Guaranteeing Hard Real Time End-to-End Communications Deadlines

K. W. Tindell
A. Burns
A. J. Wellings

Real Time Systems Research Group
Department of Computer Science
University of York
e-mail: ken@minster.york.ac.uk

ABSTRACT

In a distributed hard real time system communication between tasks on different processor must occur in bounded time. The inevitable communication delay (termed the *end-to-end* delay) is composed from the delay in transmitting a message on the communications media, and also from the delay in delivering the data to the destination task. This paper gives schedulability analysis bounding the media access delay and the delivery delay, and hence allows the end-to-end delay for a message to be bounded. Two access protocols are considered, TDMA and a 802.5-style token ring. Two approaches are also considered for delivery: an on-demand protocol in which each packet is delivered to the host when it arrives, and a periodic server approach in which the host polls for incoming messages. The schedulability analysis covers all combinations of these access and delivery methods. In addition the effect of incoming messages on the destination processor and its workload is addressed.

1. INTRODUCTION

A hard real time system is often composed from a number of periodic and sporadic tasks which communicate their results by passing messages. In a distributed hard real time system messages are sent between processors across a communications media. In order to guarantee that the timing requirements of all the tasks are met, the total communications delay between sending and receiving a message must be bounded. This total communications delay is often termed the *end-to-end* delay — the time between a message being queued by the sending task and the message fully arriving at the receiving task [13]. This delay is composed of the *access delay*, the *propagation delay*, and the *delivery delay* (Figure 1).

The access delay is the time a message queued at the sending processor spends waiting for the use of the communications media. With a shared media network architecture, a processor must compete with other processors for use of the media.

The propagation delay is usually defined as the time taken for the data to reach the destination processor once physically sent by the source processor. In this paper we are concerned with packet networks, and consider propagation delay to mean the time taken between a packet starting to be transmitted and all of the packet finally arriving at the destination processor.

The delivery delay is the amount of time taken to process the incoming data and deliver it to destination tasks. This work includes such functions as:

- Decoding packet headers
- Reassembling multi-packet messages
- Copying a message into a destination task message buffer (the buffer may be guarded by a semaphore to ensure mutual exclusion)
- Notifying the scheduler of the arrival of a message (the destination task may be blocked

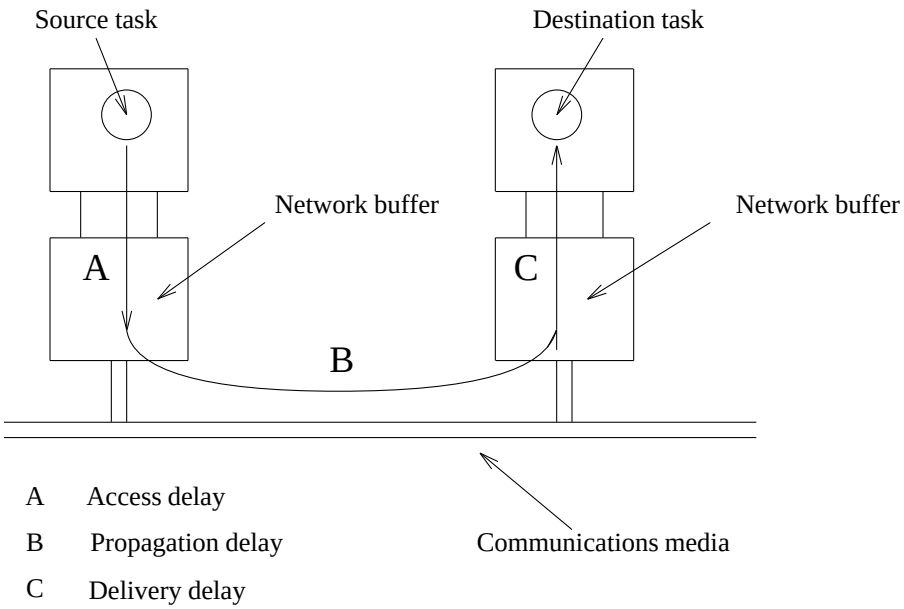

Figure 1: The Components of Communications Delay

awaiting a message).

In practice the delivery delay can be a significant part of the end-to-end delay.

Most research into hard real time communications concentrates on protocols bounding the access delay to shared communications media. For example, the MARS project [4, 6] uses a simple Time Division Multiple Access (TDMA) protocol to resolve communications media contention between processors. A simple priority queue can be used to resolve contention between local messages. Strosnider *et al* [14] provide analysis for the Rate Monotonic [8] scheduling (RMS) algorithm applied to periodic and aperiodic messages sent on an 802.5 token ring. Agrawal *et al* [1] also apply the Rate Monotonic scheduling approach to the FDDI access protocol. The 802.5 token ring protocol is an example of a *global priority* scheme — messages are packetised; before each packet is sent a *packet reservation* protocol is operated: effectively each processor 'bids' for the right to transmit the next packet, with the processor holding the highest priority* packet in the system gaining the right to transmit that packet next.

When considering end-to-end deadlines, the Rate Monotonic schedulability analysis for the 802.5 token ring is no longer sufficient. The Rate Monotonic approach for 802.5 forces the access deadline for a message m, $\mathcal{D}_m$, to be equal to the periodicity of m, T_m (a message m has a periodicity equal to the period of the task which queues m). After allowing for the delivery time this would require that any end-to-end deadline, $\mathcal{E}_m$, be greater than T_m. Clearly this is inflexible, and a more powerful scheduling approach is needed. The Deadline Monotonic scheduling (DMS) algorithm [7] applied to periodic message scheduling would allow $\mathcal{D}_m \leq T_m$, and hence permit $\mathcal{E}_m \leq T_m$ (since the DMS approach is a static priority one, with priorities assigned according to $\mathcal{D}_m$, the algorithm is a superset of the RMS algorithm). In this paper we apply DMS to an 802.5-style token ring protocol, and for comparison also apply the algorithm to a simple TDMA protocol.

*The 802.5 standard permits only 8 distinct priority levels, which is normally too few to avoid priority inversion; this paper analyses an 802.5-style protocol where there is always a sufficient number of priority levels

The paper gives schedulability analysis which bounds the worst-case access delay, D_m, for a message m, where $D_m \leq \mathcal{D}_m$ for a schedulable message.

Very little research has addressed the problems of analysing the *delivery delay*. The MARS project reports on experimental evidence suggesting that DMA cycle-stealing and interrupts from the communications controller can lead to a significant overhead [16].

In this paper we provide new schedulability analysis for two alternative delivery approaches:

- An 'on demand' approach (where incoming data is processed as soon as it arrives)
- A 'periodic server' approach (where the host processor regularly 'polls' the network buffer in the communications controller for incoming data).

This schedulability analysis is then integrated with the analysis bounding the access delays for the two access protocols to provide bounds on the end-to-end communications delay for a given message m.

The rest of the paper is structured as follows:

Section 2 describes an example system and network architecture based on hard real time periodic and sporadic tasks statically assigned to processors connected to a shared communications media via 'intelligent' network controllers. Section 3 presents scheduling theory for the two media access protocols, guaranteeing bounded access times for any given message. Section 4 presents scheduling theory for the two delivery protocols, guaranteeing bounded delivery times. Section 5 discusses some of the issues surrounding the access and delivery protocols, and summarises the schedulability analysis.

2. SOFTWARE AND NETWORK ARCHITECTURE

This section describes the hardware and software of an example network architecture for the purposes of this paper. The architecture is chosen to both represent a realistic network and to provide an example framework for analysis. Both the hardware and assumed software of the example system will be described. Figure 2 shows the hardware.

A number of processors are connected to a shared communications medium each via a Network Interface Unit (NIU). The NIU provides a low-level packet-based interface to the physical communications medium. The interface between the NIU and the host processor is via a shared network buffer. The buffer is partitioned into *outgoing* and *incoming* sections. Outgoing messages are assembled into packets by the host processor and placed in a priority-ordered queue in the outgoing network buffer. Incoming packets are placed in the incoming network buffer by the NIU in FIFO order. The NIU is capable of raising a "packet arrived" interrupt on the host processor. The network buffer has a limited capacity.

Data is transferred to and from the network buffer under the control of the host processor by executing 'move' instructions. In other network architectures data is often transferred by Direct Memory Access (DMA). However, DMA is rarely used for speed: modern processors can move data using 'move' instructions almost as fast as a DMA controller. DMA transfers are sporadic with CPU cycles 'stolen' in a non-deterministic fashion.

In this paper we analyse two alternative media access protocols, and two alternative message delivery protocols. Firstly, the media can be accessed according to an 802.5-style global priority protocol. Packets are copied by the host processor into the network buffer and ordered by priority (we assume an adequate number of priority levels). Before a packet is transmitted by an NIU a packet reservation protocol is performed where each NIU 'bids' for the right to transmit by passing a token. The exact detail of the 802.5 protocol is described by Strosnider *et al* [14]. We assume that packets are of a fixed size. The per-packet overheads of the reservation protocol are variable, but bounded. We therefore assume that the time taken to transmit a packet, including the packet reservation overheads, lies in the range $[\rho_{min} .. \rho_{max}]$.

The other protocol is based on TDMA, with each NIU allowed to transmit a certain number of packets in its 'slot'. The slot is of a fixed size such that if there are no packets to send the communications medium is idle until the slot finishes; each processor may be assigned a different

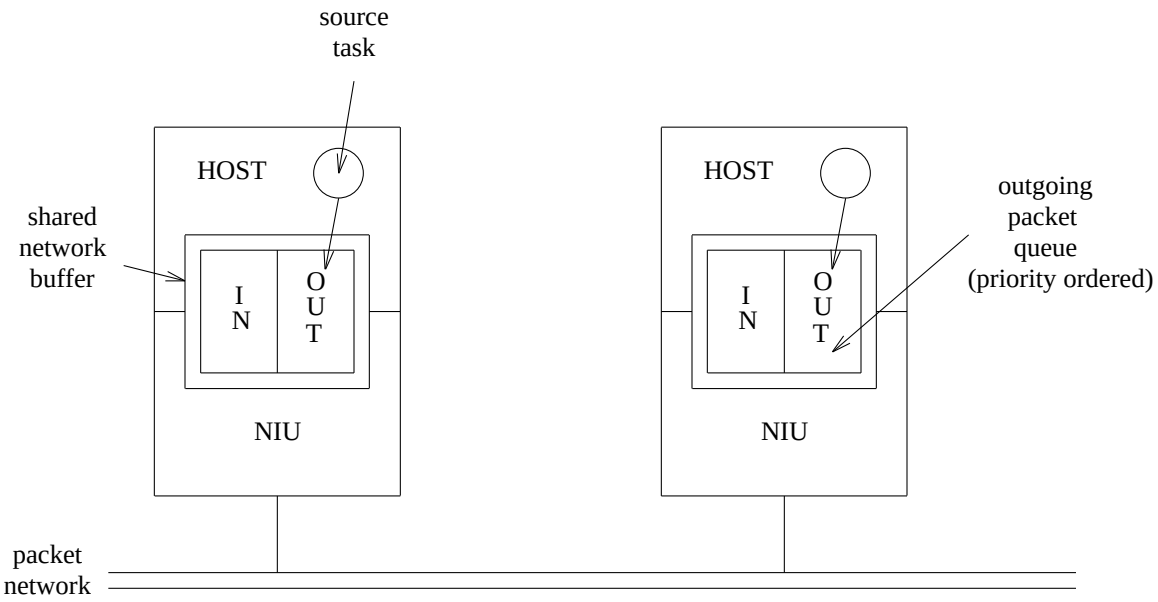

Figure 2: The Physical Architecture

fixed slot size. At the end of the slot another NIU can begin transmitting packets. After all NIUs have been given access, the cycle repeats. This is illustrated in Figure 3.

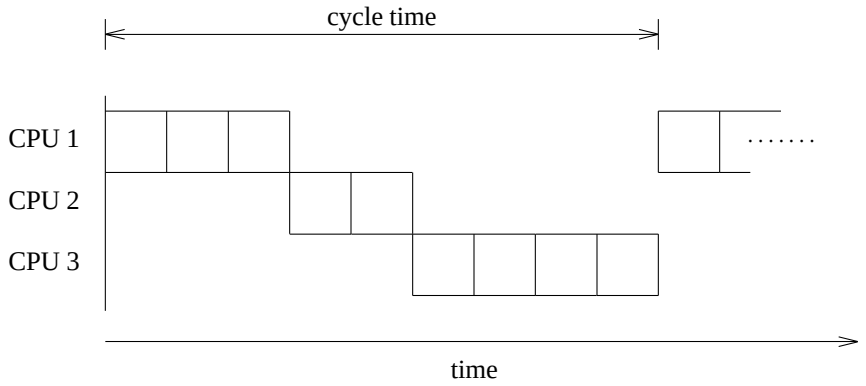

Figure 3: The Simple TDMA Cycle

Note that the TDMA protocol requires clock synchronisation between NIUs to ensure that packet transmission is collision free. Also note that the packet transmission time for the TDMA protocol does not vary (*i.e.* $\rho_{min} = \rho_{max}$) since there is no packet reservation protocol. Hence the packet transmission time is denoted ρ.

There are two alternative delivery protocols. The "on demand" protocol requires that a

"packet arrived" interrupt is raised by the NIU when a packet arrives in the network buffer. The interrupt is handled by the host processor, which releases a sporadic handler to process the packet. The handler copies the packet data from the network buffer and assembles a message. The handler delivers the message to a message buffer owned by the destination task. To prevent concurrent access, the message buffer is guarded by a semaphore; the semaphore is locked and unlocked according to the Priority Ceiling Protocol (PCP) [11]. The arrival of a message at the destination task may require that the scheduler be informed of this (the destination task could be blocked awaiting the arrival of the message). The handler might therefore update scheduling tables, marking the destination task as 'ready to run'. The scheduling tables also need to be guarded by semaphores.

In general, the handler may not have finished before another packet arrives. We assume that the current invocation of the handler is given a higher priority than subsequent invocations of the handler, and thus is always allowed to finish before subsequent packets are processed. Indeed, a sensible implementation of the packet handler would have the handler repeatedly processing packets until there were none remaining, and then 'sleeping' until a packet arrived (*i.e.* until the "packet arrived" interrupt handler re-released the sporadic packet handler).

The alternative delivery protocol is implemented with a periodic server. The server is an ordinary periodic task. When it is released it examines the network buffer for packets, processes each packet until there are none remaining, and then terminates. As with the sporadic handler the server accesses semaphore-guarded message buffers and scheduling tables.

Tasks resident on host processors are scheduled according to the Deadline Monotonic Scheduling (DMS) algorithm. Each task is assigned a static priority based upon its deadline (a short deadline results in a high priority). Some tasks, such as the sporadic handler or the periodic server, have no deadline requirement assigned to them, and a priority must be chosen using other means. With the periodic server, for example, the worst-case response time merely forms part of the end-to-end delay for a message; the timing requirement for schedulability is the end-to-end message delay, not the worst-case response time of the server, and hence it is not meaningful to assign a deadline to the server.

We assume that, per invocation, a task can send a bounded number of messages, of bounded size, to fixed destinations. We also assume that each message m can have end-to-end deadline $\mathcal{E}_m$ (measured from the time the message is queued to the time it arrives at the destination task), and that m is assigned a static priority according to this deadline (some messages may not have end-to-end deadlines, and a priority must be chosen using other means). To simplify the model we add the restriction that m must arrive at the destination task before the end of the period of the sending task (otherwise messages sent from successive invocations of the sending task could delay each other). Furthermore, we assume that this information is known *a priori*.

3. BOUNDING ACCESS DELAYS

This section applies DMS scheduling theory to the bounding of access delays for two shared media access protocols: an 802.5-style global packet priority protocol, and a simple TDMA protocol. Analysis is also provided which allows the outgoing network buffer space requirements to be bounded. Note that Appendix A contains a glossary of symbols used throughout this paper.

A task can queue a message at any point in its execution and each message m therefore inherits a periodicity T_m from the task. A message m consists of P_m fixed size packets, each of which is given the priority of the message. The packets are placed in a priority-ordered queue in the network buffer; the network controller removes packets from the queue and transmits them according to the network access protocol. Thus the access delay is the time the message m spends waiting in the queue (equivalent to the time from being queued to the time the last packet in m begins transmission), and is denoted Q_m.

3.1. Global Priority (802.5-style) Protocol

Access to the network via the global priority protocol can be considered as access via a notional system-wide network queue, where all messages generated either locally or on different processors, and are placed in this notional queue ordered by global priority.

In order to find Q_m the *interference* from system-wide higher priority messages must be found — in the time message m is in the queue it can be preempted by higher priority messages (*i.e.* higher priority messages can be queued in front of m).

The scheduling of deadline monotonic messages is analogous to scheduling deadline monotonic tasks: the periodicity of a message is akin to the period of a task; the number of packets in a message is akin to the worst-case execution time of a task. The interference on a message i from a higher priority message j could be found by calculating the number of times successive instances of message j could be queued in front of a single instance of i. This is given by:

$$\left\lceil \frac{Q_i}{T_j} \right\rceil P_j$$

However, this is not strictly true: a message could be queued anywhere within the worst-case response time of the task queueing the message. This gives rise to the 'back to back hit' problem, indicated by Rajkumar *et al* [10] (in reference to the problem of voluntary task suspension for a period of time). Figure 4 illustrates the problem; although the task has a period T_i, successive messages are $(T_i - D_i)$ close together.

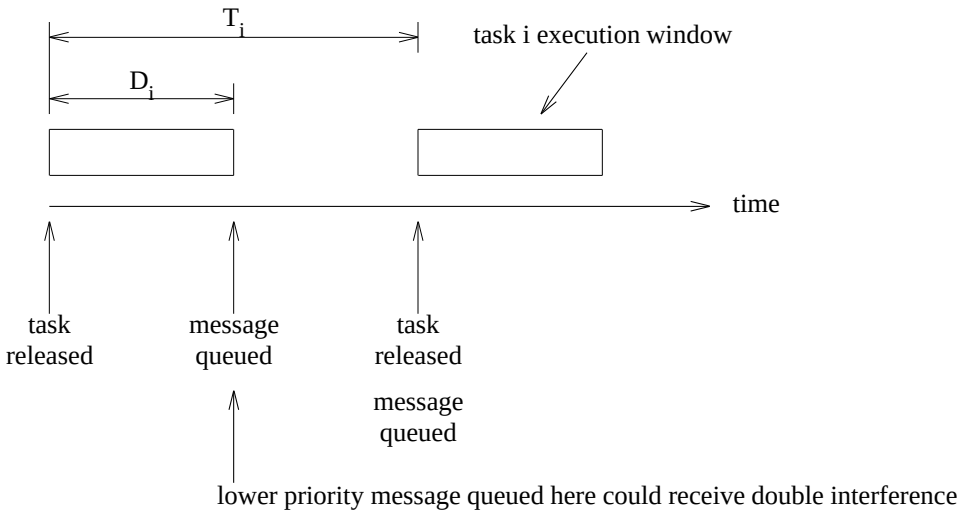

Figure 4: 'Back to Back Hits' from Successive Messages

A task could queue a message at the last possible instant on one invocation, and then as soon as possible in the subsequent invocation. If the worst-case response time of the task was equal to the period of the task two messages could be potentially queued arbitrarily close together, giving rise to a 'back to back hit'. The following equation gives the interference on a message spending time Q_m in the queue, and allows the back to back hit problem to be analysed.

$$J_m = \sum_{j \in hpm(m)} \left\lceil \frac{Q_m + D_{s(j)}}{T_j} \right\rceil P_j \qquad (1)$$

Where:

$s(j)$ The task which sends message j 10

$hpm(m)$ The set of messages in the system of higher priority than m

D_j The worst-case response time of the task queueing message j

T_j The periodicity of j, inherited from the task queuing j

P_j The number of packets from which message j is composed

The derivation of this equation is given in Appendix B. The Appendix also indicates how to find the worst-case response time of the sending task, $D_{s(j)}$.

Given the interference on a message m the queueing time can be found:

$$Q_m = \rho_{\max}\left(P_m + J_m\right)$$

Where:

$\rho_{\max}$ The maximum time taken to transmit a packet. This includes the overhead due to the packet reservation protocol before data transmission takes place, and the time taken to physically transmit the data across the network.

Note that in the 802.5 protocol the overheads when transmitting a packet can vary, and hence the packet transmission time ρ can vary. We define $\rho_{\min}$ to be the smallest packet transmission time (*i.e.* the time taken to transmit the data of a full packet), and $\rho_{\max}$ to be the largest value of ρ (*i.e.* when the maximum overheads are incurred).

In order to calculate Q_m the interference J_m must be found, and *vice versa*. Hence the equations are mutually dependent; a solution can be found by iteration:

$$Q_m^{n+1} = \rho_{\max}\left(P_m + \sum_{j \in hpm(m)} \left\lceil \frac{Q_m^n + D_{s(j)}}{T_j} \right\rceil P_j\right)$$

An initial value Q_m^0 of 0 is suitable. It can be shown that $Q_m^{n+1} \geq Q_m^n$, and hence the iteration is guaranteed to either converge (*i.e.* $Q_m^{n+1} = Q_m^n$), or exceed a threshold value (such as $\mathcal{E}_m$).

3.2. Local Priority (TDMA) Protocol

The simple TDMA protocol, which resolves message contention locally, and global contention via a 'round-robin' protocol, can be analysed in a similar way to that above. Firstly, the total number of packets transmitted in a single TDMA cycle is given by:

$$P_{TDMA} = \sum_{p=1}^{N} S_p$$

Where:

N The total number of processors given slots in the protocol

S_p The slot size, in packets, of processor p

The TDMA cycle time, T_{TDMA}, is therefore given by ρP_{TDMA}. Note that for the TDMA protocol $\rho_{\min} = \rho_{\max}$ since there is no packet reservation protocol, and hence no variation in packet transmission time. Hence ρ is used to denote the packet transmission time. The interference a message m queued on processor p experiences is similar to that experienced in the global protocol:

$$\sum_{j \in lhpm(m)} \left\lceil \frac{Q_m + D_{s(j)}}{T_j} \right\rceil P_j$$

However, $lhpm(m)$ denotes the set of messages of higher priority than m queued on processor p — *i.e.* higher priority messages queued *locally*. The interference from messages at other processors can be modelled as a single high priority message with periodicity equal to T_{TDMA} composed from $P_{TDMA} - S_p$ packets — the messages queued at processor p are only transmitted in the remaining time after the other processors have sent their packets. The interference on a message m queued at processor p is thus given by:

$$J_m = \sum_{j \in lhpm(m)} \left\lceil \frac{Q_m + D_{s(j)}}{T_j} \right\rceil P_j + \left\lceil \frac{Q_m}{T_{TDMA}} \right\rceil \left(P_{TDMA} - S_p \right)$$

As can be seen, the local priority protocol causes *priority inversion* — the messages at other processors are always treated as higher priority than local messages.

The queueing time Q_m can be found iteratively, in the same way as for the global priority protocol:

$$Q_m^{n+1} = \rho \left(P_m + \sum_{j \in lhpm(m)} \left\lceil \frac{Q_m^n + D_{s(j)}}{T_j} \right\rceil P_j + \left\lceil \frac{Q_m^n}{T_{TDMA}} \right\rceil \left(P_{TDMA} - S_p \right) \right)$$

3.3. Bounding Network Buffer Space Used

We now turn to the problem of bounding the network buffer space usage for both access protocols. Recall that we have the following message deadline requirement:

$$D_{s(m)} + \mathcal{E}_m \leq T_m$$

Hence messages sent from successive invocations of the sending task cannot interfere, and thus for both the local and global priority access protocols the worst-case outgoing network buffer space requirements on processor p occurs at a 'critical instant' (where all possible outgoing messages are queued together). The worst-case buffer space requirement, in packets, is therefore given by:

$$\sum_{m \in om(p)} P_m$$

Where:

$om(p)$ The set of outgoing messages from processor p

4. BOUNDING DELIVERY TIMES

The previous section has shown how DMS can be applied to the problem of bounding the access delay. This section addresses the problem of bounding the *delivery time*. Section 2 has described the operation of two alternative delivery approaches: the 'on demand' protocol, and the 'periodic server' protocol. This section gives analysis bounding the delivery times for messages delivered under the two protocols, and indicates how the schedulability of application tasks on the destination processor can be determined.

When incoming packets arrive in the destination processor network buffer they must be delivered to the destination tasks. The delivery mechanism assembles a message and places it in a destination task message buffer. Access to the buffer is controlled by a PCP semaphore (although other protocols, such the Four Slot Mechanism proposed by Simpson [12] could be used). The scheduler may need to be informed of the arrival of the message (so that the destination task can be released), and access to the scheduling information is also controlled by a PCP semaphore.

The delivery work performed is significant, and can rarely be ignored as 'system overheads'. Experiments with the MARS system [16] found that in an 8ms interval DMA cycle-stealing led to the loss of 1026 μs of processing time. The amount of time taken to deliver a message is therefore non-trivial. This section provides schedulability analysis which can be used to determine the worst-case delivery times for the two alternative delivery protocols described in Section 2.

The delivery protocols used have an effect on the satisfaction of two major constraints. Firstly, the computational interference experienced by other tasks running on the destination processor needs to be bounded to allow the schedulability of these tasks to be determined; secondly, the worst-case network buffer space requirement for the destination processor needs to be found — the overflow of the network buffer is a serious event and we wish to discount any system in which this could occur. Analysis is presented which allows the worst-case incoming network buffer usage to be found, and the worst-case interference on other tasks due to incoming packets.

4.1. "On-Demand" Delivery

The operation of the On-Demand protocol has been described in Section 2: as soon as a packet fully arrives at the destination NIU (*i.e.* all of the incoming packet is resident in the network buffer) a "packet arrived" interrupt is raised on the host processor. The interrupt handler releases a sporadic task which copies the data from the network buffer, unpacks the message, and performs the necessary delivery operations.

Figure 5 shows how the end-to-end delay is composed when using the on-demand delivery protocol. As can be seen the *delivery delay* for a message, D_H, is equal to the response time of the sporadic handler, which is given by:

$$D_H = C_H + I_H + K_H \tag{2}$$

Where:

C_H Worst-case execution time of the handler, including context switch costs, and the time taken to copy a packet from the network buffer (recall that data is transferred from the NIU by copying from the network buffer rather than using DMA)

K_H The worst-case blocking time the handler can experience when attempting to lock either the destination task message buffer semaphore, or the scheduling information semaphore (if the PCP is used this is equal to the longest critical section of all lower priority tasks accessing semaphores with ceilings greater than or equal to the priority of the handler)

I_H The worst-case interference an instance of the handler can experience.

In order to find the worst-case response time D_H the interference needs to be bounded. The interference on the handler is the time when other activities pre-empt the handler and consume the processor time, and comes from three sources:

- Preemption by higher priority application tasks (in practice the handler will be assigned a high priority and so there will be few tasks of higher priority)
- Packet arrival interrupts (more packets may arrive while the handler is still active)
- The remaining execution of previous invocations of the handler released to handle packets which arrived earlier

Interference from Higher Priority Tasks

The interference due to higher priority tasks can be found using standard DMS theory, and is given by:

$$\sum_{j \in hpt(H)} \left\lceil \frac{D_H + B_j}{T_j} \right\rceil C_j \tag{3}$$

Where:

D_H The worst-case response time of the handler

$hpt(H)$ The set of tasks of higher priority than the handler

B_j The maximum amount of time task j can voluntarily suspend itself

A Message *m* queued
B Final packet of *m* removed from queue and begins transmission
C Final packet of *m* fully arrives at the destination NIU
D Final packet of *m* fully processed (*m* delivered)

Figure 5: The End-to-End Delay Composition

The derivation of this equation follows from the simple DMS test given by Audsley *et al* [2], and is shown in Appendix B.

Interference from Packet Interrupts

The sporadic handler is considered to have been released as soon as the interrupt on the host is raised, and therefore, the execution of the interrupt handler which actually releases the sporadic is considered to interfere with the sporadic handler (Figure 6).

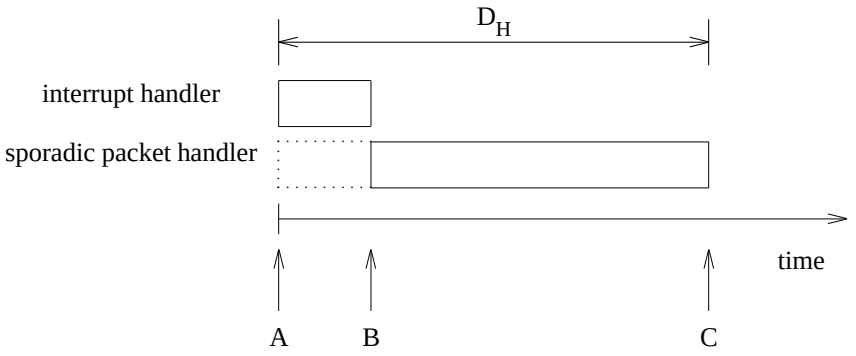

Figure 6: The Release of the Sporadic Handler

In general, the worst-case response time of the sporadic handler can exceed ρ_{min}, the minimum time between packet arrivals, and hence the interference from interrupts raised by subsequent packets arrivals must be considered. There are two ways of bounding the number of subsequent packet arrival interrupts. Firstly, packets can be assumed to arrive at the fastest possible rate (*i.e.* with an inter-arrival time of ρ_{min}). Secondly, use can be made of message passing information known *a priori*: the incoming messages are known beforehand. Given the periodicity and size (in packets) of the messages the total number of incoming packets, and therefore interrupts, can be bounded. Both of these approaches give an upper bound on packet arrival interrupts in an interval, and hence the lower upper bound can be used.

The interference from interrupts assuming that packets arrive at the maximum rate is given by:

$$\left\lceil \frac{D_H}{\rho_{min}} \right\rceil C_I \tag{4}$$

Where:

C_I Worst-case execution time of the "packet arrived" interrupt handler

The number of packets (and hence interrupts) arriving in an interval of duration D_H can also be bounded using the *a priori* message passing information. By analogy between Deadline Monotonic task scheduling and Deadline Monotonic message scheduling we can obtain a bound on the number of packets arriving; likening worst-case computation time to the number of packets in a message, the periodicity of a task to the period of a message, and the arbitrary blocking time of a task to the external blocking time of a message we can say that the number of packets arriving in an interval of duration D_H is:

$$\sum_{m \in im(p)} \left\lceil \frac{D_H + Q_m + D_{s(m)}}{T_m} \right\rceil P_m$$

Where:

$s(m)$ The task sending message m

T_m The periodicity of message m, equal to the period of task sending m

$D_{s(m)}$ The worst-case response time of the task sending message m

Q_m The worst-case time message m spends in the outgoing packet queue

$im(p)$ The set of messages incoming to processor p

The term $Q_m + D_{s(m)}$ is similar to the arbitrary blocking time of a task. However, it should be noted that this is a pessimistic bound.

We have said earlier in Section 2 that the worst-case response time of the sending task plus the worst-case end-to-end delay should be less than the period of the sending task (*i.e.* $D_{s(m)} + E_m \leq T_m$). Given this we can say that

$$\left\lceil \frac{D_H + Q_m + D_{s(m)}}{T_m} \right\rceil = 1$$

and therefore the number of packets arriving in the interval is bounded by:

$$\sum_{m \in im(p)} P_m$$

Hence the computational interference on the sporadic handler from packet arrival interrupts is bounded by:

$$\sum_{m \in im(p)} P_m C_I \tag{5}$$

Equations 4 and 5 both give upper bounds on the packet interrupt interference, and the lower of the two can be used. The packet interrupt interference is bounded by:

$$\min\left[\sum_{m \in im(p)} P_m, \left\lceil \frac{D_H}{\rho_{\min}} \right\rceil \right] C_I \tag{6}$$

Interference from Previous Handler Invocations

The interference between successive invocations of the sporadic handler is now considered. As has been mentioned earlier, in general the worst-case response time of the sporadic handler could exceed $\rho_{\min}$. This means that another invocation of the handler could be released before the current one has finished. We assume that a later invocation of the handler is given a lower priority relative to an earlier one so that a handler is never preempted by subsequent invocations of itself. We also assume that the handler never voluntarily suspends (*i.e.* it never blocks arbitrarily). Consequently, once an invocation of the handler starts to execute all previous invocations must have terminated. Hence at run-time there is no need to implement concurrent handler threads. Another important result of this approach is the effect on the blocking time an invocation can experience when attempting to lock a semaphore under the PCP.

Consider the following situation (illustrated in Figure 7). The current invocation of the handler (denoted τ_1) is running when a packet arrives, releasing τ_2. The interference on τ_2 is bounded by:

$$\min\left(d_{\tau 1} - \rho_{\min}, C_H \right)$$

Where:

$d_{\tau 1}$ The time τ_1 *actually* takes to complete

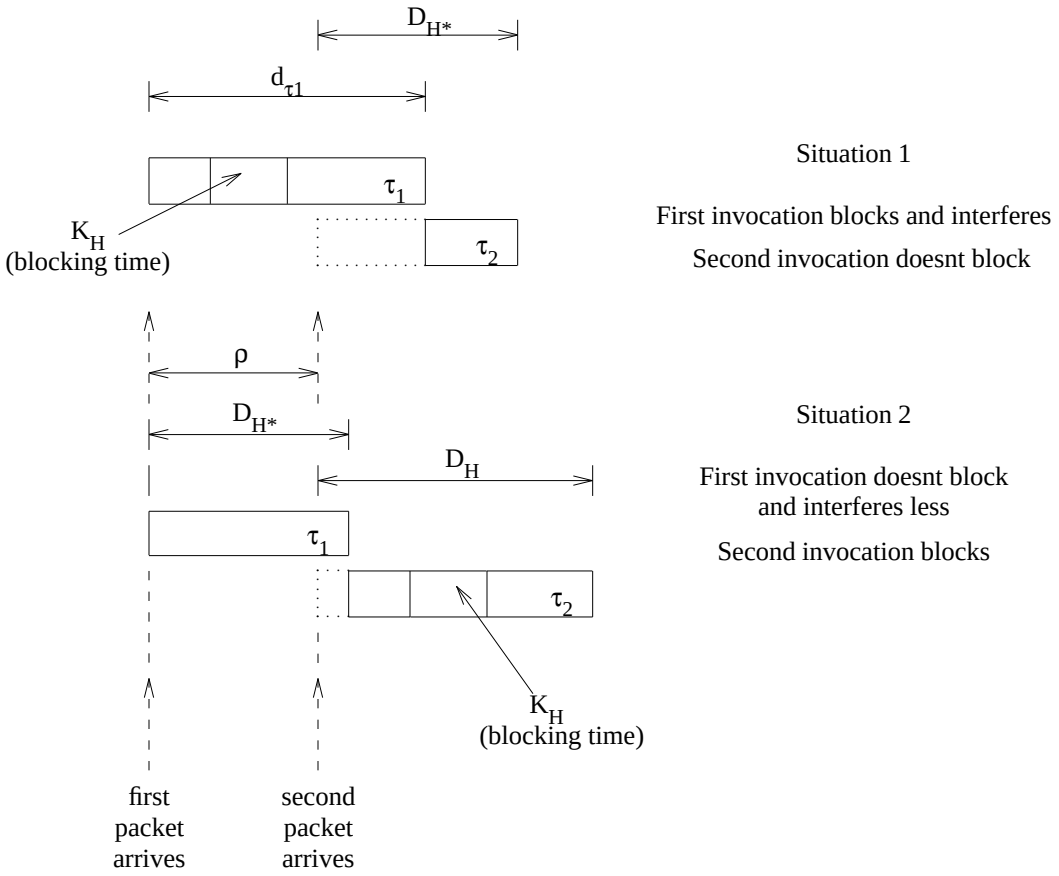

Figure 7: Interference on a Handler from the Previous Invocation

The response time $d_{\tau 1}$ might be large if τ_1 attempted to lock a semaphore under the PCP and was blocked. If $d_{\tau 1} \geq \rho_{min}$ (*i.e.* τ_1 interfered with τ_2) then no lower priority task could execute between the termination of τ_1 and the beginning of the execution of τ_2. Hence, if τ_1 locks a semaphore τ_2 will never be blocked when locking a semaphore. This result is due to the extention of the PCP property "a higher priority task can be blocked at most once" — multiple overlapping invocations of the handler behave as a single task (in fact, the delivery protocol would probably be implemented as a single task).

If τ_1 does not attempt to lock a semaphore it will finish in less time than D_H, and we adopt the notation D_{H^*} to denote this non-blocking worst-case response time (where $D_{H^*} \leq D_H$), and is given by:

$$D_{H^*} = C_H + I_H \tag{7}$$

We can say that the worst-case interference/blocking pattern occurs when τ_1 does not attempt to lock a semaphore but still interferes with τ_2 (*i.e.* $D_{H^*} > \rho_{min}$). and thus when calculating the worst-case interference on an invocation of the handler, the interference due to an earlier *non-blocking* invocation should be used. Now consider Figure 8.

Since $D_{H^*} < \rho$ the interference on τ_4 from previous invocations comes only from three previous invocations (τ_1, τ_2, and τ_3), and the maximum number of previous invocations of a handler which can interfere is given by:

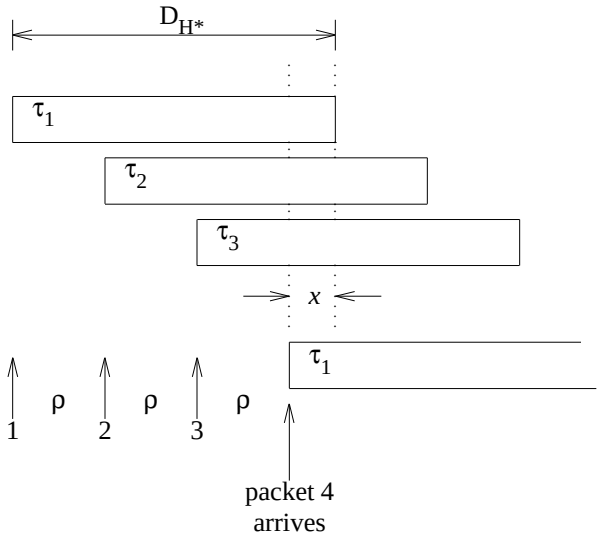

Figure 8: Interference on the Handler from Previous Invocations

$$\left\lfloor \frac{D_{H^*}}{\rho} \right\rfloor$$

The interference from all previous invocations except the oldest is C_H. The oldest (in Figure 8 the oldest is τ_1) interferes over the interval marked x. This interval is given by:

$$D_{H^*} - \left\lfloor \frac{D_{H^*}}{\rho} \right\rfloor \rho$$

The interference from τ_1 is bounded by C_H, and if x is less than C_H then the interference is bounded by x.

Hence the interference on a later invocation of the handler from all earlier invocations can be bounded by:

$$\left(\left\lfloor \frac{D_{H^*}}{\rho_{\min}} \right\rfloor - 1 \right) C_H + \min\left(D_{H^*} - \left\lfloor \frac{D_{H^*}}{\rho_{\min}} \right\rfloor \rho_{\min}, C_H \right) \tag{8}$$

As can be seen, when $D_{H^*} < \rho_{\min}$ the interference is deemed to be zero (even if $D_H > \rho_{\min}$).

Alternatively, the worst-case interference from previous invocations can be bounded by using the *a priori* message passing information. Equation 5 gave a bound on the number of packets arriving in an interval of duration D_H. Thus for a later invocation of the handler, the number of earlier invocations of a handler currently outstanding is bounded by:

$$\left(\sum_{m \in im(p)} P_m \right) - 1$$

And hence the computation interference due to earlier invocations is bounded by:

$$\left[\left(\sum_{m \in im(p)} P_m \right) - 1 \right] C_H \tag{9}$$

Since both Equations 8 and 9 are both upper bounds on the computation interference from earlier invocations, the least upper bound can be taken.

$$\min\left[\left(\left\lfloor \frac{D_{H*}}{\rho_{\min}} \right\rfloor - 1\right) C_H + \min\left(D_{H*} - \left\lfloor \frac{D_{H*}}{\rho_{\min}} \right\rfloor \rho_{\min}, C_H\right), \left[\left(\sum_{m \in im(p)} P_m\right) - 1\right] C_H\right] \tag{10}$$

The total interference on an invocation of the handler is found by summing Equations 3, 6, and 10.

$$I_H = \sum_{j \in hpt(H)} \left\lceil \frac{D_H + D_j}{T_j} \right\rceil C_j + \min\left[\sum_{m \in im(p)} P_m, \left\lceil \frac{D_H}{\rho_{\min}} \right\rceil C_I\right] + \min\left[\left(\left\lfloor \frac{D_{H*}}{\rho_{\min}} \right\rfloor - 1\right) C_H + \min\left(D_{H*} - \left\lfloor \frac{D_{H*}}{\rho_{\min}} \right\rfloor \rho_{\min}, C_H\right), \left[\left(\sum_{m \in im(p)} P_m\right) - 1\right] C_H\right] \tag{11}$$

Equations 2, 7, and 11 are mutually dependent, and a solution can be found by iteration. We define the function $I_H(a, b)$ as equal to Equation 11 with D_H replaced with a, and D_{H*} replaced with b. We can first find D_{H*}:

$$D_H^{n+1} = C_H + I_H(D_{H*}^n, D_{H*}^n) \tag{12}$$

A suitable value for D_{H*}^0 is zero. The iteration proceeds until either it converges (*i.e.* $D_{H*}^{n+1} = D_{H*}^n$) or until D_{H*}^0 exceeds some threshold (such as $\mathcal{E}_m - Q_m$). If the threshold is exceeded then the response time of the handler is too large and the system will therefore be unschedulable.

Having found D_{H*} we can now find D_H:

$$D_H^{n+1} = C_H + K_H + I_H(D_H^n, D_{H*}) \tag{13}$$

A suitable value for D_H^0 is zero. Again, the iteration proceeds until it converges or exceeds some threshold.

Bounding Interference on Other Tasks

Having bounded the delivery time for a message we now turn to the problem of bounding the interference on other tasks on the destination processor, and bounding the incoming network buffer memory requirements.

A task on the destination processor can receive interference due to communications from two sources:

- Packet arrival interrupts
- Handler computation

A task of higher priority than the sporadic handler receives interference only from packet arrival interrupts, whereas a lower priority task receives interference from both. We assume the worst-case response time of a task i on the destination processor p is D_i. In an interval of duration D_i

the maximum number of packet arrivals is given by:

$$\min\left[\left\lceil \frac{D_i}{\rho_{\min}} \right\rceil, \sum_{m \in im(p)} P_m\right] \tag{14}$$

Where:

$im(p)$ The set of messages destined for processor p

Hence for a task resident on processor p the maximum computational interference due to incoming packets on a task i is given by:

$$\left(C_I + h(i,H)C_H\right) \min\left[\left\lceil \frac{D_i}{\rho_{\min}} \right\rceil, \sum_{m \in im(p)} P_m\right]$$

Where:

$h(i,H)$ A function returning 1 if the handler H is of higher priority than i, and zero otherwise

C_I The worst-case computation time of the "packet arrived" interrupt handler

C_H The worst-case computation time of the sporadic handler

Hence the full interference a task on the destination processor can experience is given by:

$$I_i = \sum_{j \in h(i)} \left\lceil \frac{D_i + B_j}{T_j} \right\rceil C_j + \left(C_I + h(i,H)C_H\right) \min\left[\left\lceil \frac{D_i}{\rho_{\min}} \right\rceil, \sum_{m \in im(p)} P_m\right]$$

Where:

$hpt(i)$ The set of tasks of higher priority than i and located on the same processor as i

Appendix B shows how the worst-case response time for a deadline monotonic task can be found once an equation for the interference is developed.

Bounding Network Buffer Space Used

We now turn to the problem of bounding the network buffer space requirement. Recall that a packet takes time $\rho_{\min}$ from starting to fill a packet-sized slot in the network buffer to the time it fully arrives. Also recall that the sporadic handler then takes time D_H to remove and process the packet. Hence the packet requires a network buffer packet slot for an interval of duration at most $\rho_{\min} + D_H$. The maximum number of packets that can arrive in this interval, and hence the maximum network buffer space requirement (measured in packet slots) is therefore bounded by:

$$\min\left[\left\lceil \frac{D_H + \rho_{\min}}{\rho_{\min}} \right\rceil, \sum_{m \in im(p)} P_m\right]$$

4.2. Periodic Server Protocol

The periodic server protocol is a simple one: a periodic task examines the network buffer and processes all packets remaining in the buffer. The task finishes executing when all packets have been removed. Given the period of the server T_S, and the worst-case response time of the server D_S, the worst-case delivery delay for a packet once arrived in the network buffer is given by:

$$T_S + D_S \tag{15}$$

The worst-case response time of the server can be computed (in the same way as for other

Deadline Monotonic tasks) by an iterative equation:

$$D_S^{n+1} = C_S + K_S + \sum_{j \in hpt(S)} \left\lceil \frac{D_S + B_j}{T_j} \right\rceil \tag{16}$$

Where:

$hpt(S)$ The set of tasks of higher priority than S and located on the same processor as S

K_S The worst-case blocking time task S can experience when attempting to lock message buffer and scheduling semaphores

C_S The worst-case execution time task S can require

The computation time required by an invocation of the server depends upon the number of packets that are processed. The worst-case computation time C_S therefore can be bounded by finding the worst-case number of packets that could be in the network buffer when task S is released, plus the number that could continue to arrive during the execution of S.

In the worst-case S could be released, find no packets in the buffer, and terminate immediately; packets could arrive until the task is re-released (T_S later). Packets could then continue to arrive during the execution of the server. Hence the worst-case number of packets which must be dealt with by the server in a single invocation is equal to the maximum number of packets that can arrive in the interval of duration $T_S + D_S$.

The analysis for the maximum number of packets arriving in an interval has been derived for the "on demand" protocol and given earlier (Equation 14); the equation below therefore gives the maximum number of packets the periodic server must deal with in a single invocation:

$$mp_S = \min\left[\left\lceil \frac{T_S + D_S}{\rho_{\min}} \right\rceil, \sum_{m \in im(p)} P_m \right] \tag{17}$$

The worst-case execution time of the server, C_S, is a function of the worst-case number of packets that the server needs to process. We assume that the server requires computation time C_{SP} per packet processed, and an additional worst-case time C_{SF} regardless of the number of packets processed (to include the cost of context switches, for example). The worst-case time C_S is given by:

$$C_S = C_{SF} + mp_S C_{SP} \tag{18}$$

The above equations are mutually dependent — the worst-case computation time cannot be found until the worst-case response time is computed, and *vice versa*. A solution can be found by iteration:

$$D_S^{n+1} = C_{SF} + K_S + \sum_{j \in hpt(S)} \left\lceil \frac{D_S^n + B_j}{T_j} \right\rceil + C_{SP} \min\left[\left\lceil \frac{T_S + D_S^n}{\rho_{\min}} \right\rceil, \sum_{m \in imp} P_m \right] \tag{19}$$

The above equation can be shown to be monotonically increasing and hence $D_S^{n+1} \geq D_S^n$. Therefore, given $D_S^0 = 0$, the equation will either converge to a solution (*i.e.* $D_S^{n+1} = D_S^n$), or will exceed some threshold (such as T_S).

The end-to-end delay for a message m delivered using the periodic server protocol is thus:

$$Q_m + \rho + T_S + D_S \tag{20}$$

Where:

Q_m Worst-case access delay for message m, using either the global or local packet priority protocol

T_S The period of the server, chosen by an appropriate configuration technique

D_S The worst-case response time of the server calculated according to the Equation 19.

Bounding the Interference on Other Tasks

We now address the problem of bounding the interference (and hence determine the schedulability) of other tasks resident on the destination processor. This is straightforward for the "periodic server" protocol since the server is an ordinary deadline monotonic task, accessing semaphores according to the PCP, and never blocking arbitrarily (*i.e.* $B_S = 0$). Hence the interference from the server on a lower priority task i is as for a normal deadline monotonic task:

$$\left\lceil \frac{D_i}{T_S} \right\rceil C_S$$

Bounding the Network Buffer Space Used

The worst-case network buffer requirements are easily found. As mentioned above, an incoming packet still requires a packet-sized slot in the network buffer as it arrives, and hence the interval over which buffer space is required is $T_S + D_S + \rho_{\min}$. Hence the worst-case buffer space requirement, in packets, is:

$$\min\left[\left\lceil \frac{T_S + D_S + \rho_{\min}}{T_S} \right\rceil , \sum_{m \in im(p)} P_m \right] \tag{21}$$

5. DISCUSSION AND SUMMARY

The previous two sections have shown how the end-to-end communications delay for a given message in a known system can be determined. This section discusses some of the issue surrounding each of the access and delivery protocols.

5.1. Optimising TDMA with Periodic Server

The end-to-end delay is composed from the access delay Q_m, the propagation delay ρ, and the delivery delay (D_H for the "on demand" delivery protocol, and $T_S + D_S$ for the "periodic server" delivery protocol). However, there is an 'optimisation' in the worst-case end-to-end delay for the special case of the TDMA access protocol if it is used in conjunction with the periodic server delivery protocol. Firstly, the period of the delivery server is chosen to be equal to T_{TDMA}, and thus the server is always released at the same point in the TDMA cycle. This fixed-offset information can be used to determine a lower delivery time than $T_S + D_S$. Figure 9 illustrates this.

The server at processor D is synchronised so that it is released at the end of the transmission slot of processor S. The amount of time messages sent from S to D spend waiting in the network buffer in D before the periodic server on D is released is bounded by:

$$\rho_{\max}\left(S_S - 1 \right)$$

Where:

S_S The TDMA slot size for processor S

And hence the end-to-end delivery delay for a message m is bounded by:

$$Q_m + \rho S_S + D_{S(D)}$$

Where:

$D_{S(D)}$ The worst-case response time of the periodic server located on processor D

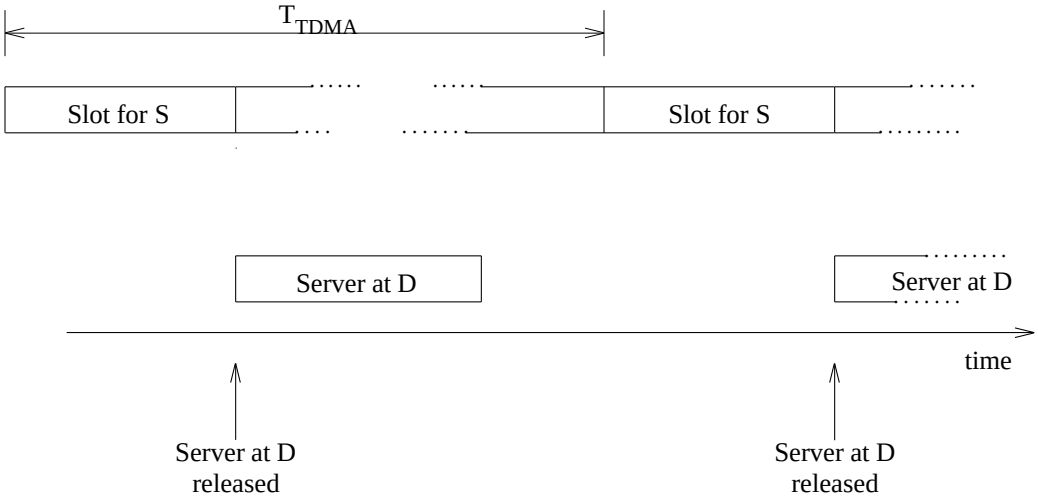

Figure 9: synchronising the periodic server with the TDMA cycle

Since $\rho S_S < T_{S(D)}$ the upper bound on the end-to-end delivery time given above is better than the one suggested in Section 4. For messages sent from other processors to processor D the upper bound on delivery delay will be larger (since the destination server will have a larger offset from the end of the transmission slot of the sending processor) but will always be less than $T_{TDMA} + D_{S(D)}$.

In general, the slot size for a processor S_p, the slot sequence (*i.e.* the order in which processors transmit for their slot), and the periodic server offset (*i.e.* the release time of the server relative to the end of a slot of another processor) are all 'free' parameters of the TDMA plus periodic server communications configuration. These could be chosen to reflect the message passing pattern. For example, the task allocation could be such that the end-to-end deadlines of messages sent from tasks on processor A to tasks on processor B are short in comparison to other message deadlines. It would therefore be sensible to set the release time of the periodic server on processor B to correspond to the end of the slot on processor A.

The Configuration Problem

The above is an example of the *configuration problem*. There are other 'free' parameters: the relative priority of the sporadic handler or the periodic server on each processor must be chosen. A high priority in relation to other tasks on a given processor reduces the response time of the handler/server and thus reduces the worst-case end-to-end delay for a message. However, this reduction in end-to-end delay is at the expense of the schedulability of other tasks on the processor (for example, a high priority handler/server may cause interference with high priority tasks and cause them to miss deadlines). The packet size is another parameter: if the packet size is too small then ρ_{min} becomes small, and the worst-case number of interrupts arriving becomes large (leading to a loss of schedulability). However, if the packet size is large then network bandwidth is wasted since packets are mostly empty, and the access delays become too large.

As can be seen, the tradeoffs when setting the communications parameters are complex — the full effect of some of the parameters can only be determined in the wider *global* setting, and hence the communications configuration problem is part of the wide *global configuration problem*. Current work in the Real Time Systems Research Group at York is applying the Simulated Annealing algorithm [5, 9] to solving these global configuration problems [15].

Priority Inversion and Soft Real Time Messages

As mentioned earlier in Section 3, the TDMA access protocol leads to *priority inversion* — a high priority message can be delayed by low priority messages. Both of the delivery protocols also suffer from priority inversion since packets are delivered in FIFO order. Furthermore, with the "on demand" protocol, the arrival of a low priority packet causes a high priority packet interrupt, and thus interferes with high priority tasks — tasks resident on the destination processor therefore also suffer priority inversion*.

A side effect of the priority inversion problem with the delivery protocol is the problem of soft real time messages. With most access protocols (such as the 802.5 protocol) soft real time messages are queued at low priority and hence are only sent when there are no hard real time messages queued [14]. However, this approach is insufficient when considering the effects of the delivery protocol. If too many soft real time packets arrive at a processor (*i.e.* in excess of the worst-case incoming load determined using the *a priori* message passing information) they could cause a high priority task to miss a deadline. The problem stems from the assumption that as long as soft real time traffic does not interfere with hard real time traffic it can be sent to any processor. This is clearly an unrealistic assumption when the full effects of the arrival of a message are considered. The problem can only be adequately solved if the soft real time message passing pattern is also quantified *a priori*.

6. CONCLUSIONS

The end-to-end deadlines of a message are the crucial deadlines which must be met. The delivery time for message is a significant component of the end-to-end time. Message delivery also has a significant effect on a destination processor. Using the analysis presented in this paper end-to-end message deadlines can be guaranteed (with a variety of media access and message delivery protocols). The schedulability of a destination processor and the worst-case network buffer space requirements can also be determined.

The delivery protocols proposed in this paper suffer from priority inversion which can affect the schedulability of both messages and tasks on the destination processor. A delivery protocol that does not suffer from this is more complex to implement, and the schedulability analysis more difficult. Because of this soft real time traffic cannot be adequately handled with the delivery protocols proposed unless soft real time message passing information is known *a priori*.

APPENDIX A: GLOSSARY

B_j	Worst-case arbitrary blocking time of task *i*
C_i	Worst-case execution time of task *i*
$\mathcal{D}_i$	Deadline for task *i*
D_i	Worst-case response time for task *i*
$\mathcal{E}_m$	End-to-end message deadline for message *m*
E_m	Worst-case end-to-end response time for message *m*
I_i	Computational interference on task *i*
J_m	Packet interference on message *m*
K_i	Worst-case Priority Ceiling Protocol blocking time
N	Number of processors in system
ρ_{max}	Maximum packet transmission time

*It is easy to envisage a priority-based "on demand" delivery protocol which invokes a new handler thread for each packet arriving, with the thread inheriting the priority of the packet. However, this approach requires run-time support for concurrent threads. The schedulability analysis for such a protocol becomes complex.

ρ_{min}	Minimum packet transmission time
P_m	Worst-case number of packets message m requires
Q_m	Access delay for message m
S_p	TDMA slot size (in packets) of processor p
T_{TDMA}	TDMA cycle time
T_x	Periodicity of task or message x
C_{SF}	Base worst-case execution time of periodic message delivery server
C_{SP}	Additional per-packet worst-case execution time of periodic message delivery server
mp_S	Worst-case number of packets periodic message delivery server handles
C_I	Worst-case execution time of "packet arrived" interrupt handler
D_{H*}	Worst-case response time of sporadic packet handler if handler does not lock a semaphore
$hmp(m)$	The set of messages in the system of higher priority than message m
$lhpm(m)$	The set of messages of higher priority than m and sent from the same processor as m
$im(p)$	The set of messages coming into processor p
$om(p)$	The set of messages going out of processor p
$s(m)$	The task which sends message m

APPENDIX B: SCHEDULING THEORY

This appendix contains detail of the background task scheduling theory used in the paper, addresses the problems of finding the worst-case response time of a task, and discusses the problem of arbitrary blocking (and extends the scheduling theory to cater for it).

Worst-case Response Time

The Deadline Monotonic Scheduling problem is usually expressed in the form "does each task meet its deadline?" Often, though, an useful alternative question is "in the worst-case how long will each task take?" We therefore make the distinction between *deadline* and *response time*: the deadline of a task i is the timing requirement, denoted $\mathcal{D}_i$; the response time is the timing performance, denoted D_i.

A sufficient but 'not necessary' test for deadline monotonic tasks is given by Audsley *et al** [3]:

"A set of tasks $\tau_1, \tau_2, ..., \tau_N$, ordered by deadline such that $\mathcal{D}_n \le \mathcal{D}_{n+1}$, is schedulable if:

$$C_i + I_i \le \mathcal{D}_i \tag{22}$$

Where C_i is the worst-case computation time of task τ_i, and I_i is the worst-case interference a task τ_i can experience, given by:

$$I_i = \sum_{j=1}^{i-1} \left\lceil \frac{\mathcal{D}_i}{T_j} \right\rceil C_j \tag{23}$$

Where T_j is the period of task τ_j"

The test is 'not necessary' (*i.e.* failing the test does not imply the task set is unschedulable) because the 'pre-emption window' in Equation 23 is assumed to be $\mathcal{D}_i$ instead of the actual computation window, *i.e.* D_i. The test becomes both sufficient and necessary if $\mathcal{D}_i$ is replaced with D_i in Equation 23:

* Note that Audsley *et al* also gives sufficient and necessary tests that are equivalent to Equation 26.

$$I_i = \sum_{j=1}^{i-1} \left\lceil \frac{D_i}{T_j} \right\rceil C_j \tag{24}$$

The worst-case response time can be found by observing:

$$D_i = C_i + I_i \tag{25}$$

Equations 23 and 24 are mutually dependent; a solution can be found by iteration:

$$D_i^{n+1} = C_i + \left\lceil \frac{D_i^n}{T_j} \right\rceil C_j \tag{26}$$

The iteration starts with $D_i^0 = 0$. The iteration proceeds until D_i^n exceeds $\mathcal{D}_i$ (task is therefore unschedulable), or when $D_i^{n+1} = D_i^n$. In the latter case the iteration has converged to a solution. The iteration is guaranteed to either halt or converge if it can be shown that $D_i^{n+1} \geq D_i^n$, which we will now prove.

Theorem

Given:

$$D_i^{n+1} = C_i + \left\lceil \frac{D_i^n}{T_j} \right\rceil C_j$$

Where C_i, C_j, and T_j are integer constants ≥ 0, and $D_i^0 = 0$.

Then:

$$D_i^{n+1} \geq D_i^n$$

Lemma

$$D_i^n \geq D_i^{n-1} \rightarrow D_i^{n+1} \geq D_i^n$$

Proof

If:

$$D_i^n \geq D_i^{n-1}$$

Then:

$$\frac{D_i^n}{T_j} \geq \frac{D_i^{n-1}}{T_j}$$

Therefore:

$$\left\lceil \frac{D_i^n}{T_j} \right\rceil \geq \left\lceil \frac{D_i^{n-1}}{T_j} \right\rceil$$

Hence:

$$C_i + \sum_{j=1}^{i-1} \left\lceil \frac{D_i^n}{T_j} \right\rceil \geq C_i + \sum_{j=1}^{i-1} \left\lceil \frac{D_i^{n-1}}{T_j} \right\rceil$$

And so:

$$D_i^{n+1} \geq D_i^n$$

□

Proof of Theorem

We have:

$$D_i^0 = 0$$

and

$$D_i^1 = C_i$$

Therefore $D_i^1 \geq D_i^0$, and hence by induction $D_i^{n+1} \geq D_i^n$. □

The Priority Ceiling Protocol [11] can be used with Deadline Monotonic Scheduling — Equation 26 is modified to include the time a higher priority task could be blocked by a lower priority task:

$$D_i^{n+1} = C_i + K_i + \left\lceil \frac{D_i^n}{T_j} \right\rceil C_j \tag{27}$$

Where K_i is the worst-case time a task of lower priority than task i can hold a semaphore with ceiling greater than or equal to the priority of task i.

The "Arbitrary Blocking" Problem

The problem of *arbitrary blocking* (sometimes called *external blocking*) is addressed here. Arbitrary blocking is the blocking of a task which occurs when a task voluntarily suspends itself for a period of time (perhaps waiting for a certain external event to occur). When a task suspends itself in this way a lower priority task may start or resume execution while the higher priority task is blocked. Arbitrary blocking can give rise to the 'back to back hit' problem, indicated by Rajkumar *et al* [10], and discussed in Section 3.1. Figure 10 further illustrates the problem.

The simple DMS interference value test given in Equation 24 would indicate that task L receives an interference of C_H. However, as can be seen, the worst-case interference is $2C_H$. This additional interference is due to the invasive effects of arbitrary blocking — a previous invocation of H can be delayed so that two invocations of H can occur closer together than would ordinarily occur.

To calculate the new worst-case interference consider Figure 10. Task H can suspend itself, per invocation, for a time not exceeding B_H. From Figure 10 it can be seen that the interference on task L is C_H if $D_L \leq T_H - B_H$. The interference is $2C_H$ if $T_H - B_H < D_H \leq 2T_H - B_H$, and $3C_H$ if $2T_H - B_H < D_H \leq 3T_H - B_H$, and so on.

In general, the interference on task L from task H is nC_H when:

$$(n-1)T_H - B_H < D_L \leq nT_H - B_H$$

$$(n-1)T_H < D_L + B_H \leq nT_H$$

$$(n-1) < \frac{D_L + B_H}{T_H} \leq n$$

$$\lceil (n-1) \rceil < \left\lceil \frac{D_L + B_H}{T_H} \right\rceil \leq \lceil n \rceil$$

Hence:

$$\lceil (n-1) \rceil < \left\lceil \frac{D_L + B_H}{T_H} \right\rceil = \lceil n \rceil$$

Hence the interference from higher priority tasks on a task i is given by:

$$I_i = \sum_{j=1}^{i-1} \left\lceil \frac{D_i + B_j}{T_j} \right\rceil C_H \tag{28}$$

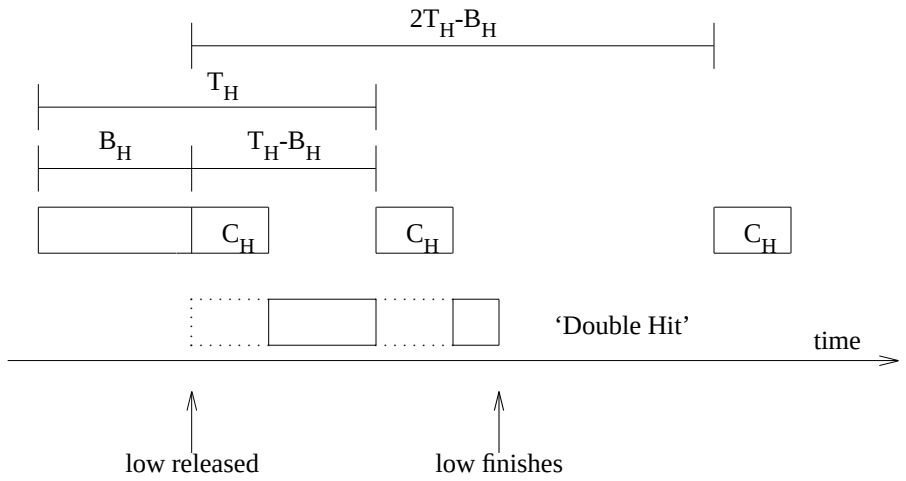

Figure 10: 'Back to Back' Interference on a Task

7. REFERENCES

1. G. Agrawal, B. Chen, W. Zhao and S. Davari, *Architecture Impact of FDDI Network on Scheduling Hard Real Time Traffic*, Workshop on Architectural Aspects of Real Time Systems (December 1991).
2. N. C. Audsley, ''Deadline Monotonic Scheduling'', YCS.146, Computer Science Department, University of York (September 1990).
3. N. C. Audsley, A. Burns, M. F. Richardson and A. J. Wellings, ''Hard Real-Time Scheduling: The Deadline Monotonic Approach'', *Proceedings 8th IEEE Workshop on Real-Time Operating Systems and Software*, Atlanta, USA (15-17 May 1991).
4. A. Damm, J. Reisinger, W. Schwabl and H. Kopetz, ''The Real-Time Operating System of MARS'', *ACM Operating Systems Review* **23**(3 (Special Issue)), pp. 141-15 (1989).
5. S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, ''Optimisation by Simulated Annealing'', *Science*(220), pp. 671-680 (1983).
6. J. W. Korte, *Grundlagen der Strassenverkehrsplanung in Stadt und Land*, Wiesbaden Bauverlag GmbH (1960).
7. J. Y. T. Leung and J. Whitehead, ''On The Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks'', *Performance Evaluation*, pp. 237-250 (Vol. 2, Part 4, Dec 1982).
8. C. L. Liu and J. W. Layland, ''Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment'', *Journal of the ACM* **20**(1), pp. 46-61 (1973).
9. N. Radcliffe and G. Wilson, ''Natural Solutions Give Their Best'', *New Scientist*, pp. 47-50

(14th April 1990).

10. R. Rajkumar, ‘‘Real-Time Synchronisation Protocols for Shared Memory Multiprocessors’’, *Proceedings 10th IEEE International Conference on Distributed Computing Systems*, Paris, IEEE Computer Society Press (28 May - 1 June 1990).
11. L. Sha, R. Rajkumar and J. P. Lehoczky, ‘‘Priority Inheritance Protocols: An Approach to Real-Time Synchronisation’’, *IEEE Transactions on Computers* **39**(9), pp. 1175-1185 (September 1990).
12. H. Simpson, ‘‘Four-Slot Fully Asynchronous Communication Mechanism’’, *IEE Proceedings*, pp. 17-30 (Vol. 137, Pt. E., No. 1, Jan 1990).
13. John A. Stankovic, ‘‘Misconceptions About Real-Time Computing: A Serious Problem for Next-Generation Systems’’, *IEEE Computer* **21**(10), pp. 10-19 (October 1988).
14. J. K. Strosnider, T. Marchok and J. Lehoczky, ‘‘Advanced Real-Time Scheduling Using the IEEE 802.5 Token Ring’’, *Proceedings of the IEEE Real-Time Systems Symposium*, Huntsville, Alabama, pp. 42-52 (1988).
15. K. Tindell, A. Burns and A. Wellings, ‘‘Allocating Real-Time Tasks (An NP-Hard Problem made Easy)’’, *Real Time Systems* (To Appear).
16. A. Vrchoticky and P. Puschner, *On The Feasibility of Response Time Predictions - An Experimental Evaluation*, PDCS Project (Esprit BRA Project 3092), Second Year Report (May 1991).