in Figure 7 showing the reduced interference following optimisation. On some resources (e.g., plant_1) tasks cannot be delayed, on others (e.g., control_1) tasks suffer reduced interference and on the network, message queueing time is substantially reduced.

**Figure 7. End-to-end response - Loop_1 - optimised analysis**

<table>
<thead>
<tr>
<th>Micro-controllers</th>
<th>Motorola 68030</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fieldbus</td>
<td>CAN</td>
</tr>
<tr>
<td>Network controllers</td>
<td>Intel 82527 [4]</td>
</tr>
<tr>
<td>Network speed and length</td>
<td>72727 bits/s and 20 m</td>
</tr>
<tr>
<td>Download from host</td>
<td>Ethernet</td>
</tr>
<tr>
<td>Software timing</td>
<td>Motorola 68230 PI/T</td>
</tr>
<tr>
<td>Backplane bus</td>
<td>VME</td>
</tr>
<tr>
<td>Real-time kernel</td>
<td>VxWorks 5.1 [12]</td>
</tr>
<tr>
<td>Development language</td>
<td>C - GNU cross-compiled</td>
</tr>
<tr>
<td>Development platform</td>
<td>DEC Alpha</td>
</tr>
</tbody>
</table>

**Figure 8. Experimental hardware and software**
by 0.08% to 50040µs. With this adjustment, network interference occurred with a period of about one minute.

Figure 9 illustrates a single execution of a control loop as two traces captured using a storage scope. The upper trace is the CAN signal on the bus and the signal in the lower trace was asserted at the beginning of the response and cleared at the end. This shows the software delays on the plant and control processors and the transmission of the CAN pressure and valve messages. Following an experimental run involving many control loop executions, the data collected at each plant processor was downloaded to the host for future analysis. The results presented are those for experimental systems which were executed for 4000 control cycles for both Loop 1 and Loop 2. Figure 10 illustrates some the results obtained - Loop 1 response times for each of 4000 control cycles. Note that many responses are clustered at around 7800µs and that responses as great as 13184µs were observed. For about 50% of the control cycles Loop 1 suffered no interference and completed in the minimum time. The remaining control responses suffered additional delay resulting from task and message interference. The regular delay of about 3 ms resulted from interference from tasks sample, alarm and display. The interference pattern of three repeating 1.5 ms delays resulted from network interference. About half-way through the run, these two sources of interference conspire to cause the worst-case delay observed during the run.

6.4. Discussion of the Results

Figure 11 shows the Loop 1 probability density graph computed from the experimental results using a bin size of 100 µs to which has been added for comparison all computed response times. It is apparent that a naive analysis grossly overestimates the worst-case Loop 1 response time by a factor of about 50%. As the pessimism is removed from the analytical scheduling model, the computed response times approach the worst-case empirical value. The analysis assumes worst-case transmission times for messages; this is unlikely in practice since messages may not suffer the worst-case bit-stuffing. The range in message transmission times for an 8-byte message at a bit rate of 72727 bits/s is [1527, 1788]µs. With the particular identifier and data fields used, message transmission times were approximately 1758µs, i.e., 30µs shorter than the worst-case. Taking this, more accurate, value for transmission time into account, the overestimation of the analytical response time is about 1620µs (an error of about 12%) which is approximately equivalent to a single network transmission time. The most optimised analytical model assumed that both messages on a control cycle may suffer interfer-
formance; this was not observed analytically. Either further optimisation is possible (and two message collisions cannot occur), or the experiments did not present the conditions which would have caused the worst-case response times.

7. Conclusions and further work

This paper has described a simple Rate Monotonic Analysis tool to support the design of periodic distributed real-time systems. This is part of an ongoing project to provide better tool support for distributed real-time system design. The example has demonstrated that distributed rate monotonic scheduling potentially is capable of predicting accurate end-to-end response times. The good agreement between analytical and empirical response times improves our confidence in the value of the analysis. However, optimisation of the scheduling model is vitally important if tight bounds are required and this is especially the case for distributed systems. An outcome of a pessimistic analysis is that it may cause the rejection of models (and therefore implementations) which are able in fact to meet their timing obligations.

The probability of experiencing the worst-case delay for a distributed response in an experiment may be very low. Many of the experiments failed to detect important interactions between system components which would lead to longer response times. Notwithstanding this limitation inherent in the empirical approach, such work is valuable in guiding the direction of analysis and improving confidence in computed results.

Further areas of development within this project include the definition of improved optimisation techniques and the verification that they can successfully be applied in larger systems. It is expected that the semantics of inter-task communication will be made more general to permit multiple immediate predecessors for tasks. This will allow the modelling of a wider range of systems. Also of interest are the integration into the analysis of hardware delays and the computation of both worst-case and best-case performance, i.e., both upper and lower bounds on response times.

References