Faculdade de Engenharia da Universidade do Porto
Departamento de Engenharia Electrotécnica e de Computadores

Supporting Real-Time Communications with Standard Factory-Floor Networks

By

Eduardo Manuel de Médicis Tovar

A dissertation submitted in partial fulfilment of the requirements for the degree of
Doctor in Electrical and Computer Engineering

July 1999

Supervised by: Prof. Francisco Vasques

Supporting Real-Time Communications with Standard Factory-Floor Networks

Abstract

Fieldbus networks are widely used as the communication support for distributed computer-controlled systems (DCCS), in applications ranging from process control to discrete manufacturing. Usually, DCCS impose real-time requirements to the communication network; that is, traffic must be sent and received within a bounded interval, otherwise a timing fault is said to occur. This motivates the use of communication networks where the Medium Access Control (MAC) protocol is able to schedule messages according to their real-time requirements.

In the past, the scope of fieldbuses was dominated by vendor-specific solutions, which were mostly restricted to specific application areas. More recently, vendor-independent standardised fieldbuses, supporting the open system concept, have started to be commonly used. Particular relevance must be given to the European Standard EN 50170, which encompasses three widely used fieldbuses: P-NET, PROFIBUS and WorldFIP.

The main research objective of this thesis is to develop analysis and methodologies to guarantee, prior to run-time, that real-time distributed computer-controlled systems (DCCS) can be successfully implemented with standard fieldbus communication networks, such as those defined by the European Standard EN 50170.

In this thesis, we characterise the MAC temporal behaviour for each one of these three EN 50170 profiles. More importantly, we provide analytical formulae for the evaluation of the worst-case response time of messages in these networks. These formulae constitute a set of powerful tools to guarantee the timing requirements of distributed time-critical applications, where distribution is supported by one of the EN 50170 profiles.

Finally, we also show how priority-based scheduling mechanisms can be implemented at the application process level of P-NET and PROFIBUS masters, in order to overcome the limitations of their FCFS outgoing queues. Moreover, we demonstrate how the methodologies used to guarantee the timing requirements of tasks scheduling, can be successfully adapted to encompass the characteristics of P-NET and PROFIBUS networks.

Keywords: Real-time systems; Real-time communications; Fieldbus networks.

Suporte à Comunicação de Tempo-Real utilizando Redes Industriais Normalizadas

Resumo

As redes industriais são largamente utilizadas para o suporte de sistemas controlados por computador distribuídos (DCCS, em inglês), em aplicações que vão desde o controlo de processos até à automação industrial. Habitualmente, os DCCS impõem requisitos de tempo-real à rede de comunicação, segundo os quais mensagens devem ser enviadas e recebidas num tempo inferior a um determinado limite, caso contrário diz-se que ocorreu uma falha temporal. Este facto motiva a utilização de redes de comunicação, para as quais o protocolo de controlo de acesso ao meio (MAC, em inglês) é capaz de escalonar as mensagens de acordo com os seus requisitos de tempo-real.

No passado, a área das redes industriais encontrava-se dominada por soluções proprietárias, as quais estavam frequentemente focalizadas para áreas de aplicação específicas. Mais recentemente, redes industriais normalizadas começaram a ser utilizadas com maior frequência. Uma particular relevancia deve ser dada à norma Europeia EN 50170 que congrega três redes industriais largamente utilizadas: P-NET, PROFIBUS e WorldFIP.

O objectivo fundamental de investigação desta tese é o de desenvolver análises e metodologias que permitam garantir, *a priori*, que aplicações DCCS de tempo-real podem ser implementadas com sucesso utilizando redes de comunicação industriais, em particular as definidas pela norma EN 50170.

Neste trabalho de tese, caracterizamos o comportamento temporal do protocolo de MAC para cada um dos três perfis da norma EN 50170. Mais precisamente, fornecemos métodos analíticos para o cálculo do tempo de resposta, no pior caso, de mensagens nestas redes. Estes métodos analíticos constituem a base de uma série de ferramentas poderosas para a garantia dos requisitos temporais de aplicações DCCS, nas quais a distribuição é suportada por um dos perfis da norma EN 50170.

Finalmente, também mostramos como mecanismos de escalonamento baseados em prioridades podem ser implementados ao nível do processo de aplicação em estações mestre de P-NET e de PROFIBUS, por forma a ultrapassar as limitações inerentes às suas filas de saída FCFS. Também demonstramos como as metodologias utilizadas para garantir os requisitos temporais no escalonamento de tarefas, podem ser adaptadas com sucesso para incorporar as características de redes P-NET e PROFIBUS.

Palavras chave: Sistemas de Tempo-Real; Comunicações de Tempo-Real; Redes Industriais.

Support de la Communication Temps-Réel en utilisant des Réseaux Industriels Normalisés

Résumé

L'utilisation des réseaux industriels pour le support de la communication dans les systèmes distribués controlés par ordinateur (DCCS, en anglais) est très répandue dans les applications industrielles. Habituellement, les applications de DCCS imposent des besoins temps-réel strict au réseau de communication, c'est-à-dire, soit le trafic est envoyé et reçu dans un délai borné, soit on dit qu'il a eu une faute temporelle. Celle-ci est la motivation pour l'usage des réseaux de communication qui possèdent des protocoles de contrôle d'accès au médium (MAC, en anglais) capables d'ordonnancer messages selon ses besoins temps-réel.

Dans le passé, le domain des réseaux industriels a été dominé par des solutions propriétaires, qui étaient souvent ciblées pour des applications spécifiques. Récemment, des réseaux industriels normalisés et donc non propriétaires, qui supportent le concept de système ouvert, ont commencé a être habituellement utilisés. Parmi ces solutions normalisées, une importance remarquable doit être donnée à la Norme Européenne EN50170, qui regroupe trois réseaux industriels très utilisés: P-NET, PROFIBUS et WorldFIP.

L'objectif principal de recherche de cette thèse est celui de developer des analyses et des métodologies por garantir, hors-ligne, que les applications DCCS temps-réel peuvent être implantées en utilisant des réseaux industriels normalisés, tels que ceux définis par la norme EN 50170.

Dans cette thèse, nous caractérisons le comportement temporel du protocole MAC pour chacun des trois profils de la norme EN 50170. En particulier, nous fournissons des formules analytiques pour l'évaluation, dans le pire cas, du temps de réponse de messages dans chacun de ces trois profils. Ces formules constituent un ensemble d'outils crucialles pour la garantie des besoins temporels dans les applications distribuées temps critique, où la distribution est supportée par un des profils de la norme EN 50170.

Finalement, nous montrons aussi comment implanter des mécanismes d'ordonnancement par priorité au niveau du processus d'application des maîtres P-NET et PROFIBUS, de façon à surpasser les limitations de ses files de sortie FCFS. En plus, nous montrons comment les méthodologies utilisées pour garantir les besoins temporels dans l'ordonnancement des tâches, peuvent être adaptées pour incorporer les caractéristiques des réseaux P-NET et PROFIBUS.

Mots-clés: Systèmes Temps-Réel; Communications Temps-Réel; Réseaux Industriels.

aos meus Pais,
ao Gonçalo, à Bé e ao Filipe

Acknowledgements

First, I would like to express my deepest gratitude to Francisco Vasques. As my thesis supervisor, he helped me select the right topic at the very beginning which importance to this thesis can never be overemphasised. Throughout the course of this work, he has given me constant encouragement, generous support, insightful comments, and invaluable suggestions, which benefit the completion of this thesis. It would be hard to find a more dedicated and concerned supervisor.

I have benefited greatly from many other people in the University of Porto, which gave me remarkable opportunities that benefited my career in a long time to come. A special thanks to Artur Cardoso for all his support and insightful comments.

I would like to acknowledge the financial support from FLAD (Fundação Luso-Americana para o Desenvolvimento) and from DEMEGI-FEUP.

Special thanks to the Polytechnic Institute of Porto and its School of Engineering (IPP and ISEP, respectively), for all the efforts and financial support, which made possible the prosecution of this work. I address them to Prof. Luís Soares (President of the IPP), to Vítor Santos (Chairman of the Executive Board of ISEP), to Carlos Ramos (Director of the CIM Centre of ISEP), to Jorge Mendes (Head of the Computer Engineering Department of ISEP) and to João Rocha (Co-ordinator of the Industrial Informatics Group). I also acknowledge the support received from my colleagues of the Computer Engineering Department and from the CIM Centre of ISEP.

Thanks also to the Department of Computer Science of the University of York. I have benefited a lot from the visits to the Real-Time Systems Group. Thanks for all the fruitful discussions and for the inspiring atmosphere.

IPP-HURRAY! - an **IPP** Research Group **HU**gging **R**eal-time and **R**eliable **A**rchitectures for computing s**Y**stems. Mário, Miguel and Luís: many thanks for all the support and friendship.

Finally, I am most grateful to my family and friends, for their dedicated support and understanding.

Porto, 23 July 1999

Eduardo Manuel de Médicis Tovar

Table of Contents

Chapter 1

Overview

The main focus of this thesis is "how to guarantee real-time communications using standard fieldbus networks". In this chapter we give an overview of the overall research domain, in order to establish the context and research objectives of this thesis. Finally, at the end of this chapter, we summarise the main contributions of this work

1.1. Introduction

In the past decade, manufacturing schemes have changed dramatically. In particular, the computer integrated manufacturing (Rembold *et al.*, 1993) concept has been stressed as a means to achieve greater production competitiveness. The driving forces behind the changes also resulted from the increased development and utilisation of new technologies, which make massive use of microprocessor-based equipment.

Integration implies that the different subsystems of the manufacturing environment interact and co-operate with each other. This means transfer, storage and processing of information in a widespread environment. In other words, integration requires efficient support for data communications.

Nowadays, communication networks are available to virtually every aspect of the manufacturing environment, ranging from the production planning to the field level (Pimentel, 1990). However, the use of communication networks at the field level is a much more recent trend (Decotignie and Pleinevaux, 1993). Indeed, only more recently network interfaces become cost-effective for the interconnection of devices such as sensors, actuators, and small controllers which, in the majority of the cases, are expected to be cheaper than the equipment typically interconnected at upper control levels of the manufacturing environment (e.g., workstations or numerically-controlled machines).

1.1.1. Distributed Computer-Controlled Systems (DCCS)

This thesis is focused on the field level, where typically the process-relevant field devices are used by a computer system to automatically conduct the process.

A computer-controlled system can be decomposed into a set of three subsystems (Fig. 1.1): the controlled object; the computer system; and the human operator (Kopetz, 1997). Collectively, the controlled object and the human operator can be referred to as the environment of the computer system.

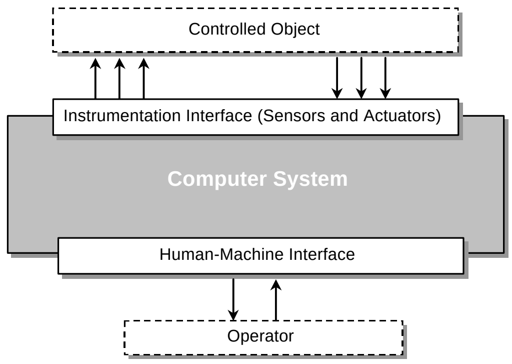


Fig. 1.1 This figure illustrates the interactions between the three subsystems that typically compose a computer-controlled system

The interface between the human operator and the computer system is called the man-machine interface, and the interface between the controlled object and the computer system is called the instrumentation interface. The man-machine interface consists of input devices (e.g., keyboard) and output devices (e.g., display) that interface to the human operator. The role of the man-machine interface is to provide the computer system with, for example, process control set points or device parameters for sensors and actuators. It is also used to provide the operator with process control supervision. The instrumentation interface consists of sensors and actuators that transform the physical signals in the controlled object into a digital form suitable to the computer system and vice-versa.

The role of the computer system is to react to stimuli from the controlled object (this is the essence of a computer-controlled system) or the operator (Kopetz, 1997). Basically, the computer system should be able to receive, via the instrumentation interface, information about the status of the controlled object, compute new commands according to the references provided by the man-machine interface, and transmit those new commands to the actuators, also via the instrumentation interface. To perform these operations, the computer system should be provided with a control application program.

A computer-controlled system can have a centralised architecture. By centralised architecture we mean that there is only one single computer system unit, which has I/O capabilities to support both the instrumentation and the man-machine interfaces. The field devices (sensors and actuators) are connected to the computer system via point-to-point links. Fig. 1.2a illustrates such an architecture.

There are several advantages if a field level communication network is used as a replacement for the point-to-point links (between the sensors/actuators and the computer system). The main advantage is an economical one. Indeed, this is perhaps its single best advantage (Pimentel, 1990). As it can be depicted from Fig. 1.2, a cost reduction can be obtained if a single wire, as a network communication medium, replaces a significant part of the point-to-point wires. Naturally, the use of a single wire brings other important advantages, such as easier installation and maintenance, easier detection and localisation of cable faults, and easier expansion due to the modular nature of the network.

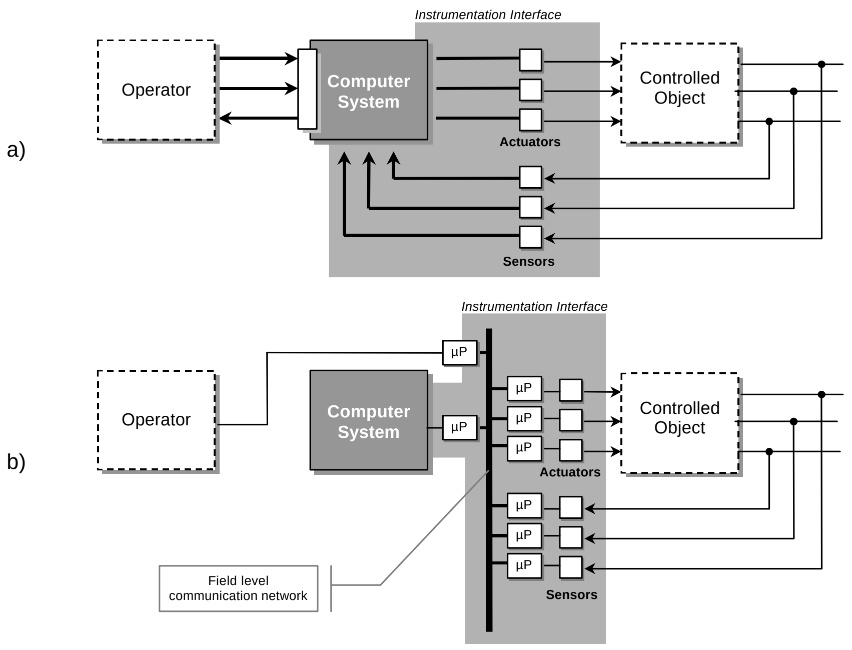


Fig. 1.2 This figure highlights the main advantage of a decentralised computer-controlled architecture (b) when compared to a centralised computer-controlled architecture (a)

Typically, a field level network will be a broadcast network (like in most types of local area networks), where several network nodes share a common communication channel. Messages are transmitted from a source node to a destination node via the shared communication medium. A major problem occurs when at least two nodes attempt to send messages via the shared medium at about the same time. This problem is solved by a medium access control (MAC) protocol. Although network protocols for the field level are expected to be simpler than those used in upper level networks, the connection of sensors and actuators[1] to a shared medium implies the use of a microprocessor-based network interface. This network interface implements all the required field network protocols, and most notably the MAC protocol. The network interface brings some processing capabilities closer to the sensors and actuators, and this constitutes an additional advantage of a decentralised computer-controlled architecture, as those processing capabilities may be used to perform some signal conditioning or pre-processing operations before sending the data over the network.

With the increased availability of low cost technology, the decentralised computer-controlled architecture can easily evolve to a distributed computer-controlled architecture. Basically, the difference lies on the distribution of the algorithms related to the control application. In a centralised computer-controlled architecture all the control

[1] In Fig. 1.2b each field device is individually connected to the field level network. In actual applications, each network node will typically encompass a group of I/O points.

algorithms are implemented in a single computer system. In a decentralised computer-controlled architecture, all the control algorithms run also in a single computer system (now also a network node), even if some of the processing tasks (signal conditioning or pre-processing operations) can be executed in the network nodes that interface the network to the sensors and actuators. Contrarily, in a distributed computer-controlled architecture the tasks of the control algorithms may be distributed throughout several computing nodes. Fig. 1.3 depicts the organisation of a distributed computer-controlled architecture.

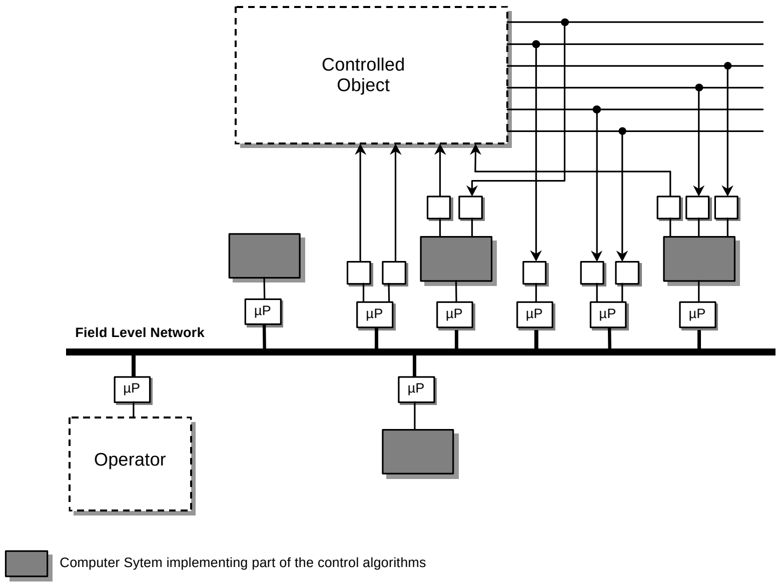


Fig. 1.3 This figure shows an example of a distributed computer-controlled architecture

The ability to support distributed control algorithms is another advantage achievable by the use of field level networks. This eases the design of computer-controlled systems where distribution of control, decentralisation of measurement tasks, and number of intelligent microprocessor-controlled devices is ceaselessly increasing. Control systems based on distributed computer-controlled architectures are labelled as distributed computer-controlled systems (DCCS) (Prince and Soloman, 1981).

1.1.2. Timing Requirements of Computer-Controlled Systems

Most of the computer-controlled systems are also real-time systems. For instance, assume that one of the inputs of the computer system concerns an alarm condition. The computer system must be able to handle such an alarm condition (process that input and produce outputs accordingly), within a bounded time interval. Thus, a computer system must not only react to stimuli from the controlled object, which in essence means the provision of new commands based on the current state of the controlled object, but emphatically it must react to stimuli of the controlled object within time intervals dictated by its environment.

Suppose a simple centralised computer-controlled architecture. The control program that runs on the computer system reads inputs from field devices, processes these inputs, and then produces outputs to be sent to other field devices. Assume that the input data from the field devices arrives to the computer system through asynchronous interrupts, which are then processed in a first-come-first-served (FCFS) basis. Assume also that the control program has a number of individual tasks (processes), each one assigned to the processing of one of the inputs.

If there are as many input buffers as input devices, it may be a timing requirement to be fulfilled by the computer system that no input values are overwritten before their complete processing. Thus, we can say that the relative deadline for each task (assume that they become ready to execute at the time of the input arrival) corresponds to the periodicity of the related input arrivals. If the time interval between the arrival of an input and the completion of the processing for that input is denoted as the response time of a task then, the maximum admissible response time of the task must be smaller than the periodicity of the related input arrival.

It is obvious that the longest response time for a task occurs when all the inputs arrive simultaneously to the computer system. As for this scenario the interrupt requests can be queued in any arbitrary order, the longest response time for a task is given by the sum of the worst-case execution times of each individual input-related task. Fig 1.4 illustrates this reasoning, for a system with 4 inputs, where the worst-case response time for task 4 is longer than the its deadline, and therefore deadlines can be missed.

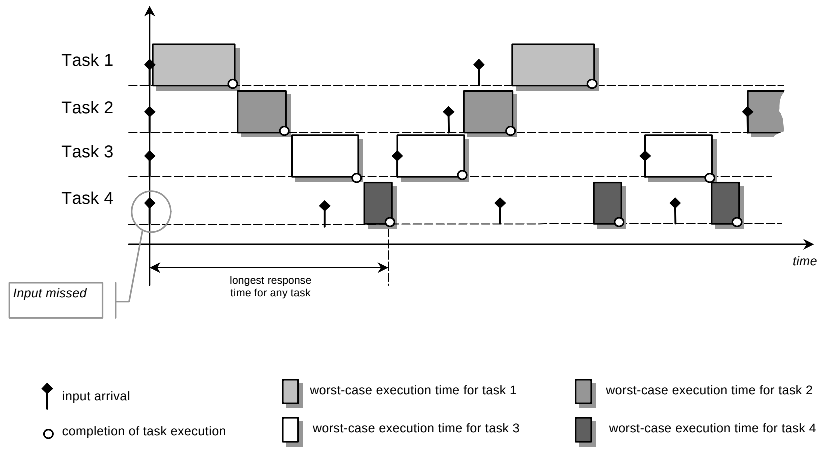

Task 1
Task 2
Task 3
Task 4
time
longest response time for any task
Input missed
input arrival
completion of task execution
worst-case execution time for task 1
worst-case execution time for task 2
worst-case execution time for task 3
worst-case execution time for task 4

Fig. 1.4 This figure illustrates an example of a system with 4 inputs, each one associated with a single control task. As inputs are handled in a FCFS basis, deadlines can not be guaranteed

This simple example brings to evidence that when timing requirements must be fulfilled, a FCFS approach may not be appropriate to deal with asynchronous interrupts. A solution is often to assign different priority levels to the inputs. For example, the smaller the arrival period, the higher is the priority level assigned to the related interrupt.

In this case, and assuming that tasks cannot be pre-empted, the longest response time for task 4 occurs if input 4 arrives to the computer system just after input 1, as the task related to input 1 has the longest execution time. Fig. 1.5 illustrates this situation.

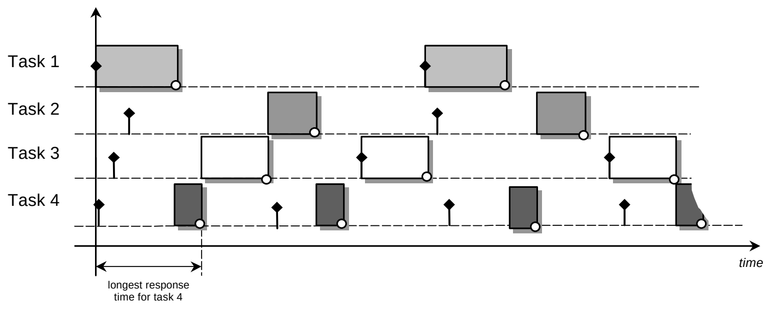


Fig. 1.5 This figure illustrates the same example of Fig. 1.4, where inputs are handled according to their priority level

In the previous example, we assumed that once a task starts its execution, it will proceed until completion, even if a higher-priority input arrives. However, most of the computer systems allow pre-emption. In a pre-emptable system, the processing of an input may be pre-empted at the arrival of a higher-priority input, and will only be resumed when there is no processing remaining at higher priorities. In these systems, the worst-case response time of the highest priority task corresponds to its worst-case execution time.

An alternative to the asynchronous interruption is the software polling of the inputs. In the case of inputs arriving through asynchronous interrupts, tasks are said to be event-triggered tasks. If the inputs arrive through software polling at specified intervals, the tasks responsible for reading the inputs are said to be time-triggered tasks. Most of the real-time computer-controlled systems need to encompass both types of tasks. For example, in a simple control loop for controlling temperature, the temperature sensor information may be cyclically polled by a timed-triggered task, whereas an alarm sensor may be connected to the computer system through an interrupt line which is handled by an event-triggered task.

Independently of the tasks' operating mode, the issue of guaranteeing real-time requirements is one of checking, prior to run-time, the feasibility of the system's task set; that is, checking if the worst-case execution time of the tasks is smaller than its admissible response time.

Obviously the reliability of such a feasibility test depends on some external factors such as the components of the operating system (operating system kernel, interrupt handling mechanisms, programming languages, synchronisation between tasks, resources shared by the tasks, etc.). This means, for instance, that the time for task switching must be considered. It also means that a careful evaluation of the worst-case execution times of the tasks must be made, as its knowledge is crucial for the development and analysis of real-time systems. It is also important to mention that an implicit requirement for a real-time system is that it must be dependable, since in the case of a system error, the timing requirements may not be guaranteed.

In this thesis we will not assess the dependability of the analysed systems, but we are aware that the proposed real-time approaches to guarantee real-time communications at the factory-floor level are only guaranteed if certain levels of dependability are provided to the computer system.

1.1.3. Real-Time Aspects in DCCS Systems

A distributed computer-controlled system is implemented by a set of computational devices. Each computational device runs a number of tasks. These tasks communicate their results by passing messages between computational devices across a field level communication network. In order to guarantee that the timing requirements of DCCS are met, the communication delay between a sending task queuing a message, and the related receiving task being able to access that message, must be upper bounded. This total delay is termed end-to-end communication delay (Tindell *et al.*, 1995), and is composed of the following four major components:

1. generation delay: time taken by the sender's task to generate and queue the related message;
2. queuing delay: time taken by the message to gain access to the field level communication network;
3. transmission delay: time taken by the message to be transmitted on the field level communication network;
4. delivery delay: time taken to process the message at the destination processor before finally delivering it to the destination task.

The queuing delay is a consequence not only of contention between message requests from the same network node, but also with message requests from other network nodes. The impact of the first factor in the overall queuing delay depends on the policy used to queue the messages, while the second factor depends on the behaviour and timing characteristics of the MAC protocol.

The worst-case response time of the distributed tasks must be evaluated considering the end-to-end communication delay, since its execution may involve more than one communication transaction.

Assume, for example, that in a controller, a task which reads a remote sensor, is cyclically executed. Two of the crucial operations of that task are sending a request to the remote node and receiving the related response. For this simple case, the response time for the task results from the concatenation of 9 components (Fig. 1.6).

The evaluation of end-to-end communication delay starts when the sending task is released, and starts competing against the other running tasks for the processor. The task suspends as soon as the message request is passed to the communications device (①). Then, the message request waits in a queue until it gains access to the communication medium. This queuing delay depends on how the queue is implemented (first-come-first-served queue, priority queue, etc.) and how the medium access control (MAC) behaves (②). The message request is then transmitted. This time interval depends on the data rate and length of the bus and also depends on the propagation delay (③).

The message indication is then queued in the remote communication device (④). The receiving task processes the message indication, and performs the actual reading of the required data. The response frame is produced and queued (⑤). The message response will suffer similar types of delays. A queuing delay in the remote transmitting queue (⑥), a transmission delay (⑦), a queuing delay in the local receiving queue (⑧), and finally the time for the local task to process the response (⑨).

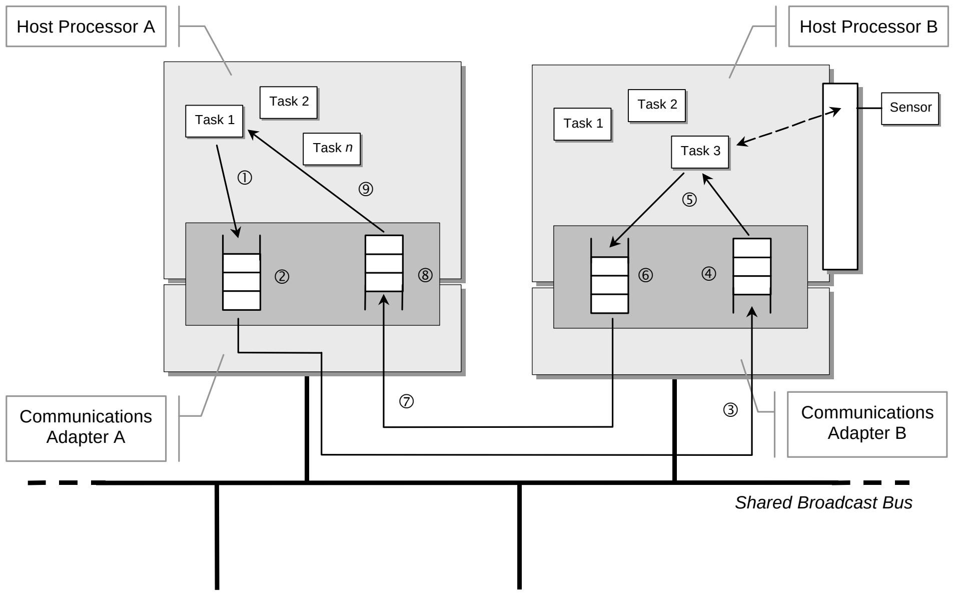


Fig. 1.6 This figure identifies the 9 components that contribute to the worst-case response time of a task in network node A (task 1). The role of this task is to obtain data from a process sensor located in a network node B

In terms of the response time analysis of tasks, distribution brings the need to include the end-to-end communication delays, as one of the components of the overall task's response time. In this thesis, we will essentially focus on the provision of methodologies for the evaluation of the worst-case messages' response times in fieldbus networks, as these are the foundation for the feasibility analysis of real-time distributed computer-controlled systems (DCCS).

1.2. Research Objectives

Broadcast local area networks aimed at the interconnection of sensors, actuators and controllers are commonly known as fieldbus networks. In the past, the scope of fieldbuses was dominated by vendor-specific solutions, which were mostly restricted to specific application areas. Moreover, the concepts behind each proposed fieldbus network were highly dependent on the manufacturer of the automation system. Each one had different technical implementations and claimed to fulfil different application requirements, or to fulfil the same requirements but with different technical solutions. More recently, vendor-independent standardised fieldbuses, supporting the open system concept, have started to be commonly used. Particular relevance must be given to the European Standard EN 50170 (Cenelec, 1996), which encompasses three widely used fieldbuses: P-NET (Pnet, 1994), PROFIBUS (Profibus, 1992) and WorldFIP (Afnor, 1990).

The main research objective of this thesis is to develop analysis and methodologies to guarantee, prior to run-time, that real-time distributed computer-controlled systems (DCCS) can be successfully implemented with standard fieldbus communication networks, such as those defined by the European Standard EN 50170.

1.3. Research Approach

A potential leap towards the use of fieldbus networks as the communication support of distributed computer-controlled systems lies on:

1. the evaluation of their MAC timing behaviour, since this is a crucial step to obtain analytical models enabling the evaluation of the messages' queuing and transmission delays;
2. the evaluation of worst-case response times of tasks in non pre-emptive contexts, since this analysis can be potentially adapted for the evaluation of message response times in communication networks.

1.4. Organisation of the Thesis

The remainder of this thesis is organised as follows. In Chapter 2 we survey some relevant results on the evaluation of the worst-case response time of tasks scheduled in single processor environments. We survey some relevant results for the priority-based schedulability analysis of real-time tasks, both for the fixed and dynamic priority assignment schemes. We give a special emphasis to the worst-case response time analysis in the non pre-emptive context, which is fundamental for the communication schedulability analysis.

In Chapter 3 we describe the main fieldbus network standards, and survey the most relevant previous work addressing their real-time characteristics. Particular relevance is given to the CAN (ISO 11898), P-NET, PROFIBUS and WorldFIP protocols.

Concerning the real-time characteristics of these network standards, there are some important results for the CAN protocol, which clearly characterises its real-time capabilities. Contrarily, the three protocols comprising the EN 50170 standard have not been devoted much analysis. This is particularly true for the P-NET and PROFIBUS cases. Consequently, the real-time characteristics of P-NET, PROFIBUS and WorldFIP will be especially focused within the following chapters of this thesis.

Hence, in Chapters 4, 5 and 6 we develop methodologies for the analysis and evaluation of the worst-case messages' response time in P-NET, PROFIBUS and WorldFIP networks, respectively.

Finally, in Chapter 7 we propose an approach for implementing priority-based scheduling mechanisms at the application process level of P-NET and PROFIBUS masters. A worst-case response time analysis is then provided, considering both the fixed and dynamic priority assignments, which demonstrate the relevance of the proposed approach.

The thesis concludes with Chapter 8, which summarises our contributions and suggests future work.

1.5. Main Contributions of this Thesis

The main contributions of this thesis are:

1. The worst-case response time analysis of P-NET networks, for both the cases of single-segment (Tovar and Vasques, 1998a; Tovar *et al.*, 1999a) and multiple-segment networks (Tovar *et al.*, 1998b);
2. The improvement of the worst-case response time analysis of PROFIBUS networks, for both the cases of unconstrained low-priority traffic (Tovar and Vasques, 1998c; Tovar and Vasques, 1998d) and constrained low priority-traffic (Tovar and Vasques, 1998e);
3. The accurate characterisation of the PROFIBUS token cycle time (Tovar and Vasques, 1999b);
4. A methodology for setting the WorldFIP bus arbitrator tables (Tovar and Vasques, 1999c), which is based in response time analysis considering both the rate monotonic (Tovar and Vasques, 1999d) and the earliest deadline first approaches (Tovar and Vasques, 1999e).
5. The improvement of the worst-case response time analysis of WorldFIP aperiodic buffer exchanges (Tovar and Vasques, 1999c; Tovar and Vasques, 1999f);
6. To show how the analysis for the worst-case response time of tasks in non pre-emptive contexts can be applied to P-NET (Tovar *et al.*, 1998f) and PROFIBUS (Tovar and Vasques, 1999g), if local scheduling mechanisms are used to improve their real-time capabilities.

1.6. References

Afnor (1990). Normes FIP NF C46-601 to NF C46-607. Union Technique de l'Electricité.

Cenelec (1996). General Purpose Field Communication System. EN 50170, Vol. 1/3 (P-NET), Vol. 2/3 (PROFIBUS), Vol. 3/3 (FIP), Cenelec.

Decotignie, J.-D. and Pleinevaux, P. (1993). A Survey on Industrial Communication Networks. In *Annales des Télécomunications*, Vol. 48, No. 9-10, pp. 55-63.

ISO 11898. (1993). Road Vehicle - Interchange of Digital Information - Controller Area Network (CAN) for High-Speed Communication. ISO.

Kopetz, H. (1997). Real-Time Systems: Design Principles for Distributed Embedded Applications. Kluwer Academic Publishers.

Pimentel, J. (1990). Communication Networks for Manufacturing. Prentice-Hall International Editions.

Pnet (1994). The P-NET Standard. International P-NET User Organisation ApS.

Prince, S. and Solomon, M. (1981). Communication Requirements for a Distributed Computer Control System. In *IEE Proceedings*, Vol. 128, No. 1, pp. 21-34.

Profibus (1992). PROFIBUS Standard DIN 19245 part I and II. Translated from German, Profibus Nutzerorganisation e.V.

Rembold, U., Nnaji, B. and Storr, A. (1993). Computer Integrated Manufacturing and Engineering. Addison-Wesley.

Tindell, K., Burns, A. and Wellings, A. (1995). Analysis of Hard Real-Time Communications. In *Real-Time Systems*, 9, pp. 147-171.

Tovar, E. and Vasques, F. (1998a). A Communication Support for Real-Time Distributed Computer Controlled Systems. In *Proceedings of the IEE International Workshop on Discrete Event Systems*, pp.178-183. Published by IEE.

Tovar, E., Vasques, F. and Burns, A. (1998b). Supporting Real-Time Distributed Computer-Controlled Systems with Multi-hop P-NET Networks. Polytechnic Institute of Porto, Technical Report HURRAY-TR-9813, September 1998, to appear in *Control Engineering Practice*, Pergamon Publishers.

Tovar, E. and Vasques, F. (1998c). Guaranteeing Real-Time Message Deadlines in PROFIBUS Networks. In *Proceedings of the 10th Euromicro Workshop on Real-Time Systems*, pp. 79-86, Published by IEEE Computer Society Press.

Tovar, E. and Vasques, F. (1998d). Real-Time Fieldbus Communications Using PROFIBUS Networks. Polytechnic Institute of Porto, Technical Report HURRAY-TR-9803, April 1998, to appear in *IEEE Transactions on Industrial Electronics*.

Tovar, E. and Vasques, F. (1998e). Setting Target Rotation Time in PROFIBUS Based Real-Time Distributed Applications. In *Proceedings of the 15th IFAC Workshop on Distributed Computer Control Systems*, pp. 1-6, Published by Pergamon, an Imprint of Elsevier Science.

Tovar, E., Vasques, F. and Burns, A. (1998f). Adding Local Priority-Based Dispatching Mechanisms to P-NET Networks: a Fixed Priority Approach. Polytechnic Institute of Porto, Technical Report HURRAY-TR-9822, December 1998, to appear in the *Proceedings of the 11th Euromicro Conference on Real-Time Systems*.

Tovar, E., Vasques, F. and Burns, A. (1999a). Communication Response Time in P-NET Networks: Worst-Case Analysis Considering the Actual Token Utilisation. Department of Computer Science, University of York, Technical Report YCS 312, January 1999, submitted for publication to the *Journal of Real-Time Systems*, Kluwer.

Tovar, E. and Vasques, F. (1999b). Cycle Time Properties of the PROFIBUS Timed Token Protocol. Polytechnic Institute of Porto, Technical Report HURRAY-TR-9811, August 1998, to appear in *Computer Communications*, Elsevier Science.

Tovar, E. and Vasques, F. (1999c). Distributed Computing for the Factory-Floor: a Real-Time Approach using WorldFIP Networks. Polytechnic Institute of Porto, Technical Report HURRAY-TR-9908, March 1999, submitted to *Computers in Industry*, Elsevier Science.

Tovar, E. and Vasques, F. (1999d). Factory Communications: on the Configuration of the WorldFIP Bus Arbitrator Table. Polytechnic Institute of Porto, Technical Report HURRAY-TR-9909, March 1999, submitted to *ETFA'99*.

Tovar, E. and Vasques, F. (1999e). Engineering Real-Time Applications with WorldFIP: Analysis and Tools. Polytechnic Institute of Porto, Technical Report HURRAY-TR-9912, April 1999.

Tovar, E. and Vasques, F. (1999f). Contributions for the Worst-Case Response Time Analysis fo Real-Time Sporadic Traffic in WorldFIP Networks. Polytechnic Institute of Porto, Technical Report HURRAY-TR-9910, March 1999, to appear in the *WIP session proceedings of the Euromicro RTS'99*.

Tovar, E. and Vasques, F. (1999g). From Task Scheduling in Single Processor Environments to Message Scheduling in a Profibus Fieldbus Network. In *Lecture Notes in Computer Science*, No. 1586, pp. 339-352.

Chapter 2

Schedulability Analysis of Tasks in Single Processor Systems: Review of Relevant Work

In this chapter we survey some relevant results for the priority-based schedulability analysis of real-time tasks, both for the fixed and for the dynamic priority assignment schemes. We give emphasis to the worst-case response time analysis in non pre-emptive contexts, since that analysis is of paramount importance to the message schedulability analysis in communication networks.

2.1. Introduction

Real-time computing systems are defined as those systems in which the correctness of the system depends not only on the logical result of computation, but also on the time at which the results are produced (Stankovic, 1988). There are various examples of real-time computing systems, such as command and control systems, flight control systems or robotics.

A typical real-time computing system has a real-time program running on the system, which reads inputs from input devices, processes these inputs, and often produces outputs to be sent to output devices. The time between the arrival of an input from a device and the completion of the processing for that input is called the response time for the device (Joseph and Pandya, 1986). The relative deadline for the device can be defined as the maximum interval between the instant of the input arrival and the completion of the processing for that input. Hence, the response time for a device must be smaller or equal to its relative deadline.

Assume that each input device is assigned a task (process) of the application program and that the tasks share a same processor. The problem of determining whether the system will meet its peak processing load, or in other words, whether no input from any device will be lost, becomes one of schedulability analysis of tasks (Burns, 1991).

A round-robin scheduling policy ensures that each task gets a share of the processor. However, such an approach may not be suitable for real-time systems. Assume the following example (Krishna and Shin, 1997):

"Consider a computer controlling an aircraft. Among its tasks are maintaining stability and keeping the cabin temperature within acceptable limits. Suppose the aircraft encounters turbulence that makes it momentarily unstable. The computer is then supposed to adjust the control surfaces to regain stability. If we use round-robin scheduling for this application, the computer may switch context partway through making the control adjustments in order to spend time making sure the cabin temperature is just right. The result may well be a crash, and the fact that the cabin is being maintained at optimum temperature will be scant consolation to the passengers as the airliner falls out the sky. What we want is to give the stability-maintenance task a very high priority, which ensures that when stability is threatened, all other interfering tasks are elbowed out of the way to allow this all-important task enough computer cycles."

It follows that the consideration of priority levels is crucial to a real-time computing system. If different inputs have different response time requirements, we need to consider different priority levels to schedule the related processing tasks. Consider a real-time system, within which several devices are connected at different priority levels to a single processor computer system. An input being processed, will be pre-empted when another input of higher priority arrives, and will only be resumed when there is no processing remaining at higher priorities.

Assume that the input from a device is saved in a buffer, until it is overwritten by the next input of the same device. The problem is to determine whether for a given assignment of priority levels, the system will meet its peak processing load (i.e. no input from any device will be lost). A more basic problem is how to assign devices to priorities in order to meet the system-processing load.

The remainder of this chapter is organised as follows. In Section 2.2 we outline some of the classic concepts of real-time systems. These aspects include the characterisation of the tasks and the description of the most commonly used priority assignment schemes. As throughout this thesis we will deal with offline schedulability analysis, in Section 2.3 we provide a brief comparison between the main two approaches for performing such schedulability analysis: based on the utilisation of the processor; based on the actual response time of the tasks. In Sections 2.4 and 2.5 we survey the most important results for the schedulability analysis of tasks in single processor systems, for the case of fixed and dynamic priority assignment, respectively. In both cases of priority assignment schemes, we present feasibility tests based on the utilisation of the processor and on the task's response time, and both for pre-emptive and non pre-emptive contexts.

2.2. Classical Concepts of Real-Time Systems

2.2.1. Characterisation of Tasks

In the previous section we mentioned that, in the simplest case, input devices produce inputs at regular intervals. However, in distributed computer-controlled systems (DCCS) not all devices operate in such manner. For example, some may have minimum and maximum time intervals between consecutive inputs, and others may even produce inputs at random intervals. As a consequence, tasks can be characterised according to their predictability. As it will be seen, this characteristic of the tasks affects their schedulability analysis.

Concerning the predictability, three basic types of tasks can be defined: periodic, aperiodic and sporadic.

Periodic tasks, as their name implies, are released on a regular basis. They are characterised by their period, their deadline and their required execution time per period.

The deadline is often assumed to be equal to the period (the processing of an input must be completed, at most, before the next input from the same device).

Aperiodic tasks are released only occasionally, and are usually triggered by an external event. To allow worst-case calculations to be made, a minimum period between any two aperiodic inputs (from the same device) is often defined. If this is the case, the task involved is said to be sporadic, and its period corresponds to its minimum inter-arrival time.

Tasks can also be characterised according to their criticality, depending on the consequences of not being executed before their deadlines. Concerning their criticality real-time tasks can be soft, hard or safety-critical real-time tasks.

Hard real-time tasks are those whose timely execution is critical. If deadlines are missed, severe faults may occur in the system. If the fault is catastrophic, the task is said to be a safety-critical real-time task. Time-utility functions are used in (Burns, 1991) to characterise the types of tasks (Fig. 2.1). For a hard real-time task, if the computation is completed before the deadline, the result will be fully useful; otherwise, it will not have any utility. For a safety-critical real-time task, if the computation is completed before the deadline, the result will be fully useful; otherwise it will have a negative utility.

In most large real-time systems, including DCCS, not all tasks will be hard or safety-critical. Some will even have no deadlines associated, and others will have merely soft deadlines. Soft real-time tasks are, as the name implies, not critical to the application. However, they do deal with time-varying data and hence the utility of result may diminish as the end of computation overpasses the deadline; but remain always positive.

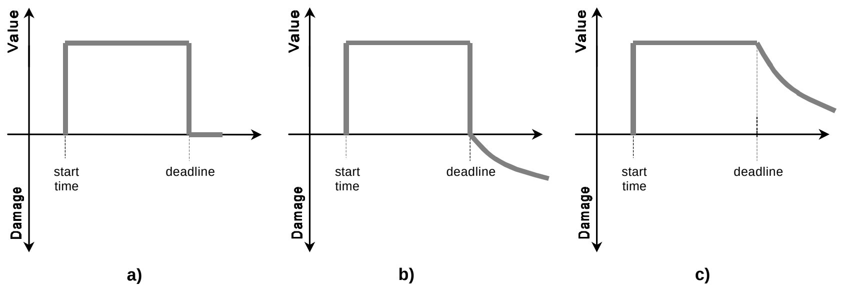


Fig. 2.1 a), b) and c) illustrate the time-utility function for a hard real-time task, a safety-critical real-time task and a soft-real time task, respectively

2.2.2. Scheduling Tasks in Real-Time Systems

Scheduling involves the allocation of time (and resources) to tasks, in such a way that timing requirements (or other performance requirements) are met. Scheduling has been perhaps the most widely research topic within real-time systems. As a consequence, there are multiple taxonomies for the scheduling schemes and for the methodologies for the schedulability analysis.

In a single processor computing system, a set of tasks shares a common resource: the processor. Schedulability analysis has to be performed to predict whether the tasks will meet their timing constraints.

The schedulability analysis can be performed online or offline (Fig. 2.2). In the first case the schedulability of the task set is analysed at run-time, whereas in the latter it is performed prior to run-time (pre-run-time schedulability analysis). In (Ramamritham and Stankovic, 1994) the authors use a different notation: dynamic and static scheduling, to denote systems that perform and do not perform the schedulability analysis at run-time, respectively. The offline scheduling has several advantages over the online scheduling: it requires little run time overhead and the schedulability of the task set is guaranteed before execution. However, it requires a prior knowledge of the tasks' characteristics, which fortunately is possible in most of real-time systems. If the tasks' characteristics are not known prior to run time, schedulability analysis must be performed online. There are basically two types of online schedulers (Ramamritham and Stankovic, 1994): planning-based and best-effort schedulers. In the former, when a new task arrives, the scheduler tries to re-define a new schedule, which is able to comply with both the requirements of the new task and the requirements of the previously scheduled tasks. The new task is only accepted for execution if the schedule is found feasible. In the latter, when a new task arrives, the scheduler does not try to perform a new schedule. The new task is accepted for execution, and the systems tries to do its best to meet deadlines. However, no guarantees are provided for the new coming task, as it may be aborted during execution.

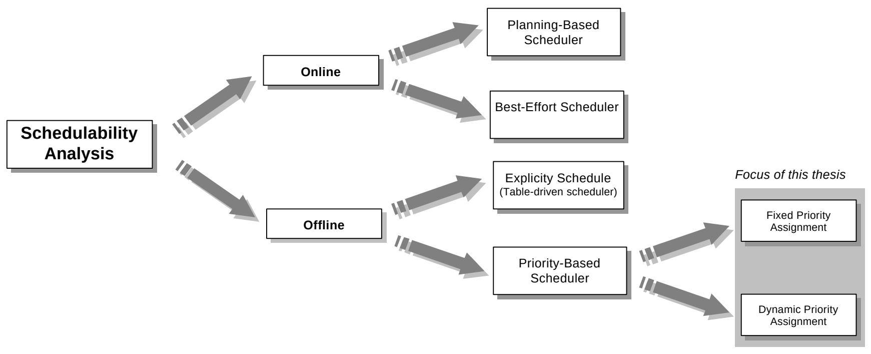


Fig. 2.2 This figure classifies some of the most important types of schedulability analysis

Two types of offline scheduling paradigms are also described in the literature, depending on whether the schedulability analysis produces itself a schedule (or plan) according to which tasks are dispatched at run-time. The table-driven approach (or cyclic executive) is the best known example of an offline scheduling that produces a schedule. The major drawback of the table-driven approach is that it imposes severe restrictions on the period of the tasks (Locke, 1992).

The priority-based approach is one example of offline scheduling where no explicit schedule is constructed. At run-time, tasks are executed in a highest-priority-first basis. Priority-based approaches are much more flexible and accommodating than table-driven approaches.

In the remainder of this chapter we will focus our attention on the offline scheduling paradigms, for tasks dispatched according to priority-based schemes. We assume the following notation (Burns and Wellings, 1996):

Table 2.1: Notation Used for the Schedulability Analysis of Tasks

Notation	***Description***
C	Worst-case computation time of the task
T	Minimum time between task releases (period)
D	Relative deadline of the task
P	Priority level assigned to the task
R	Worst-case response time of a task
U	Utilisation of the task (C/T)
N	Number of tasks in the system

2.2.3. Priority Assignment Schemes

One of the most used priority assignment schemes is to give the tasks a priority level based on its period: the smaller the period, the higher the priority; that is, $T_i < T_j \Rightarrow P_i > P_j$. This assignment is intuitively explained by the fact that more critical devices will provide inputs more frequently (via asynchronous interrupts), or will be polled more frequently. Thus, if they have smaller periods, their worst-case response time must also be smaller. This type of priority assignment is known as the rate monotonic (RM) assignment, and the related pre-run-time schedulability analysis was firstly introduced in (Liu and Layland, 1973).

If some of the tasks are sporadic, it may not be reasonable to consider the relative deadline equal to the period. A different priority assignment can then be to give the tasks a priority level based on its relative deadline: the smaller the relative deadline, the higher the priority; that is, $D_i < D_j \Rightarrow P_i > P_j$. This type of priority assignment is known as the deadline monotonic (DM) assignment (Leung and Whitehead, 1982).

In both RM and DM priority assignments, priorities are fixed, in the sense that they do not vary along time. At run-time, tasks are dispatched highest-priority-first. A similar dispatching policy can be used if the task, which is chosen to run, is the one with the earliest deadline. This also corresponds to a priority-driven scheduling, where the priorities of the tasks vary along time. Thus, the earliest deadline first (EDF) is a dynamic priority assignment scheme. Pre-run-time schedulability analysis for tasks dispatched according to the EDF assignment scheme was also introduced in (Liu and Layland, 1973).

In all three cases, the dispatching phase will take place either when a new task is released or the execution of the running task ends.

2.2.4. Pre-emptive and Non Pre-emptive Systems

In a priority-based scheduler, a higher-priority task may be released during the execution of a lower-priority one. If the tasks are being executed in a pre-emptive context, the higher-priority task will pre-empt the lower-priority one. Contrarily, in a non pre-emptive context, the lower-priority task will be allowed to complete its execution before the higher-priority task starts execution. This situation can be described as a priority inversion due to non pre-emption (a higher-priority task is delayed by a lower-priority one).

2.2.5. Characteristics of the Priority Assignment Schemes

The EDF priority assignment scheme has several advantages over the fixed priority assignment schemes (Spuri, 1996). A first advantage is that it always achieves higher processor utilisation, as was demonstrated in (Liu and Layland, 1973). Additionally, and contrarily to the RM or DM, EDF as been shown to be optimal when arbitrary deadlines are assumed, that is when the relative deadlines are allowed to be greater than the period of the tasks (Lehoczky, 1990).

On the other hand, fixed priority assignment schemes have some important advantages over EDF. Indeed, the EDF dispatching policy is computationally more demanding at run time. Although this aspect has more impact for the task scheduling in single processor environments, it should not be discarded for message scheduling in communication networks (Zuberi and Shin, 1995; Meshi *et al.*, 1996). Most hard real-time systems also have soft real-time components, which can execute at lower priority levels. In EDF, these tasks may occasionally delay execution of more stringent tasks (Sha *et al.*, 1991). Another important drawback of the EDF dispatching policy is its inability to deal with transient overloads (for instance due to exceptions or error recovery actions), since in such a situation some tasks may not meet their deadlines. Contrarily, with a fixed priority assignment approach, a subset of the more critical tasks would still be able to meet their deadlines. With an EDF approach, this is much more difficult to achieve (Buttazzo and Stankovic, 1993). At last, but not least, the analytical methods for computing worst-case response times are much more complex in the case of EDF, even though they are to be used offline.

Consider the following example (Table 2.2), which illustrates the differences between RM and EDF scheduling. We assume relative deadlines equal to the tasks' periods.

Table 2.2: Task Set Example A

Task	*Computation Time (C)*	*Period (T)*
A	35	80
B	10	55
C	5	20

Fig. 2.3 illustrates a time-line (Gantt chart) of the schedule for this task set, assuming that all of them share a common initial release time (at time instant 0), and the tasks are pre-emptable. In Fig 2.3, a) and b) represent the time-lines for a RM-based and an EDF-based schedule, respectively.

2.3. Approaches for the Pre-Run-Time Schedulability Analysis

Real-time computing systems with tasks dispatched according to a priority-based policy (we consider only RM/DM or EDF), must be tested a-priori in order to check if, during run time, no deadline will be lost. This feasibility test is called the pre-run-time schedulability analysis of the task set.

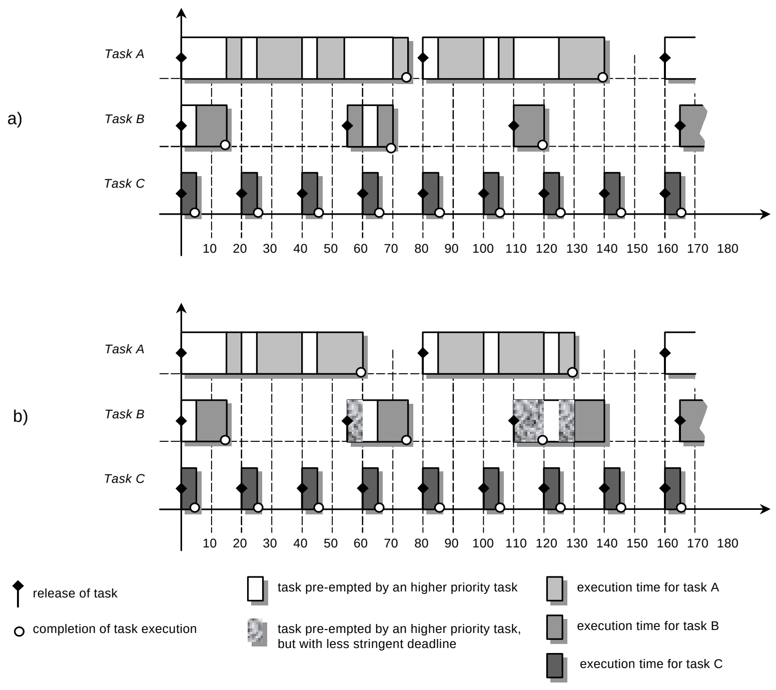


Fig. 2.3 This figure illustrates the schedule for the tasks characterised by Table 2.2, according to a) the RM priority assignment scheme, and b) the EDF priority assignment scheme. Note that in the EDF schedule task *B* is occasionally delayed by task *A*

It can be shown that for periodic tasks, a set of tasks is schedulable if and only if there is a feasible schedule for the LCM (least common multiple) of the periods (Lawler and Martel, 1981). Moreover, it can also be shown that if the tasks share a common request time (known as the critical instant), it is a pre-run-time schedulability sufficient condition that the tasks are schedulable for the longest of the periods (Liu and Layland, 1973). This suggests that a time-line could be used to perform the schedulability analysis. For instance, and concerning the example shown in Table 2.2, where the longest period is 80, Fig. 2.3 shows that the schedule generated by both RM and EDF schemes are feasible for the task set (if all the tasks share a common initial release time). However, time-line approaches may not be effective for systems with a large number of tasks. Hence, analytical methods are preferable.

There are mainly two types of analytical methods to perform pre-run-time schedulability analysis. One is based on the analysis of the processor utilisation. The other is based on the response time analysis for each individual task. In (Liu and Layland, 1973), the authors demonstrated that by considering only the processor utilisation of the task set, a test for the pre-run-time schedulability analysis could be obtained.

Contrarily, a response time test must be performed in two stages. First, an analytical approach is used to predict the worst-case response time of each task. The values obtained are then compared, trivially, with the relative deadlines of the tasks.

The utilisation-based tests have a major advantage: it is a simple computation procedure, which is applied to the overall task set. By this reason, they are very useful for implementing schedulers that check the feasibility online. However, utilisation-based tests have also important drawbacks, when compared with their response-time counterparts. They do not give any indication of the actual response times of the tasks. More importantly, and apart from particular task sets, they constitute sufficient but not necessary conditions. This means that if the task set passes the test, the schedule will meet all deadlines, but if it fails the test, the schedule may or may not fail at run-time (hence, there is a certain level of pessimism). It is also worth mentioning that the utilisation-based tests cannot be used for more complicated task models (Tindell, 1992).

In the next two sections, we survey the most relevant feasibility tests for task sets scheduled both with fixed and dynamic priority schemes, and for both pre-emptive and non pre-emptive contexts. Depending whether the tests are applied to the overall task set or individually to each task, they are classified as utilisation-based tests or response time tests, respectively.

2.4. Feasibility Tests: Case of the Fixed Priority Assignment

2.4.1. Basic Utilisation-Based Test

For the RM priority assignment, Liu and Layland introduced an utilisation-based pre-run-time schedulability test, which, when satisfied, guarantees that tasks will always be completely executed before their deadlines:

$$\sum_{i=1}^{N} \frac{C_i}{T_i} \le N \times \left(2^{1/N} - 1\right) \qquad \textbf{(2.1)}$$

This utilisation-based test is valid for periodic independent tasks, with relative deadlines equal to the period, and for pre-emptive systems. As mentioned in the previous section, typically the utilisation-based tests are sufficient but not necessary conditions. For instance, for the task set shown in Table 2.2, the test fails (0.87 < 0.78 is false), but the task set is schedulable, as can be seen by the time-line of Fig. 2.3a.

In (Lehoczky *et al.*, 1990), the authors provide an exact analysis, as given below[1]:

$$\min_{(k,l)\in R_i} \left\{ \sum_{j=1}^{i} \left(U_j \times \frac{T_j}{l \times T_k} \times \left\lceil \frac{l \times T_k}{T_j} \right\rceil \right) \right\} \le 1, \quad \forall_{i,\ 1 \le i \le n} \qquad \textbf{(2.2)}$$

where $R_i = \{(k,l)\}$ with $1 \le k \le i$ and $l = 1, ..., \lfloor T_i/T_k \rfloor$.

[1] The ceiling function $\lceil x \rceil$ is used to denote the smaller integer greater than or equal to *x*. Similarly, the floor function $\lfloor x \rfloor$ is used to denote the larger integer smaller than or equal to *x*.

It is clear that inequality (2.2) is not an easy to use utilisation-based test, hence loosing one of the advantages inherent to the more basic formulations: its simplicity.

2.4.2. Extended Utilisation-Based Tests

Formulations for the utilisation-based tests with deadlines smaller than periods are not available, to our best knowledge. It is however possible to formulate simple utilisation-based test for the case of non pre-emptive tasks.

In (Sha *et al.*, 1990), the authors update the basic utilisation based test (2.1) to include blocking periods, during which higher-priority tasks are blocked by lower-priority ones, to solve the problem of non-independence of tasks (for instance tasks that share resources which are protected by mutual exclusion):

$$\left(\sum_{i=1}^{i}\frac{C_i}{T_i}\right)+\frac{B_i}{T_i}\leq i\times\left(2^{1/i}-1\right),\ \ \forall_{i,\ 1\leq i\leq N} \qquad \textbf{(2.3)}$$

where B_i is the maximum blocking a task i can suffer (Sha *et al.*, 1990). Inequality (2.3) assumes that $P_{i+1} \leq P_i$, $\forall_{i<N}$; that is, tasks are ordered by decreasing priority.

In a non pre-emptive context, a higher-priority task can also be "blocked" by a lower-priority task. Assuming that the tasks are completely independent, the maximum blocking time a task can suffer is given by:

$$\begin{cases} B_i = 0, & \text{if } P_i = \min\limits_{j=1,..,N}\{P_j\} \\ B_i = \max\limits_{j\in lp(i)}\{C_j\}, & \text{if } P_i \neq \min\limits_{j=1,..,N}\{P_j\} \end{cases} \qquad \textbf{(2.4)}$$

where *lp(i)* denotes the set of lower-priority tasks (than task *i*).

Therefore, inequality (2.3) can be used as an utilisation-based test for a set of non pre-emptable but independent tasks, with the blocking for each task as given by (2.4). Moreover, accepting an increased level of pessimism, inequality (2.4) can be updated to an even simpler formulation:

$$\left(\sum_{i=1}^{N}\frac{C_i}{T_i}\right)+\max_{i,\ 1\leq i\leq N}\left\{\frac{B_i}{T_i}\right\}\leq i\times\left(2^{1/N}-1\right) \qquad \textbf{(2.5)}$$

Note that if all tasks have the same computation time, (2.5) considers that each task may be blocked at the rate of the highest-priority task.

2.4.3. Response Time Tests for the Pre-emptive Context

In (Joseph and Pandya, 1986) the authors proved that the worst-case response time R_i of a task i is found when all tasks are synchronously released (critical instant) at their maximum rate. R_i is defined as:

$$R_i = I_i + C_i \qquad \textbf{(2.6)}$$

where I_i is the maximum interference that task i can experience from higher-priority tasks in any interval $[t, t + R_i)$. The maximum interference (I_i) occurs, when all higher-priority tasks are released synchronously with task i (the critical instant). Without loss of generality, it can be assumed that all processes are released at time instant 0.

Consider a task j with higher-priority than task i. Within the interval $[0, R_i)$, it will be released $\lceil R_i/T_j \rceil$ times. Therefore, each release of task j will impose an interference of C_j. Hence, the overall interference is given by:

$$I_i = \sum_{j \in hp(i)} \left(\left\lceil \frac{R_i}{T_j} \right\rceil \times C_j \right) \qquad \textbf{(2.7)}$$

where $hp(i)$ denotes the set of higher-priority tasks (than task i). Substituting this value back in equation (2.2), the worst-case response time R_i of a task t_i is given by:

$$R_i = \sum_{j \in hp(i)} \left(\left\lceil \frac{R_i}{T_j} \right\rceil \times C_j \right) + C_i \qquad \textbf{(2.8)}$$

Equation (2.8) embodies a mutual dependence, since R_i appears in both sides of the equation. In fact all the analysis underlay this mutual dependence, since in order to evaluate R_i, I_i must be found, and vice-versa. The easiest way to solve such equation is to form a recurrence relationship (Audsley *et al.*, 1993):

$$W_i^{m+1} = \sum_{j \in hp(i)} \left(\left\lceil \frac{W_i^m}{T_j} \right\rceil \times C_j \right) + C_i \qquad \textbf{(2.9)}$$

The recursion ends when $W_i^{m+1} = W_i^m = R_i$, and can be solved by successive iterations starting from $W_i^0 = C_i$. Indeed, it is easy to show that W_i^m is non-decreasing. Consequently, the series either converges or exceeds T_i (in the case of RM) or D_i (in the case of DM). If the series exceeds T_i (or D_i), the task t_i is not schedulable.

2.4.4. Response Time Tests for the non Pre-emptive Context

In (Audsley *et al.*, 1993) the authors updated the analysis of Joseph and Pandya to include blocking factors introduced by periods of non pre-emption, due to the non-independence of the tasks. The worst-case response time is then updated to:

$$R_i = B_i + \sum_{j \in hp(i)} \left(\left\lceil \frac{R_i}{T_j} \right\rceil \times C_j \right) + C_i \qquad \textbf{(2.10)}$$

which may also be solved using a similar recurrence relationship. B_i is also as given by equation (2.4).

Some care must be taken using equation (2.10) for the evaluation of the worst-case response time of non pre-emptable independent tasks. In the case of pre-emptable tasks, with equation (2.8) we are finding the processor's level-i busy period preceding the completion of task i; that is, the time during which task i and all other tasks with a priority level higher than the priority level of task *i* still have processing remaining. For the case of non pre-emptive tasks, there is a slight difference, since for the evaluation of the processor's level-*i* busy period we cannot include task *i* itself; that is, we must seek the time instant preceding the execution start time of task *i*.

Therefore, equation (2.6) can be used to evaluate the task's response time of a task set in a non pre-emptable context and independent tasks, where the interference must be now re-defined:

$$I_i = B_i + \sum_{j \in hp(i)} \left(\left\lceil \frac{I_i}{T_j} \right\rceil \times C_j \right) \qquad \textbf{(2.11)}$$

Consider the following worst-case response time evaluation, assuming the task set shown in Table 2.2, for both pre-emptive and non pre-emptive contexts.

The worst-case response time of task *B* for the pre-emptive context (2.9) is:

$$W_B^0 = 10; \quad W_B^1 = \left(\left\lceil \frac{10}{20} \right\rceil \times 5 \right) + 10 = 15; \quad W_B^2 = \left(\left\lceil \frac{15}{20} \right\rceil \times 5 \right) + 10 = 15$$

Iterations stop at this point since $W_B^2 = W_B^1 = 15$, and thus $R_B = 15$, which coincides with the value given by the time-line (Fig. 2.2).

The worst-case response time of task *B* for the non pre-emptive context (2.11), considering that the blocking is equal to C_A, is:

$$W_B^0 = 35; \quad W_B^1 = 35 + \left(\left\lceil \frac{35}{20} \right\rceil \times 5 \right) = 45; \quad W_B^2 = 35 + \left(\left\lceil \frac{45}{20} \right\rceil \times 5 \right) = 50;$$

$$W_B^3 = 35 + \left(\left\lceil \frac{50}{20} \right\rceil \times 5 \right) = 50$$

Therefore, $R_B = 10 + 50 = 60$. This result shows that the task set example of Table 2.2 is not schedulable in a non pre-emptive context, as task *B* has a response time larger than its period.

Note also that a re-definition for the critical instant must be made. The maximum interference occurs when task *i* and all other higher-priority tasks are synchronously released just after the release of the longest lower-priority task (than task *i*).

2.5. Feasibility Tests: Case of the Dynamic Priority Assignment

2.5.1. Basic Utilisation-Based Test

For the EDF priority assignment, Liu and Layland also introduced an utilisation-based pre-run-time schedulability test (inequality (2.12)).

$$\sum_{i=1}^{N} \frac{C_i}{T_i} \le 1 \tag{2.12}$$

Similarly to the pre-run-time schedulability test for the RM case (2.1), this result is only valid for sets of non pre-emptive, independent, and periodic tasks, for which the relative-deadline is equal to the period.

Inequality (2.12) can easily be updated to include blocking periods due to the non-independence of the tasks. In (Baker, 1991), the author updated inequality (2.12) to:

$$\left(\sum_{i=1}^{i} \frac{C_i}{T_i} \right) + \frac{B_i}{T_i} \le 1, \quad \forall_{i,\ 1 \le i \le N} \tag{2.13}$$

where B_i is the maximum blocking a task i can suffer, considering the stack resource protocol (SRP). Inequality (2.13) assumes that $T_{i+1} \ge T_i$, $\forall_{i<N}$; that is tasks are ordered by decreasing period.

The key idea behind the SRP is that when a job needs a resource which is not available, it is blocked at the time it attempts to pre-empt, rather than later, when it actually may need the shared resource. This makes inequality (2.13) valid for sets of non pre-emptable tasks, dispatched according to the EDF scheme.

Similarly to the updating of (2.3) to (2.5), inequality (2.13) can be updated to a simpler (but more pessimistic) test:

$$\left(\sum_{i=1}^{N} \frac{C_i}{T_i} \right) + \max_{i,\ 1 \le i \le N} \left\{ \frac{B_i}{T_i} \right\} \le 1 \tag{2.14}$$

where B_i is now defined as:

$$B_i = \max_{j \ne i} \{ C_j \} \tag{2.15}$$

Another relevant result from (Baker, 1991) is that (2.13) can also be extended to task sets within which tasks can have relative deadlines smaller than periods:

$$\left(\sum_{i=1}^{i} \frac{C_i}{D_i} \right) + \frac{B_i}{D_i} \le 1, \quad \forall_{i,\ 1 \le i \le N} \tag{2.16}$$

As a corollary, inequality (2.12) can be extended for task sets within which $D_i \le T_i$:

$$\sum_{i=1}^{N} \frac{C_i}{D_i} \le 1 \tag{2.17}$$

These simple utilisation-based tests ((2.14) and (2.16)) are however quite pessimistic. Less pessimistic utilisation-based tests will now be addressed in Sections 2.5.2 and 2.5.3, for pre-emptive and non pre-emptive tasks, respectively. Later, in Sections 2.5.4 and 2.5.5, very recent results on response time analysis will be addressed, for pre-emptive and non pre-emptive tasks, respectively.

2.5.2. Extended Utilisation-Based Tests for the Pre-emptive Context

In (Zheng, 1993) the author extends the results of Liu and Layland in order to consider sporadic tasks, where inequality (2.12) is updated to:

$$\sum_{i=1}^{N} \left\lceil \frac{t - D_i}{T_i} \right\rceil^{+} \times C_i \le t,\ \forall_{t \ge 0} \qquad \textbf{(2.18)}$$

with $\lceil x \rceil^{+} = 0$ if $x < 0$. The proof for this inequality is intuitive. Assume that at time $t = 0$, there are no pending tasks. Then, a necessary condition to guarantee the tasks' deadlines is that the amount of time, T, needed to transmit all tasks generated during $[0, t]$ with absolute deadlines $\le t$, is not greater than t. Since the minimum inter-arrival time for a task i is T_i, there are at most $\lceil (t - D_i)/T_i \rceil^{+}$ requests for that task during $[0, t]$ with deadlines $\le t$. Those requests will need, at most, $\lceil (t - D_i)/T_i \rceil^{+} \times C_i$ time to be completed. Thus, the maximum value for T is given by $\sum_{i=1,\ldots,n}(\lceil (t - D_i)/T_i \rceil^{+} \times C_i)$. Note that if $D_i = T_i$, inequality (2.18) is satisfied if (2.12) is satisfied, since in this case $\lceil (t - T_i)/T_i \rceil^{+} \le t/T_i$.

This different formulation has advantages over (2.17), in the sense that it turns out to be a sufficient and a necessary condition (theoretically without any level of pessimism). However, inequality (2.18) can not be classified as a simple test (when compared to (2.17)). It has an additional problem, since it must be checked over an infinite continuous length interval $[0, \infty)$.

However, considering that expression $\sum_{i=1,\ldots,n}(\lceil (t - D_i)/T_i \rceil^{+} \times C_i)$ does only change at $k \times T_i + D_i$ time instants, inequality (2.18) does only need to be checked for these time instants. Consider the task set example given by Table 2.3.

Table 2.3: Task Set Example B

Task	*Computation Time (C)*	*Period (T)*	*Deadline (D)*	*Utilisation (U)*
A	30	80	60	0.375
B	10	40	40	0.250
C	5	25	15	0.200

For this task set example, the left-hand side of inequality (2.18) is plotted against its right-hand side (Fig. 2.4), and thus the task set is schedulable by the EDF priority assignment in a pre-emptive context.

Although the consideration of steps for the evaluation of inequality (2.19) eases its use, the problem still remains for the upper limit for t. Different authors have addressed this issue. It is possible to prove that if the total utilisation of the processor is ≤ 1 (condition (2.12)), it exists a point $t_{\max}$, such that $\sum_{i=1,\ldots,n}(\lceil (t - D_i)/T_i \rceil^{+} \times C_i) \le t$ always hold for $\forall_{t \ge t\max}$. Consequentely, inequality (2.18) can be re-written as follows:

$$\sum_{i=1}^{N} \left\lceil \frac{t - D_i}{T_i} \right\rceil^{+} \times C_i \le t,\ \forall_{t \in S},\ \text{with } S = \left(\bigcup_{i=1}^{N} \{ D_i + k \times T_i, k \in \aleph \} \right) \cap [0, t_{\max}) \qquad \textbf{(2.19)}$$

In (Baruah *et al.*, 1990a) and (Baruah *et al.*, 1990b) the authors demonstrated that t_{max} could be given by $(U/(1-U)) \times \max_{i=1,\ldots,N}\{(Ti-Di)\}$, where U represents the overall processor's utilisation $(\sum_{i=1,\ldots,N}(C_i/T_i))$. This result was further improved in (Ripoll *et al.*, 1996), where the upper limit for t is defined as $t_{max}=((\sum_{i=1,\ldots,N}(1-D_i/T_i) \times C_i)/(1-U)$.

Although this last formulation gives a smaller value for t_{max}, it still suffers from the same disadvantage: as the overall utilisation approaches 1, t_{max} becomes very large.

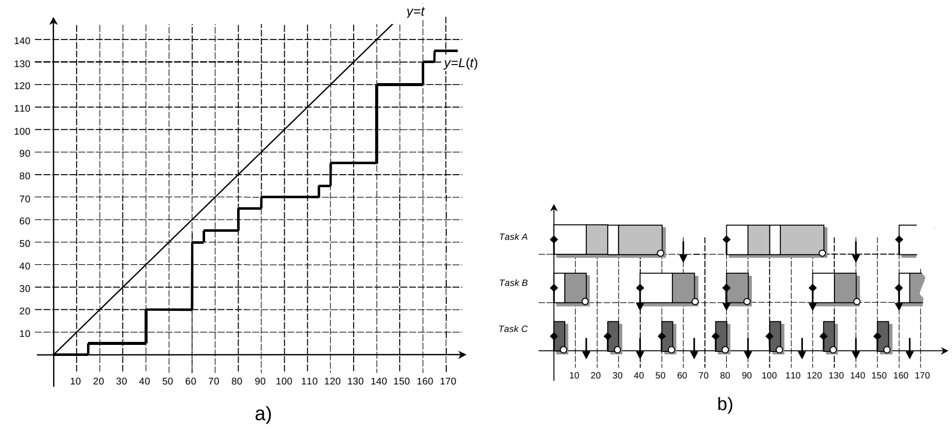


Fig. 2.4 This figure illustrates a time-line (b) for the synchronous *asap* release pattern of task set shown in Table 2.3. In a), the left-hand side of inequality (2.18), denoted as $L(t)$, is represented

For this reason, another approach is considered in (Rippoll *et al.*, 1996) and (Spuri, 1995), where the authors demonstrate that $t_{max} = L$ (synchronous processor's busy period). The synchronous processor's busy period is defined as the time interval from the critical instant up to the first instant when there are no more pending tasks in the system. For instance, for the time-line shown in Fig. 2.4b), $L = 65$. Analytically, L can be found as follows:

$$L = \sum_{i=1}^{N} \left\lceil \frac{L}{T_i} \right\rceil \times C_i \qquad \textbf{(2.20)}$$

Equation (2.20) is solved by recurrence, starting with $L^0 = \sum_{i=1,\dots,N} C_i$. When $L^{m+1} = L^m = L$, the solution has been found (note that this recurrence relationship converges if, and only if condition (2.12) is verified). For the task set of Table 2.3, it follows that:

$$L^0 = 45; \quad L^1 = \left\lceil \frac{45}{80} \right\rceil \times 30 + \left\lceil \frac{45}{40} \right\rceil \times 10 + \left\lceil \frac{45}{25} \right\rceil \times 5 = 30 + 20 + 10 = 60;$$
$$L^2 = \left\lceil \frac{60}{80} \right\rceil \times 30 + \left\lceil \frac{60}{40} \right\rceil \times 10 + \left\lceil \frac{60}{25} \right\rceil \times 5 = 30 + 20 + 15 = 65;$$
$$L^3 = \left\lceil \frac{65}{80} \right\rceil \times 30 + \left\lceil \frac{65}{40} \right\rceil \times 10 + \left\lceil \frac{65}{25} \right\rceil \times 5 = 30 + 20 + 15 = 65$$

and iterations stop, as $L^3 = L^2 = L = 65$.

2.5.3. Extended Utilisation-Based Tests for the non Pre-emptive Context

For the non pre-emptive context, a similar test was presented in (Zheng, 1993) and (Zheng and Shin, 1994):

$$\sum_{i=1}^{N} \left\lceil \frac{t - D_i}{T_i} \right\rceil^{+} \times C_i + \max_{j=1,\ldots,N} \{C_j\} \le t, \ \forall_{t \ge D_{\min}}, \ \text{ with } D_{\min} = \min_{j=1,\ldots,N} \{D_j\} \qquad \textbf{(2.21)}$$

Comparing to the test for the pre-emptive context (2.18), the inclusion of the blocking factor is intuitive (see Section 2.5.1.). However, in (George *et al.*, 1996) the authors discuss the pessimism inherent to the inequality (2.21). The main argument is that in this inequality it is considered that the cost of possible priority inversions is always initiated by the longest task and, moreover, it is effective during the entire interval under analysis. To reduce this level of pessimism, they suggest the following modification:

$$\sum_{i=1}^{N} \left\lceil \frac{t - D_i}{T_i} \right\rceil^{+} \times C_i + \max_{\substack{j=1,\ldots,N \\ D_j > t}} \{C_j\} \le t, \ \ \forall_{t \in S}, \ \ \text{ with } \max_{\substack{j=1,\ldots,N \\ D_j > t}} \{C_j\} = 0 \ \text{ if } \nexists_j : D_j > t \qquad \textbf{(2.22)}$$

That is, the blocking factor is only included if its deadline occurs after *t*.

Considering that the execution time of a task is expressed as a multiple of the system's tick, the blocking task must start its execution one tick before the critical instant. As a consequence, in the diverse formulations which have been including blocking factors due to the system's non pre-emptability, such blocking could be expressed as (C_i - 1).

2.5.4. Response Time Tests for the Pre-emptive Context

The worst-case response time analysis for pre-emptive EDF scheduling was first introduced in (Spuri, 1996). The starting point for such analysis was that the worst-case response time for a general task set is not necessarily obtained considering the critical instant, as defined for the fixed priority case. In his work, Spuri demonstrated that the worst-case response time of a task *i* is found in the processor's deadline-*i* busy period (analogous to the processor's level-*i* busy period in the case of fixed priorities). However, the longest processor's deadline-*i* busy period may occur when all tasks but task *i* (contrarily to the case of fixed priority assignment) are synchronously released and at their maximum rate.

This means that, in order to find the worst-case response time of task *i*, we need to examine multiple scenarios within which, while task *i* has an instance released at time *a*, all other tasks are synchronously released at time *t* = 0. As an example, consider the task set shown in Table 2.5.

Table 2.5: Task Set Example C

Task	*Computation Time (C)*	*Period (T)*	*Deadline (D)*	*Utilisation (U)*
A	1	4	4	0.250
B	2	6	6	0.333
C	2	9	9	0.222
D	2	15	15	0.133

Considering that all tasks are synchronously released at time instant 0, then a time-line is as shown in Fig. 2.5.

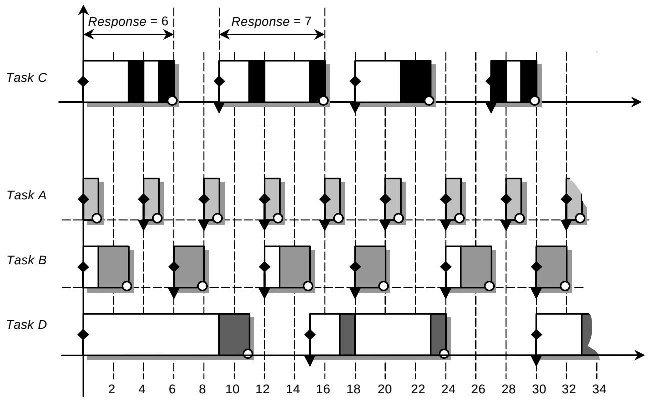


Fig. 2.5 This time-line illustrates the fact that, when evaluating the synchronous busy period, the worst-case response time does not occur for the first instance of task *C*

From Fig. 2.5 we can conclude that the instance of task *C* which is released at time instant $t = 9$ ($a = 9$) has a higher response time than the instance which is released at $t = 0$ ($a = 0$). Thus, given a value of *a*, the response of an instance of task *i*, which is released at time *a*, is:

$$R_i(a) = \max\{C_i, L_i(a) - a\} \qquad \textbf{(2.23)}$$

where $L_i(a)$ is the length of the deadline-*i* busy period, which starts at time instant $t = 0$. $L_i(a)$ can be evaluated by the following iterative computation:

$$L_i(a) = \sum_{\substack{j \neq i \\ D_j \leq a + D_i}} \left(\min\left\{ \left\lceil \frac{L_i(a)}{T_j} \right\rceil, 1 + \left\lfloor \frac{a + D_i - D_j}{T_j} \right\rfloor \right\} \times C_j \right) + \left(1 + \left\lfloor \frac{a}{T_i} \right\rfloor \right) \times C_i \qquad \textbf{(2.24)}$$

Equation (2.17) can be solved by recurrence, starting with $L_i^0(a) = 0$. When $L_i^{m+1}(a) = L_i^m(a) = L_i(a)$, the solution has been found. Obviously, in equation (2.24), the computational load only considers tasks that have deadlines earlier than D_i. Consider the task set example of Table 2.6.

Table 2.6: Task Set Example D

Task	*Computation Time (C)*	*Period (T)*	*Deadline (D)*
A	1	4	4
B	2	10	10

Consider that $a = 0$. At time instant $t = 9$, and for task *B*, the number of instances released for task *A* is 3 ($\lceil 9/4 \rceil = 3$). However, from those 3 releases, only the first 2 have absolute deadlines earlier than the deadline of task *B* (since $1 + \lfloor (10 - 4)/4 \rfloor = 2$). Assume again the scenario of Table 2.5. Using equation (2.24), with $a = 0$, the evaluation of the response time for task *C* is:

$$L_C^0(0)=0;\quad L_C^1(0)=0+(1+0)\times 2=2;$$
$$L_C^2(0)=(\min\{1,2\})\times 1+(\min\{1,1\})\times 2+(1+0)\times 2=5;$$
$$L_C^3(0)=(\min\{2,2\})\times 1+(\min\{1,1\})\times 2+(1+0)\times 2=6;$$
$$L_C^4(0)=(\min\{2,2\})\times 1+(\min\{1,1\})\times 2+(1+0)\times 2=6$$

Substituting this result back into equation (2.23), gives $R_C(0) = 6$. The same computation, but now $a = 9$ (also illustrated in the time-line given by Fig. 2.5) gives:

$$L_C^0(0)=0;\quad L_C^1(0)=0+(1+1)\times 2=4;$$
$$L_C^2(0)=(\min\{1,4\})\times 1+(\min\{1,3\})\times 2+(\min\{1,1\})\times 2+(1+1)\times 2=9;$$
$$L_C^3(0)=(\min\{3,4\})\times 1+(\min\{2,3\})\times 2+(\min\{1,1\})\times 2+(1+1)\times 2=13;$$
$$L_C^4(0)=(\min\{4,4\})\times 1+(\min\{3,3\})\times 2+(\min\{1,1\})\times 2+(1+1)\times 2=16;$$
$$L_C^5(0)=(\min\{4,4\})\times 1+(\min\{3,3\})\times 2+(\min\{2,1\})\times 2+(1+1)\times 2=16$$

Substituting this result back into equation (2.23), gives $R_C(9)=\max\{2, (16 - 9)\} = 7$, and thus it is now clear that the worst-case response time of a task *i* is not necessarily found when all tasks are synchronously released.

Finally, in the general case, the worst-case response time for a given task *i* is:

$$R_i = \max_{a\geq 0}\{R_i(a)\} \qquad \textbf{(2.25)}$$

The remaining problem is how to determine the values of *a*. Looking to the right-hand side of equation (2.24), we can easily understand that its value only changes at $k \times T_j + D_j - D_i$ steps.

$$a \in \bigcup_{j=1}^{N}\{k\times T_j + D_j - D_i, k\in \aleph_0\}\cap[0, L[\qquad \textbf{(2.26)}$$

with *L* as given by equation (2.20).

2.5.5. Response Time Tests for the non Pre-emptive Context

The worst-case response time analysis for the non pre-emptive EDF scheduling was introduced in (George *et al.*, 1996). The main difference from the analysis for the pre-emptive case is that a task instance with a later absolute deadline can possibly cause a priority inversion. Thus, and similarly to what was said for the fixed priority case (Section 2.4.4), instead of analysing the deadline-*i* busy period preceding the completion time of task *i*, we must analyse the busy period preceding the execution start time of the task's instance. Consequently, the response time of the t_i 's instance released at time *a* is:

$$R_i(a)=\max\{C_i, L_i(a)+C_i-a\} \qquad \textbf{(2.27)}$$

where $L_i(a)$ is now the length of the busy period (preceding execution).

Thus, $R_i(a)$ can be evaluated by means of the following iterative computation:

$$L_i(a) = \max_{D_j > a + D_i} \{C_j\} + \sum_{\substack{j \neq i \\ D_j \leq a + D_i}} \left(\min\left\{ 1 + \left\lfloor \frac{L_i(a)}{T_j} \right\rfloor, 1 + \left\lfloor \frac{a + D_i - D_j}{T_j} \right\rfloor \right\} \times C_j \right) + \left\lfloor \frac{a}{T_i} \right\rfloor \times C_i \qquad \textbf{(2.28)}$$

This equation may be solved also by recurrence. Note again that the blocking factor could be written as (C_i – 1). Note also that in order to analyse the busy period, the start of execution time, $1 + \lfloor L_i(a) / T_j \rfloor$, is used instead of $\lceil L_i(a) / T_j \rceil$.

2.6. Summary

In this chapter we provide a comprehensive survey of the most relevant results for the pre-run-time schedulability analysis of task sets in single processor systems.

The feasibility tests have been classified as utilisation-based tests and response time tests, according to the information which is provided by its evaluation; that is, in the former and indication is provided on the overall processor utilisation, while on the latter, the actual response time of each individual task is provided as a result.

Feasibility tests for fixed and dynamic priorities (both for the pre-emptive and non pre-emptive contexts) are provided (when available).

The emphasis is given to feasibility tests for non pre-emptable, independent task sets, within which tasks may have deadlines smaller than periods, as they will be the foundation of Chapter 7, where they will be adapted to encompass the characteristics of P-NET and PROFIBUS networks.

Finally, it is important to mention that this chapter is not an extended survey of all the important scheduling aspects, which could be pertinent to DCCS. The presented results are those strictly necessary as the background for the remaining chapters of this thesis. Just to mention some of the aspects which were not addressed in detail in this chapter, we can refer the problem of shared resources (Sha *et al.*, 1990; Rajkumar *et al.*, 1988), the problem of co-operative scheduling (Tindell, 1992; Tindell, 1994; Tindell and Clark, 1994; Palencia and Harbour, 1998), the problem of arbitrary deadlines (Lehoczky, 1990; Tindell and Clark, 1994; Palencia and Harbour, 1998) or the problem of finding the worst-case execution time of tasks (Puschner and Koza, 1989).

2.7. References

Audsley, N., Burns, A., Richardson, M., Tindell, K and Wellings, A. (1993). Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling. In *Software Engineering Journal*, Vol. 8, No. 5, pp. 285-292.

Baker, T. (1991). Stack-Based Scheduling of Real-Time Processes. In *Real-Time Systems*, Vol. 3, No. 1, pp. 67-99.

Baruah, S., Howell, R., Rosier, L. (1990a). Algorithms and Complexity Concerning the Pre-emptive Scheduling of Periodic Real-time Tasks on One Processor. In *Real-Time Systems*, 2, pp. 301-324.

Baruah, S., Mok, A. and Rosier, L. (1990b). Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor. In *Proceedings of the 11th Real-Time Systems Symposium (RTSS'90)*, pp. 182-190.

Burns, A. (1991). Scheduling Hard Real-Time Systems. In *Software Engineering Journal*, Special Issue on Real-Time Systems, May 1991, pp. 116-128.

Burns, A. and Wellings, A. (1996). Real-Time Systems and Programming Languages. Addison-Wesley, 2nd Edition.

Buttazzo, G. and Stankovic, J. (1993). RED: Robust Earliest Deadline Scheduling. In *Proceedings of the 3rd International Workshop on Responsive Computing Systems*.

George, L., Rivierre, N., Spuri, M. (1996). Preemptive and Non-Preemptive Real-Time Uni-Processor Scheduling. Technical Report No. 2966, INRIA.

Joseph, M. and Pandya, P. (1986). Finding Response Times in a Real-Time System. In *The Computer Journal*, Vol. 29, No. 5, pp. 390-395.

Krishna, C. and Shin, K. (1997). Real-Time Systems. McGraw-Hill Series in Computer Sciences.

Lawler, E. and Martel, C. (1981). Scheduling Periodically Occurring Tasks on Multiple Processors. In *Information Processing Letters*, Vol. 12, No. 1, pp. 9-12.

Lehoczky, J., Sha, L. Ding, Y. (1989). The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behaviour. In *Proceedings of the 10th IEEE Real-Time Systems Symposium*, pp. 166-171**.**

Lehoczky, J. (1990). Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines. In *Proceedings of the 11th IEEE Real-Time Systems Symposium*, pp. 201-209**.**

Leung, J. and Whitehead, J. (1982). On the Complexity of fixed-priority Scheduling of Periodic Real-Time Tasks. In *Performance Evaluation*, Vol. 22, No. 4, pp. 237-250.

Liu, C. and Layland, J. (1973). Scheduling Algorithms for Multiprograming in Hard-Real-Time Environment. In *Journal of the ACM*, Vol. 20, No. 1, pp. 46-61.

Locke, C. (1992). Software Architecture for Hard Real-Time Applications: Cyclic Executives vs. Fixed Priority Executives. In *Real-Time Systems*, Kluwer Academic Publishers, Vol. 4, No. 1, pp. 37-53.

Meshi, A., Natale, M. and Spuri, A. (1996). Earliest Deadline Message Scheduling with Limited Priority Inversion. In *Proceedings of the Workshop on Parallel and Distributed Real Time Systems*.

Puschner, P. and Koza, C. (1989). Calculating the Maximum Execution Time of Real-Time Programs. In *Real-Time Systems*, Vol. 1, No. 2, pp. 159-176.

Palencia, J. and Harbour, M. (1998). Schedulability Analysis for Tasks with Static and Dynamic Offsets. In *Proceedings of IEEE Real-Time Systems Symposium*, pp. 26-37.

Rajkumar, R., Sha, L., and J. Lehoczky (1988). Real-Time Synchronisation Protocols for Multiprocessors. In *Proceedings of the IEEE Real-Time Systems Symposium*, pp. 259-269.

Ramamritham, K. and Stankovic, J. (1994). Scheduling Algorithms and Operating Systems Support for Real-Time Systems. In *Proceedings of the IEEE*, Vol. 82, No. 1, pp. 55-67.

Ripoll, I., Crespo, A., Mok, A. (1996). Improvement in Feasibility Testing for Real-time Systems. In *Real-Time Systems*, 11, pp. 19-39.

Sha, L., Rajkumar, R. and J. Lehoczky (1990). Priority Inheritance Protocols: an Approach to Real-Time Synchronisation. In *IEEE Transactions on Computers*, Vol. 39, No. 9, pp. 1175-1185.

Sha, L., Klein, M. and Goodenough, J. (1991). Rate Monotonic Analysis for Real-Time Systems. Carnegie Mellon University, Technical Report CMU/SEI-91-TR-6 1991.

Spuri, M. (1995). Earliest Deadline Scheduling in Real-time Systems. PhD Thesis, Scuola Superiore Santa Anna, Pisa.

Spuri, M. (1996). Analysis of Deadline Scheduled Real-Time Systems. Technical Report No. 2772, INRIA.

Stankovic, J. (1988). Real-Time Computing Systems: the Next Generation. In *Tuturial: Hard Real-Time Systems*, Stankovic, J. and K. Ramamritham (Editors), IEEE Computer Society Press, Los Alamitos, USA, pp. 14-38.

Tindell, K. (1992). An Extendible Approach for Analysing Fixed Priority Hard Real-Time Tasks. Department of Computer Science, University of York, Technical Report YCS-189.

Tindell, K. (1994). Adding Time-Offsets to Schedulability Analysis. Department of Computer Science, University of York, Technical Report YCS-221.

Tindell, K. and Clark, J. (1994). Holistic Schedulability Analysis for Distributed Hard Real-Time Systems. In *Microprocessors and Microprogramming*, Vol. 40, pp. 117-134.

Zheng, Q. (1993). Real-Time Fault-Tolerant Communication in Computer Networks. PhD Thesis, University of Michigan.

Zheng, Q., Shin, K. (1994). On the Ability of Establishing Real-Time Channels in Point-to-Point Packet-Switched Networks. In *IEEE Transactions on Communications*, Vol. 42, No. 2/3/4, pp. 1096-1105.

Zuberi, K. and Shin, K. (1995). Non-Preemptive Scheduling of Messages on Controller Area Network for Real-Time Control Applications. In *Proceedings of Real-Time Technology and Applications Symposium*, pp. 240-249.

Chapter 3

Real-Time Communications with Fieldbus Networks: Analysis of Previous Relevant Work

In this chapter, we describe the main characteristics of relevant fieldbus networks, which directly target the support of distributed computer-controlled systems (DCCS). Particular relevance is given to CAN, P-NET, PROFIBUS and WorldFIP protocols. We will also survey some of the most relevant results concerning the ability of these fieldbus protocols to support real-time communications.

3.1. Introduction

Local area networks (LANs) are becoming increasingly popular in industrial computer-controlled systems. LANs allow field devices like sensors, actuators and controllers to be interconnected at low cost, using less wiring and requiring less maintenance than point-to-point connections (Lenhart, 1993). Besides the economic aspects, the use of LANs in industrial computer-controlled systems is also reinforced by the increasing decentralisation of control and measurement tasks, as well as by the increasing use of intelligent microprocessor-controlled devices. Broadcast LANs aimed at the interconnection of sensors, actuators and controllers are commonly known as fieldbus networks.

Similarly to other types of LANs, fieldbus networks are based on a layered structure derived from the seven-layer OSI model (Day and Zimmermann, 1983). However, due to the specialised requirements that must be met, the use of a full seven-layered architecture is precluded. Since transmission of states associated with sensors and actuators across the networks can be avoided, the network layer is not needed. The transport layer is also not needed, since as the network layer is not present, its most important functions (e.g., error control, and reliable data transfer with error recovery) can be performed by the data link and application layers, respectively. Similarly, the session layer is not needed, since its basic functions (e.g., process-to-process communications) can be performed by the application layer, and its more sophisticated functions (e.g., dialog synchronisation) are not needed in the context of fieldbus applications.

Consequently, a typical fieldbus network is based on a three-layered structure - physical layer, data link layer and application layer - even if some of these layers embody functionalities similar to those found in the other four layers of the OSI reference model.

There are multiple services and protocols that can be chosen for each of those three layers. The choice depends, essentially, on the original objectives of the fieldbus designers; that is:

1. either the fieldbus is to be merely a means to simplify the wiring between devices;
2. or the fieldbus is to be the backbone of a distributed real-time computing system.

These two different points of view about fieldbuses are one of the essential reasons for the proliferation of fieldbus systems (Thomesse, 1997). Other reasons relate to the lack of a unique and generic international standard. More than 30 product names or standards appeared, as the need for fieldbus has been felt in each industrial area, and since the beginning of the 80s, the international standardisation efforts have been trying to emerge in a sea populated by tens of already available products and services.

3.1.1. What Fieldbuses?

Relevant national fieldbus initiatives started in the beginning of the 80s: Factory Instrumentation Protocol (FIP), in France; Process Network (P-NET), in Denmark; Controller Area Network (CAN), in Germany; Process Field Bus (PROFIBUS) also in Germany.

In parallel, the standardisation efforts started at the international level, within the International Electrotechnical Commission (IEC). Several architectures were proposed for consideration. Some, like the MIL 1553B (Haverty, 1986), HART (Rosemount, 1991) or BITBUS (Intel, 1984) were already based on existing products, or at least prototypes. Others, like FIP or PROFIBUS, were, at the time, only paper proposals. All these proposals were based on different views on what should be a fieldbus. Therefore, till the end of the 80s no progress has been achieved at this level. In fact, only in 1993 the first (and, up to this moment, the only one) international standard has been agreed: the physical layer (IEC 1158-2, 1993).

At the national and regional levels, the standardisation has made more progress. P-NET is a Danish national standard since 1990 (DS 21906, 1990), PROFIBUS is a German standard since 1990 (DIN 19245, 1990), and FIP (later re-baptised as WorldFIP) is a French standard since 1989 (NF C46, 1989). Different technical options were taken for each of them.

As a consequence of the difficulty to achieve a truly international fieldbus standard, in 1995 the CENELEC (European Committee for Electrotechnical Standardisation) proposed an interim European standard, comprising the three national standards existing in Europe: P-NET, PROFIBUS and WorldFIP. This initiative led, in 1996, to the EN 50170 (EN 50170, 1996). Although this European standard is a set of not compatible profiles, it simplifies the choice in Europe, from several tens of fieldbus options down to three.

An additional proposal is being considered as a forth profile within the EN 50170: the Fieldbus Foundation. Fieldbus Foundation was formed in late 1994 from a merger of WorldFIP North America and the Interoperable Systems Project. In 1996, Fieldbus Foundation introduced its own specification (BSI DD 238, 1996), which is now being considered for the EN 50170.

Other committees within international standardisation bodies have been working to define other LAN technologies for specific application domains. Within ISO (International Organisation for Standardisation) and IEC, some of the more relevant are ISO TC72 (Textile Industry), ISO/IEC-JTC1 SC25 (Home Automation), IEC TC9 (Trains), ISO TC8 (Shipbuilding), ISO TC67 (Mineral-oil Industry), ISO TC82 (Mining Industry). At the European level, some of the more relevant are CEN TC247 (Building Automation) and CEN TC251 (Medical Domain and Hospitals).

Still, lots of different fieldbuses are being developed and sold. And some are also being standardised at the international level. For instance, Interbus-S (DIN 19258, 1995) and Actuator to Sensor Interface (ASI, 1996), among others, are being considered as new standards (EN 50254, 1996) for "High Efficiency Communications Subsystems for Small Data Packages". Also being considered is the PROFIBUS-PA (DIN 19245-4, 1996) and the Device WorldFIP (NF C46-638, 1996), simplified variants of PROFIBUS and WorldFIP, respectively. For instance, ASI does only aim to inter-connect Boolean-state devices, and its frame only supports a reduced number of data bits.

Other initiatives have experienced a different evolution. CAN is a success story. It was originally designed for use within road vehicles to solve cabling problems arising from the growing use of microprocessor-based components in vehicles. CAN was standardised by ISO (ISO 11898, 1993) has a "Road Vehicle - Interchange of Digital Information" system and since then it is a standard for the automotive applications, a domain area where VAN (ISO 11519, 1995) is a small contender. Due to its very interesting characteristics, CAN is also being considered for the automated manufacturing and distributed process control environments (Zuberi and Shin, 1997), and is being used as the communication interface in proprietary architectures, such as DeviceNet (Noonen *et al.*, 1994; Rockwell, 1997), which targets DCCS.

3.1.2. Fieldbuses for DCCS

When compared to other LANs, fieldbuses must fulfil different requirements. In (Pimentel, 1990) the author defines the following generic requirements to support DCCS.

1. Ability to handle very short messages in an efficient manner. Clearly, adding 60 bytes of overhead to every 2 bytes of information is not efficient at all.
2. Ability to handle periodic and aperiodic traffic. Periodic traffic is due to sample data, and aperiodic traffic is due to event conditions, such as a conveyor failure.
3. Bounded response times, to support both periodic traffic, and event-driven traffic, such as alarm messages.
4. No single point failure. The design should provide a minimum level of redundancy, to cover failure of devices, which may bring the network down (e.g., cables and master controllers).
5. Low network interface cost. This requirement implies serial communications to save on cable costs, and virtually all features of protocols and their implementations must be significantly simpler than in networks used at other levels in the automated manufacturing hierarchy.

In our opinion, P-NET, PROFIBUS and WorldFIP are the International Fieldbus Standards (EN 50170, 1996) which directly target DCCS. Standards from other domains are primarily aimed at the lower level functionality, particularly for remote I/O (EN 50254, 1996) or specific application areas like automotive (ISO 11898, 1993). In general they do not aim to cover DCCS applications. However, due to its characteristics, the CAN protocol, combined with an upper level protocol like DeviceNet is also suitable for some DCCS applications. Some other technologies, which do not meet the full requirements of a fieldbus, are being used in areas which do not need peer-to-peer communication and away from time-critical distributed control. Two very different examples of such fieldbuses are HART and Echelon/LONWorks (Echelon, 1993).

Therefore, in the remainder of this chapter we will focus our attention on the following four communication networks: CAN (Section 3.2), P-NET (Section 3.3), PROFIBUS (Section 3.4) and WorldFIP (Section 3.5). For each of these protocols, we describe their main characteristics and analyse some of the most relevant results concerning their ability to support real-time communications. As it will be highlighted, extensive response time analysis have only been already performed for the CAN protocol. Concerning the other three fieldbus protocols, fewer results are available, and thus the ability to support real-time communication with these three fieldbus protocols will be the main focus of this thesis.

3.2. Controller Area Network (CAN)

3.2.1. Main Characteristics of the CAN Protocol

The CAN protocol implements a priority-based bus with a carrier sense multiple access with collision avoidance (CSMA/CA) MAC. In this protocol any station can access the bus when the bus becomes idle. However, contrarily to Ethernet-like networks, the collision resolution is non-destructive, in the sense that one of the messages being transmitted will succeed. The collision resolution mechanism is very simple and is supported by the frame structure, namely by its twelve (or thirty, if the extended specification is used) leading bits, denoted as start bit and identifier fields (Fig. 3.1).

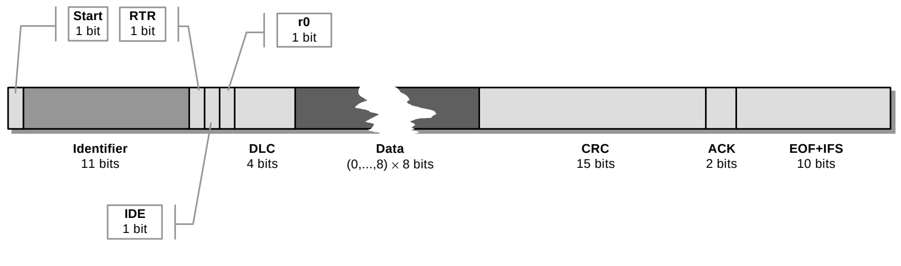


Fig. 3.1 This figure shows the structure of a CAN message. Although the identifier is said to have 11 bits, CAN 2.0B specification allows for 29 bits in this field. Description for the specific fields (start, RTR, IDE, r0, DLC, CRC, ACK and EOF+IFS) can be found in (ISO 11898, 1993)

This identifier field serves for two different purposes. On one hand it identifies a message stream in a CAN network: a temporal sequence of messages concerning, for instance, the remote reading of a specific process variable. On the other hand, it is a priority field, which enables the collision resolution mechanism to schedule the contending messages.

This collision resolution mechanism in CAN works as follows: when the bus becomes idle, every station with pending messages will start to transmit. Due to its open-collector nature, the CAN bus acts as AND-gate, where each station is able to read the bus status. During the transmission of the identifier field, if a station is transmitting a "1" and reads a "0", it means that there was a collision with at least one higher-priority message, and consequently this station aborts the message transmission. The highest-priority message being transmitted will proceed without perceiving any collision, and thus will be successfully transmitted. Obviously, each message stream must be uniquely identified.

To illustrate this collision resolution mechanism consider the following message stream set (Table 3.1) and the related collision resolution (Fig. 3.2).

Table 3.1: Message Stream Set Example

Message	*Identifier field*
A	01000111111
B	01000011111
C	01000001111
D	01000000111

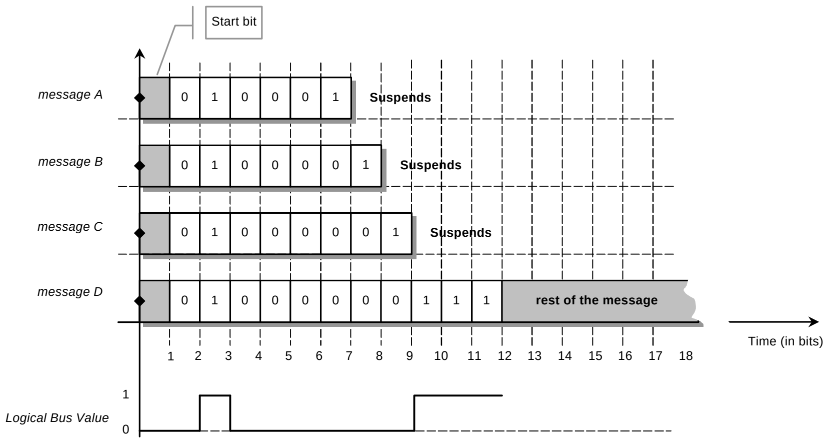


Fig. 3.2 This figure illustrates the collision resolution mechanism in CAN networks. The message *D* has the lowest binary identifier; that is, it is the highest-priority message on the bus

This collision resolution mechanism imposes that the different stations contending for the bus synchronously start transmitting their highest-priority pending message. It follows that this requirement brings strict limitations to the physical characteristics of the network: its bus length and its transmission data rate. For instance, considering a bus length of 40m, the maximum data rate is 1Mbps. Longer buses are only possible at the cost of a data rate reduction.

3.2.2. Real-Time Communications with CAN: Review of Relevant Work

The CAN network is based on a priority-bus. Therefore, the schedulability analysis of tasks in single processor systems can be easily adapted to the schedulability analysis of CAN messages.

In (Tindell *et al.*, 1994; Tindell *et al.*, 1995), the authors addressed in detail the analysis of real-time communications in CAN, assuming fixed priorities for message streams. In such case, the worst-case response time of a queued message, measured from the release of the queuing task to the time the message is fully transmitted, is:

$$R_m = J_m + I_m + C_m \tag{3.1}$$

This equation is analogous to equation (2.6). J_m is the queuing jitter of message stream S_m, inherited from the worst-case response time $R_{sender(m)}$ (where *sender*(*m*) denotes the task which queues the message *m*). The term I_m represents the worst-case queuing delay - longest time between placing the message in the priority-ordered outgoing queue, and the start of the message transmission.

The deadline monotonic (DM) priority assignment can be directly implemented in a CAN network, by setting the identifier field of each message stream to a unique priority, according to the DM rule. Therefore, by analogy with equation (2.11):

$$I_m = B_m + \sum_{\forall j \in hp(m)} \left(\left\lceil \frac{I_m + J_j + t_{bit}}{T_j} \right\rceil \times C_j \right) \tag{3.2}$$

where B_m is the worst-case blocking factor, which is equal to the longest time taken to transmit a lower priority message, and given by:

$$B_m = \max_{\forall k \in lp(m)} \{0, C_k\} \tag{3.3}$$

The set *lp*(*m*) is the set of message streams with lower-priority than message stream S_m (see equation (2.7)). t_{bit} is the time taken to transmit a bit on the bus and *hp*(*m*) is the set of message streams in the system with higher-priority than the message stream S_m.

C_m is the longest time taken to transmit a message from stream S_m. CAN has a 47 bit overhead per message, and a stuff width of 5 bits. Only 34 of the 47 bits of overhead are subject to stuffing, so C_m can be defined as:

$$C_m = \left(\left\lfloor \frac{34 + 8 \times db_m}{5} \right\rfloor + 47 + 8 \times s_m \right) \times t_{bit} \tag{3.4}$$

where db_m is the number of data bytes in the message.

An alternative for the fixed priority assignment are the dynamic priority schemes, such as the non pre-emptive earliest deadline first (EDF). In (Zuberi and Shin, 1995; Zuberi and Shin, 1997), the authors analyse how the EDF could be used to schedule CAN messages. In these works, the authors propose the use of a mixed traffic scheduler (MTS), which attempts to give a high utilisation (like EDF) while using the standard 11-bit format for the identifier field.

The goal of the MTS scheduler is to make the identifier fields of different message streams to reflect the deadlines of messages. However, considering that each message must have a unique identifier field (which is a requirement of CAN), they suggested the division of the identifier field into three sub-fields, as shown in Fig. 3.3.

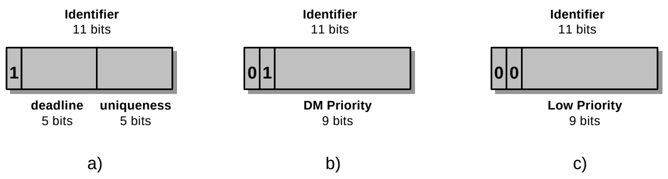


Fig. 3.3 This figure illustrates the proposed structure for the identifier field. a) is for the messages that are to be scheduled according to the EDF. b) is for messages to be scheduled according to the DM, and finally c) is for low-priority messages. The first bit of the identifier field ensures that higher priority messages are scheduled according to the EDF (note that Zuberi and Shin assume a Wired-OR bus, thus '1' being the dominant bit)

For the higher-priority message, the deadline field is derived from the deadline of the message. To deal with the case where two messages have the same deadline, the one with the highest uniqueness code will win (note that Zuberi and Shin assume a Wired-OR bus, thus '1' being the dominant bit). The uniqueness code also serves to identify the message for reception purposes. To encode the deadline field, the authors solved the two following problems.

1. The first is that the remaining slack time of a message changes with every clock tick. This requires identifiers of all messages to be continually updated, and also that each local clock must be synchronised.
2. The second is that, in a typical system, message streams may have largely different deadlines, which rises a problem with the length of the identifier field (only 5 bits to encode the deadline).

To solve the second problem, the authors divide time into regions and encode deadlines according to which region they fall in. Deadlines are then expressed relatively to a periodically increasing reference called the start of epoch (SOE). Fig. 3.4 illustrates this concept for a m = 2 case.

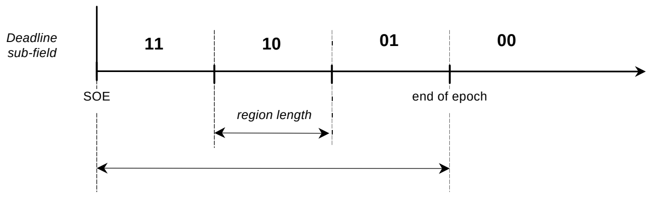


Fig. 3.4 This figure illustrates the value of the deadline field, considering that it is encoded in only two bits

For the pre-run-time schedulability analysis of the EDF traffic, analytical expressions described in Sections 2.5.3 and 2.5.5 can be used.

To our best knowledge, these are the most relevant works on how to guarantee real-time communications using CAN networks.

3.3. Process Network (P-NET)

3.3.1. Main Characteristics of the P-NET Protocol

P-NET is a multi-master standard based on a virtual token-passing (VTP) scheme. In P-NET all communication is based on a message cycle principle, where a master sends a request and the addressed slave immediately returns a response. Fig. 3.5 illustrates the hybrid-operating mode of the P-NET's MAC.

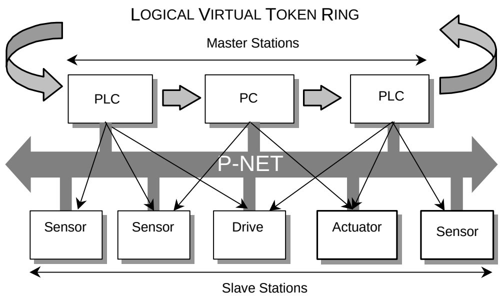


Fig. 3.5 Token passing and master-slave procedures in P-NET networks

The P-NET standard uses a data rate of 76800bps. This data rate resulted from weighing up the conflicting requirement for data to be transported as fast as possible, but not at such speed as to negate the use of standard microprocessor UARTS, or restrict the usable distance or cable type (Jenkins, 1997).

The VTP scheme is implemented using two protocol counters. The first one, the access counter (AC), holds the node address of the currently transmitting master. When a request has been completed and the bus has been idle for t = 40 bit periods (520μs at 76,8Kbps), each one of the access counters is incremented by one. The master whose access counter value equals its own unique node address is said to be holding the token, and is allowed to access the bus. When, as the access counter is incremented, it exceeds the “maximum number of masters”, the access counter in each master is reset to one. This allows the first master in the cycling chain to gain access again.

The second counter, the idle bus bit period counter (IBBPC), increments for each inactive bus bit period. Should any transactions occur, the counter is reset to zero. As explained above, when the bus has been idle for 40 bit periods following a transfer, all the access counters are incremented by one, and the next master is thus allowed to access the bus.

If a master has nothing to transmit (or indeed is not even present), the bus will continue to be inactive. Following a further period of s = 10 bit periods (130μs), the idle bus bit period counter will have reached 50, (60, 70,…) so all the access counters will be incremented again, allowing the next master access. The virtual token passing will continue every 10 bit periods, until a master does require access.

The P-NET standard allows each master to perform at most one message cycle per token visit. This is an important idea for the timing analysis of P-NET MAC mechanisms.

After receiving the token, the master must transmit a request before a certain time has elapsed. This is denoted as the master's reaction time, and the standard imposes a worst-case value of up to r = 7 bit periods. A slave is allowed to access the bus between 11 and 30 bit periods after receiving a request, measured from the beginning of the stop bit in the last byte of the request frame. The maximum allowed delay is then 30 bit periods (390μs). This delay is denoted as the slave's turnaround time. To illustrate these basic MAC procedures and the notation used, please refer to Fig. 3.6.

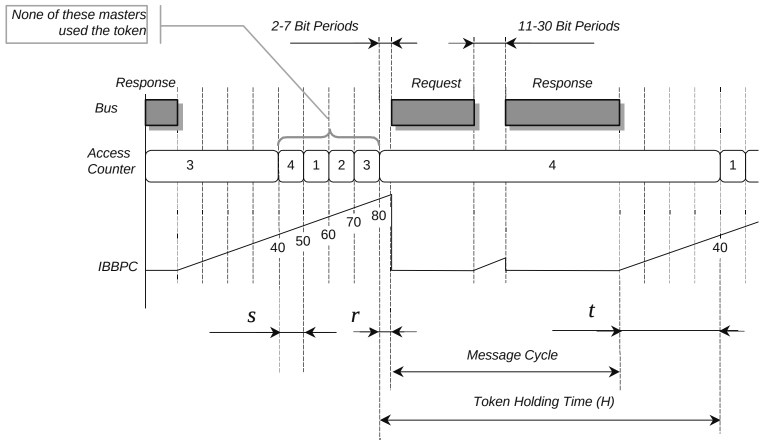


Fig. 3.6 This figure illustrates the concepts of message cycle, token holding time (H), slave's turnaround time, master's reaction time (r), idle token time (s) and token passing time (t)

It is also important to understand the idea of a P-NET message cycle length. A P-NET frame (Fig. 3.7) contains five fields: node address field (2 bytes); control/status field (1 byte); information length field (1 byte); information field (0-63 bytes); error detection field (1-2 bytes). The node address field may have up to 24 frame bytes. P-NET uses these complex addresses if special devices (P-NET hopping devices) are used to relay frames between different segments.

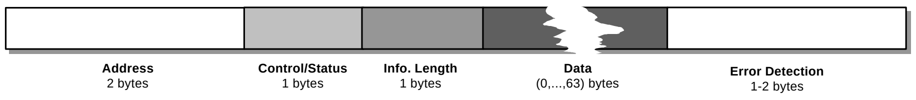


Fig. 3.7 This figure illustrates the structure of a P-NET frame. Although the address field is represented with only 2 bytes, it can go up to 24 frame bytes

As each frame byte in P-NET actually corresponds to 11 bits, a frame may have up to 759 bits (69×11 bits). In P-NET all the frame bytes are sent asynchronously, with one start bit (logical zero), 8 data bits (with LSB first), one address/data bit and one stop bit. Within a frame, a start bit must immediately follow a stop bit.

Thus, considering the case where both the request and response frames have 759 bits (realistically it is more likely that either the request will be longer, in cases of data being sent to a slave, or the response will be longer, in cases of data being received from a slave), the overall sum for the longest message cycle is 1548 bit periods, corresponding to 20.15ms at 76800bps. This includes the worst-case slave's turnaround time (30 bit periods). Table 3.2 gives the worst-case duration of the token holding time in P-NET, with the explicit weight of the different contributing components.

Table 3.2: Worst-Case Duration for the Token Holding Time

Component	**Worst-Case Duration of Token Holding Time** (in bit periods)
Master's Reaction Time	7
Request Transmission Time	759
Slave's Turnaround Time	30
Response Transmission Time	759
Token Passing	40
Total	**1595**

3.3.2. Real-Time Communications with P-NET: Review of Relevant Work

To our best knowledge, there is no previous relevant work on how to support real-time communications with P-NET networks.

3.4. PROcess FIeld BUS (PROFIBUS)

3.4.1. Main Characteristics of the PROFIBUS Protocol

The PROFIBUS MAC protocol is based on a token passing procedure used by master stations to grant the bus access to each other, and a master-slave procedure used by master stations to communicate with slave stations. Fig. 3.5 can also be used to illustrate such hybrid MAC protocol. The PROFIBUS token passing procedure uses a simplified version of the Timed-token protocol (Grow, 1982).

These MAC procedures are implemented at the layer 2 of the OSI reference model, which, in PROFIBUS, is called Fieldbus Data Link (FDL). In addition to controlling the bus access and the token cycle time (a feature that will be later explained), the FDL is also responsible for the provision of data transmission services for the FDL user (e.g., the application layer).

PROFIBUS supports four data transmission services: Send Data with No acknowledge (SDN); Send Data with Acknowledge (SDA); Request Data with Reply (RDR) and Send and Request Data (SRD).

The SDN is an unacknowledged service used for broadcasts from a master station to all other stations on the bus. Conversely, all other transmission services are based on a real dual relationship between the initiator (master station holding the token) and the responder (slave or master station not holding the token). An important characteristic of these services is that they are immediately answered (as in P-NET networks), with a response or an acknowledgement. This feature, also called "immediate-response", is particularly important for the real-time bus operation.

In addition to these services, industrial applications often require the use of cyclical transmission methods. A FDL-controlled polling method (cyclical polling) may be used to scan field devices, such as sensors or actuators. PROFIBUS enables a poll list to be created at the FDL layer, allowing the execution of cyclical polling services based on RDR and SDR services.

An important PROFIBUS concept is the message cycle. A message cycle consists of a master's action frame (request or send/request frame) and the associated responder's acknowledgement or response frame. User data may be transmitted in the action frame or in the response frame. The acknowledgement or response must arrive within a predefined time, the slot time, otherwise the initiator repeats the request. At the network set-up phase, the maximum number of retries, before a communication error report, must be defined in all master stations. The PROFIBUS real-time analysis presented in the following section is based on the knowledge of the message cycle duration.

One of the main functions of the PROFIBUS MAC is the control of the token cycle time. After receiving the token, the measurement of the token rotation time begins. This measurement expires at the next token arrival and results in the real token rotation time (T_{RR}). A target token rotation time (T_{TR}) must be defined in a PROFIBUS network. The value of this parameter is common to all masters, and must be chosen small enough to meet the responsiveness requirements of all masters. When a station receives the token, the token holding time (T_{TH}) timer is given the value corresponding to the difference, if positive, between T_{TR} and T_{RR}.

In PROFIBUS there are two main categories of messages: high-priority and low-priority. These two categories of messages use two independent outgoing queues. If at the arrival, the token is delayed, that is, the real token rotation time (T_{RR}) was greater than the target token rotation time (T_{TR}), the master station may execute, at most, one high-priority message cycle. Otherwise, the master station may execute high-priority message cycles while $T_{TH} > 0$. T_{TH} is always tested at the beginning of the message cycle execution. This means that once a message cycle is started it is always completed, including any required retries, even if T_{TH} expires during the execution. We denote this occurrence as a T_{TH} overrun. The low-priority message cycles are executed if there are no high-priority messages pending, and while $T_{TH} > 0$ (also evaluated at the start of the message cycle execution, thus leading to a possible overrun of T_{TH}).

Apart from distinguishing high and low-priority message cycles, the PROFIBUS MAC differentiates three subtypes of low-priority message cycles: poll list, non-cyclic low-priority (application layer and remote management services) and Gap List message cycles. The Gap is the address range between two consecutive master addresses, and each master must periodically check the Gap addresses to support dynamic changes in the logical ring.

After all high-priority messages have been carried out, poll list message cycles are started. If the poll cycle is completed within T_{TH}, the requested low-priority non-cyclical messages are then carried out, and a new poll cycle starts at the next token arrival with available T_{TH}. If a poll cycle takes several token visits, the poll list is processed in segments, without inserting requested low-priority non-cyclical messages. Low-priority non-cyclical message cycles are carried out only at the end of a complete poll cycle. At most one Gap address is checked per token visit, if there is still available T_{TH}, and there are no pending messages. Fig. 3.8 synthesises the PROFIBUS MAC procedures.

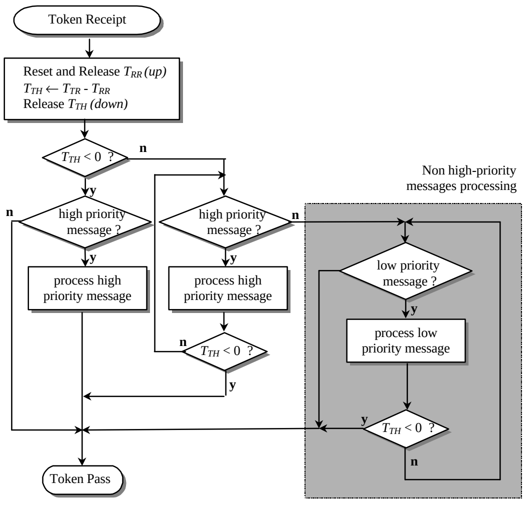

Token Receipt
Reset and Release T_{RR} (up)
$T_{TH} \leftarrow T_{TR} - T_{RR}$
Release T_{TH} (down)
$T_{TH} < 0$?
n
y
high priority message ?
process high priority message
Non high-priority messages processing
low priority message ?
process low priority message
Token Pass

Fig. 3.8 This figure illustrates how PROFIBUS masters handle the two types of basic traffic

To illustrate the token passing mechanisms between the *n* master stations (n = 4) please refer to Fig. 3.9, where the i^{th} real token rotation time, as seen by master 4 (T^4_{RR}), corresponds to the time of the network token rotation (none of the stations used the token to transmit messages). At that i^{th} token visit, master 4 uses part of its available token holding time (T^4_{TH}) to transmit two message cycles.

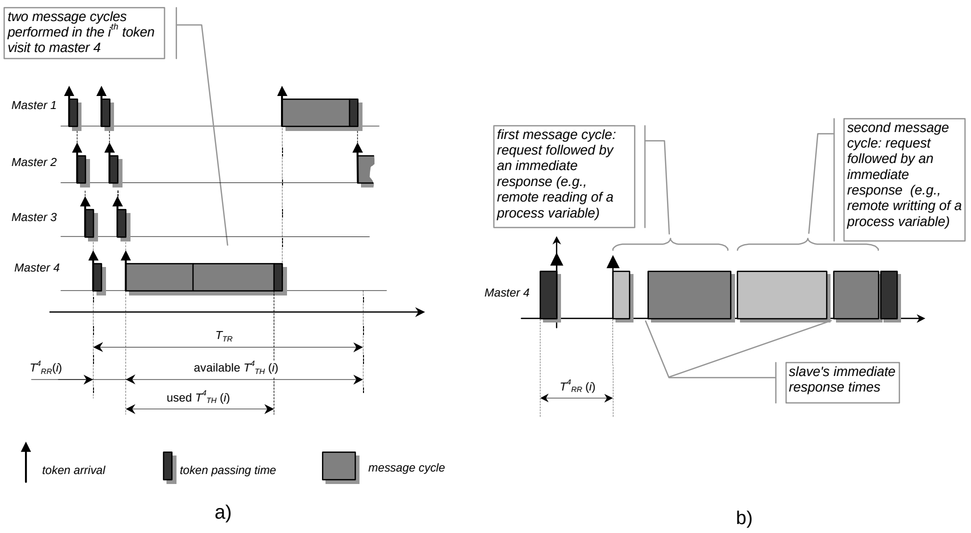

two message cycles performed in the i^{th} token visit to master 4
Master 1
Master 2
Master 3
Master 4
T_{TR}
$T^4_{RR}(i)$
available T^4_{TH} (i)
used T^4_{TH} (i)
token arrival
token passing time
message cycle
a)
first message cycle: request followed by an immediate response (e.g., remote reading of a process variable)
second message cycle: request followed by an immediate response (e.g., remote writting of a process variable)
Master 4
T^4_{RR} (i)
slave's immediate response times
b)

Fig. 3.9 Example of token usage. b) details the message cycles transmitted by master 4 in a)

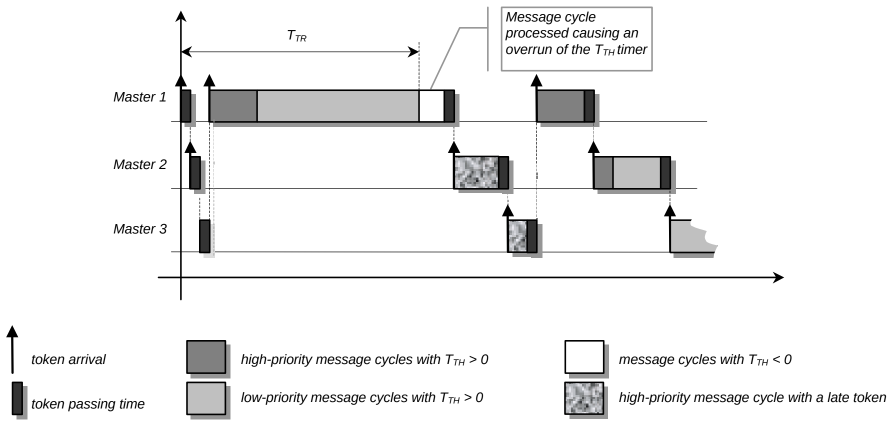
3.4.2. Real-Time Communications with PROFIBUS: Review of Relevant Work

Compared to the timed-token protocol (Grow, 1982), the main difference in the PROFIBUS token passing consists in the absence of synchronous bandwidth allocation (H_i). For the timed-token protocol this is a relevant station parameter, since it specifies the amount of time a station has to transfer its synchronous (real-time) traffic.

In PROFIBUS, the absence of synchronous bandwidth allocation prevents the use of the traditional real-time analysis for the timed-token protocol. In fact, real-time solutions for networks based on the timed-token protocol, such as (Agrawal *et al.*, 1994; Zheng and Shin, 1995), for FDDI networks (ISO 9314-2, 1989) or (Montuschi *et al.*, 1992) for the IEEE 802.4 token bus (IEEE 802.4, 1985), rely on the possibility of allocating specific bandwidth for the real-time traffic.

These results cannot however be applied to PROFIBUS, as significant differences to the timed-token protocol exist. We consider the following two differences as the most relevant ones (Tovar and Vasques, 1998).

1. In PROFIBUS there is no synchronous capacity allocation (H_i). If a station receives a late token (T_{RR} is greater than T_{TR}), then, at most, only one high-priority message may be transmitted. As a consequence, low-priority traffic may drastically affect the capabilities of the PROFIBUS networks to support high-priority (real-time) traffic. Fig. 3.10 illustrates this situation. Contrarily, in the original timed-token protocol the station can transmit synchronous (high-priority) messages during H_i time, even if it has received a late token.
2. In PROFIBUS, both high-priority and low-priority message cycles may overrun the T_{TH} timer. As previously stressed, in PROFIBUS a message cycle can be initiated with a residual T_{TH} value and will be performed until the end. In the timed-token protocol this overrun of T_{TH} is only possible for asynchronous (low-priority) messages, as the transmission of synchronous messages can only be started if the message cycle fits in the time allocated for synchronous transmission.



T_{TR}
Message cycle processed causing an overrun of the T_{TH} timer
Master 1
Master 2
Master 3
token arrival
token passing time
high-priority message cycles with $T_{TH} > 0$
low-priority message cycles with $T_{TH} > 0$
message cycles with $T_{TH} < 0$
high-priority message cycle with a late token

Fig. 3.10 This figure illustrates how low-priority traffic can affect the high-priority traffic capabilities in PROFIBUS networks

In (Vasques, 1996; Vasques and Juanole, 1994) the authors derive pre-run-time schedulability analysis for the PROFIBUS protocol, considering two complementary approaches.

1. If the low-priority traffic is unconstrained, then the real-time traffic requirements may be satisfied, considering that, at least, one pending high-priority message is transmitted per token visit.
2. If the low-priority traffic can be constrained (controlling the number of low-priority message transfers at each master station), then, by an appropriate setting of the T_{TR} parameter, all pending real-time traffic is guaranteed to be transmitted at each token visit.

For the first approach, a deadline constraint is proposed, which, if satisfied, guarantees that the real-time traffic is schedulable:

$$\sum_{M_i \in k} \frac{1}{\left\lfloor P_i / T_{cycle} \right\rfloor} \leq 1, \quad \forall_k \qquad \textbf{(3.5)}$$

where M_i is a high-priority message in master k, P_i is its period, and T_{cycle} is the maximum time interval between two consecutive token arrivals to the master.

It is however easy to show that inequality (3.5) does not consider that PROFIBUS message requests are queued in a FCFS (First-Come-First-Served) queue. Assume, as an example, that a master issues three high-priority messages, with periods of 40ms, 15ms and 10ms, respectively. The highest value of T_{cycle} that satisfies inequality (3.5) is 5ms; that is, $1 / \lfloor 40 / 5 \rfloor + 1 / \lfloor 15 / 5 \rfloor + 1 / \lfloor 10 / 5 \rfloor = 0.958 \leq 1$. If the request with a period (deadline) of 10 ms is the last one in the FCFS queue, and the token as just been passed to the next master, that message request will only be processed after 3 token visits. Consequently, it may suffer (considering that only one request will be processed at each token visit) a queuing delay of up to 15ms, which is a value greater than its period (deadline).

Inequality (3.5) has additionally two major drawbacks.

1. It does not give any estimation of the worst-case response time of each individual message.
2. It does not allow the consideration of sporadic high-priority messages, which typically will have relative deadlines much smaller than their period.

Inequality (3.5) is a function of T_{cycle}. In (Vasques, 1996; Vasques and Juanole, 1994) the authors present the following evaluation for T_{cycle}, which will the basis to set the T_{TR} parameter:

$$T_{cycle} = T_{TR} + t + (n-1) \times C \qquad \textbf{(3.6)}$$

where t represents the network latencies (including the token-passing time), n is the number of masters in the network, and C is the maximum length of a high-priority message. This evaluation of T_{cycle} suffers from a number of inaccuracies, which will later be addressed in Chapter 5.

For the second approach (constrained low-priority traffic profile), the authors define T_{cycle} as follows:

$$T_{cycle} = \sum C_i + \sum C_j + t \tag{3.7}$$

where $\sum C_i$ gives the load concerning high-priority traffic per token cycle, and $\sum C_j$ the allowable load concerning low-priority traffic, also per token cycle. Therefore, the value for the T_{TR} parameter must respect the following condition:

$$T_{TR} \geq T_{cycle} + \max_k \left\{ \sum_i C_i \right\} \tag{3.8}$$

with, of course, the following restriction:

$$\min\{P_i\} \geq T_{cycle}, \forall_{i,k} \tag{3.9}$$

Condition (3.8) guarantees that even when the token rotation time is at its maximum value, master *k* will still be able to process all its pending high-priority traffic. There are, however, some limitations inherent to such analysis.

1. The authors do not identify the low-priority traffic supported by PROFIBUS. It is important to note that some of the low-priority traffic cannot be controlled at the user level.
2. The authors do not discuss how is it possible to implement such an approach.

In Section 5.5, we will address in detail these open issues.

Finally, it is worth mentioning a previous work based on the use of the poll list at the FDL to support real-time communication in PROFIBUS (Li and Stoeckli, 1994). In this approach, message deadlines are guaranteed since the token cycle time is bounded. The major drawback of this approach is that, in order to evaluate the token cycle time, neither high-priority traffic nor low-priority traffic (other than cyclic traffic) are allowed. This prevents the transfer of event-driven messages with high-priority, such as alarms. Furthermore, remote management services (which in PROFIBUS are mapped into low-priority non-cyclic services) are also not covered by this approach.

To our best knowledge, these are the most relevant works on how to guarantee real-time communications with PROFIBUS networks.

3.5. Factory Instrumentation Protocol (WorldFIP)

3.5.1. Main Characteristics of the WorldFIP Protocol

A WorldFIP network interconnects stations with two types of functionalities: bus arbitration and production/consumption functions. At any given instant, only one station can perform the function of active bus arbitration. Hence, in WorldFIP, the medium access control (MAC) is centralised, and performed by the active bus arbitrator (BA).

WorldFIP supports two basic types of transmission services: exchange of identified variables and exchange of messages. In this section we address WorldFIP networks supporting only exchange of identified variables, since they are the basis of the WorldFIP real-time services. The exchange of messages, which is used to support manufacturing message services (ISO 9506, 1990), is out of the scope of this work.

In WorldFIP, the exchange of identified variables is based on a producer/distributor/consumer (PDC) model, which relates producers and consumers within a distributed system. In this model, for each process variable there is one, and only one producer, and several consumers. For instance, consider the variable associated with a process sensor. The station that provides the variable value will act as the variable producer and its value will be provided to all the consumers of the variable (e.g., the station that acts as process controller for that process variable or the station that is responsible for building an historical data base).

In order to manage transactions associated with a single variable, a unique identifier is associated with each variable. The WorldFIP data link layer (DLL) is made up of a set of produced and consumed buffers, which can be locally accessed (through application layer (AL) services) or remotely accessed (through network services).

The AL provides two basic services to access the DLL buffers: *L_PUT.req*, to write a value in a local produced buffer, and *L_GET.req* to obtain a value from the local consumed buffer. None of these services generate activity on the bus.

Produced and consumed buffers can also be remotely accessed through a network transfer (service also known as buffer transfer). The bus arbitrator broadcasts a question frame *ID_DAT*, which includes the identifier of a specific variable. The DLL of the station that has the corresponding produced buffer responds with the value of the variable using a response frame *RP_DAT*. The DLL of the station that contains the produced buffer then notifies the local AL with a (*L_SENT.ind*). The DLL of the station(s) that has the consumed buffers accepts the value contained in the *RP_DAT*, overwriting the previous value and notifying the local AL with a *L_RECEIVED.ind*. These mechanisms are illustrated in Fig. 3.11.

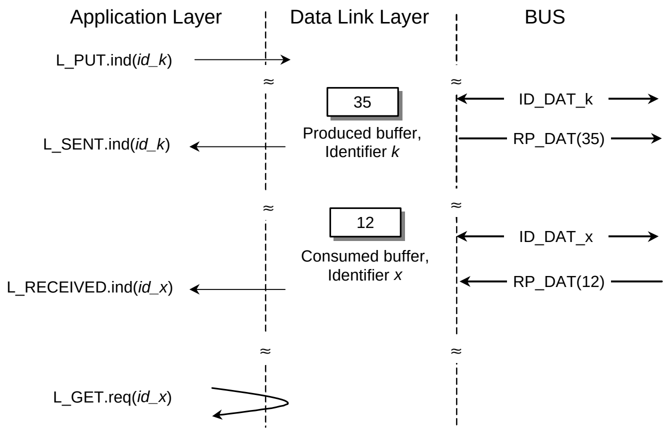


Fig. 3.11 This figure illustrates the case of a station with one produced buffer (for identifier *k*) and one consumed buffer (for identifier *x*).

A buffer transfer implies the transmission of a pair of frames: *ID_DAT*, followed by a *RP_DAT*. We denote this sequence as an elementary transaction. The duration of this transaction equals the time needed to transmit the *ID_DAT* frame, plus the time needed to transmit the *RP_DAT* frame, plus twice the turnaround time (t_r). The turnaround time is the time elapsed between any two consecutive frames. Fig. 3.12 illustrates the concept of an elementary transaction in WorldFIP.

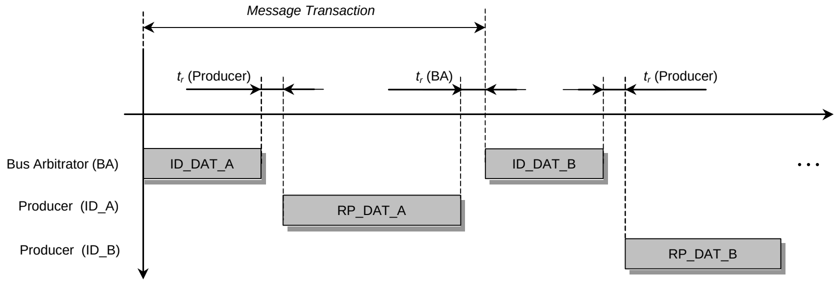


Fig. 3.12 Concept of elementary transaction in WorldFIP networks

Every transmitted frame is encapsulated with control information from the physical layer. Specifically, the frame is placed between a DTR field (begin of frame) and an FTR field (end of frame). A WorldFIP frame begins with a control byte, which is used by network stations to recognise the type of frame, and ends with two FCS (Frame Check Sequence) bytes, used by the frame receivers to verify the integrity of the received frame. The structure of both *ID_DAT* and *RP_DAT* frames is as illustrated in Fig. 3.13. As it can be depicted, an *ID_DAT* frame has always 64 bits, whereas a *RP_DAT* frame has at least 48 bits. Note that the turnaround time (t_r) is imposed (Afnor, 1990) to be within the interval 10 bits < t_r < 70 bits.

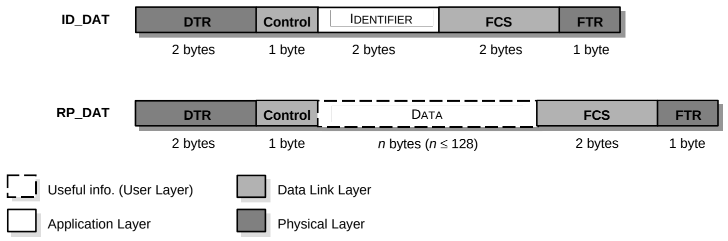


Fig. 3.13 Structure of the *ID_DAT* and *RP_DAT* frames

In WorldFIP networks, the bus arbitrator table (BAT) regulates the scheduling of all buffer transfers. In practice, two types of buffer transfers can be considered: periodic and aperiodic (sporadic). The BAT imposes the schedule of the periodic buffer transfers, and also regulates the aperiodic buffer transfers.

Assume a distributed system within which 6 variables are to be periodically scanned, with scan frequencies as shown in Table 3.3. The WorldFIP BAT must be set in order to cope with these timing requirements.

Table 3.3: Example Set of Periodic Buffer Transfers

Identifier	**A**	**B**	**C**	**D**	**E**	**F**
Periodicity (ms)	1	2	3	4	4	6

Two important parameters are associated with a WorldFIP BAT: the microcycle (elementary cycle) and the macrocycle. The microcycle imposes the maximum rate at which the BA performs a set of scans. Usually, the microcycle is set equal to the highest common factor (HCF) of the required scan periodicities. Using this rule, and for the example shown in Table 3.3, the value for the microcycle is set to 1ms. A possible schedule for all the periodic scans can be as illustrated in figure 3.15, where we consider *C*=97,6µs for each elementary transaction.

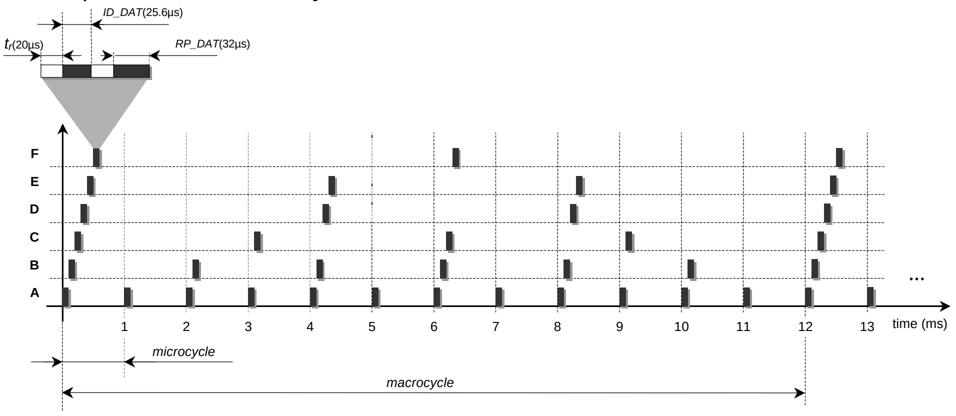


Fig. 3.14 Illustration of a possible schedule for the example of Table 3.3

It is easy to depict that, for this example, the sequence of microcycles repeats each 12 microcycles. This sequence of microcycles is said to be the macrocycle, and its length is given by the lowest common multiple (LCM) of the scan periodicities. As a consequence, the scanning periods of the periodic variables must be multiples of the microcycle.

In a WorldFIP system, not all identified variables are to be included in the BAT. Some variables may only be occasionally exchanged, and thus do not need to be periodically scanned. Typically such exchanges will concern application events or alarms, which by their own nature do not occur with a periodic pattern. Therefore, it is preferable to map these variables into aperiodic buffer transfers, in order to reduce the network load.

The BA handles aperiodic buffer transfers only after processing the periodic traffic in a microcycle. The portion of the microcycle reserved for the periodic buffer exchanges is denoted as the periodic window of the microcycle. The time interval left after the periodic window until the end of the microcycle is denoted as the aperiodic window of the microcycle. The aperiodic buffer transfers take place in three stages (Fig. 3.15a).

1. When processing the BAT schedule, the BA broadcasts an *ID_DAT* frame concerning a periodic variable, say identifier *X*. The producer of variable *X* responds with a *RP_DAT* and sets an aperiodic request bit in the control field of its response frame (*RP_DAT_RQ*). The bus arbitrator stores variable *X* in a queue

of requests for transfers of aperiodic variables. Two priority levels can be set when the request for aperiodic transfer is made: Urgent or Normal. The BA has two queues, one for each priority level (Fig. 3.15b).

2. In the aperiodic window, the BA uses an identification request frame (*ID_RQ*) to request the producer of the identifier *X* to transmit its list of pending aperiodic requests. The producer of *X* responds with a *RP_RQ* frame (list of identifiers). This list of identifiers is placed in another BA's queue, the ongoing aperiodic queue (Fig. 3.15b).
3. Finally, the BA processes the requests for aperiodic transfers that are stored in its ongoing aperiodic queue. For each transfer of aperiodic variables, the BA uses the same mechanism as the used for the periodic buffer transfers (*ID_DAT* followed by *RP_DAT*).

A station that requests an aperiodic transfer can be: the producer of the variable; the consumer of the variable; both producer and consumer; neither producer nor consumer (third-party variables). It is however important to note that a station can only request aperiodic transfers using responses to periodic variables that it produces and which are configured in the BAT.

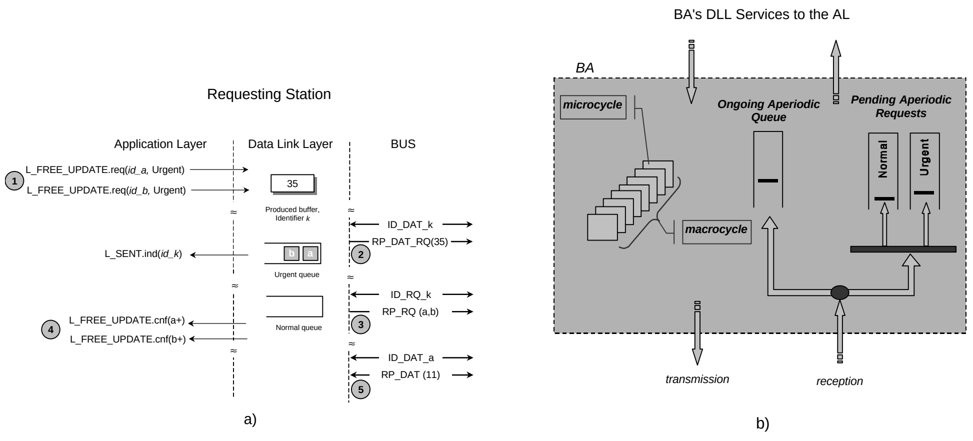


Fig. 3.15 a) illustrates the sequence of transactions concerning an aperiodic buffer transfer request. b) illustrates the structure of the BA's queues

It is worth mentioning that the schedule shown in Fig. 3.14 represents a macrocycle composed of synchronous microcycles, that is, for the specific example, each microcycle starts exactly 1ms after the previous one. Within a microcycle, if there is spare time after processing the aperiodic traffic, the BA transmits padding identifiers, to indicate to the other stations that it is still functioning. A WorldFIP BA can also manage asynchronous microcycles, not transmitting padding identifiers at the end of the microcycle. In such case, a new microcycle starts as soon as the periodic traffic is performed and there are no pending aperiodic buffer transfers or message transfers. Initial periodicities are not respected, since identifiers may be more frequently scanned.

3.5.2. Real-Time Communications with WorldFIP: Review of Relevant Work

For the periodic traffic, end-to-end communication deadlines can be easily guaranteed, since the BAT implements a static schedule for the periodic variables. Therefore, real-time guarantees for periodic traffic very much rely on methodologies for building the WorldFIP BAT. Several authors have already addressed this issue; examples are the works by Laine (Laine, 1991) and by Kim *et al.* (Kim *et al.*, 1998). It is also worth to mention some works, while not directly focusing WorldFIP networks, address generic fieldbus networks with some of the characteristics of WorldFIP networks (Raja *et al.*, 1993; Raja and Noubir, 1993; Almeida *et al.*, 1999).

Concerning the aperiodic traffic, some previous results for pre-run-time schedulability analysis can be found in (Vasques, 1996; Vasques and Juanole, 1994). Those results are however quite pessimistic. Later, in (Pedro and Burns, 1997) the level of pessimism is identified and an improved analysis is proposed. Next, we briefly describe such improved analysis.

The response time analysis of an aperiodic request is given by:

$$R_n = s_k + d + T_s^{\max}\left\lceil \frac{R_n - s_k}{T_{mc}} \right\rceil + \sum_j C_j \qquad \textbf{(3.10)}$$

In equation (3.10), d represents the sum of turnaround times concerning the transactions related to aperiodic requests, T_s^{max} is the maximum length of a periodic window and C_j is the length of aperiodic transactions. Finally, s_k is the length of the dead interval - concept introduced in (Vasques and Juanole, 1994) - in a station *k*. As a station can only request an aperiodic transfer in a response to an `ID_DAT` locally produced, an aperiodic request may only be notified to the BA after a time s_k after the request have been locally queued:

$$s_k = \min\{T_i\} \qquad \textbf{(3.11)}$$

In equation (3.11) T_i corresponds to the periods of periodic variables that station *k* produces.

The analysis by Pedro and Burns still contains, however, the following drawbacks.

1. The authors consider equal lengths for periodic windows since they do not discuss the BAT construction, and thus they cannot evaluate the actual length of the periodic window in each microcycle. Thus, the overall results are also very pessimistic.
2. The authors do not consider that at the end of each microcycle some transactions would simply not fit, and thus would not be schedulable.
3. Finally, the authors do not also consider the communication jitter for the evaluation of the dead interval. Note that within each microcycle the variables are not scanned exactly in the same "slot", and they may be even scanned at irregular intervals (not with the same number of microcycles in between), depending on the methodology used to build the BAT.

The analysis presented in Chapter 6, improves previous results from (Pedro and Burns, 1997) considering the three above mentioned aspects, in an integrated manner with the methodologies for building the WorldFIP BAT, thus performing an integrated analysis of periodic and aperiodic traffic.

3.6. Summary

Fieldbus networks are widely used as the communication support for distributed computer-controlled systems (DCCS), in applications ranging from process control to discrete manufacturing. There are several advantages in the use of fieldbuses as a replacement for the traditional point-to-point links between sensors/actuators and computer-based control systems. Besides economical reasons (cable savings), fieldbuses allow an increased decentralisation and distribution of the processing power over the field.

Usually, DCCS impose real-time requirements; that is, traffic must be sent and received within a bounded interval, otherwise a timing fault is said to occur. This motivates the use of communication networks within which the MAC protocol is able to schedule messages according to their real-time requirements. Therefore, a potential leap towards the use of fieldbuses in such time-critical applications lies on the accurate characterisation of the temporal behaviour of its MAC protocol.

In this chapter we describe four different fieldbus solutions: CAN, P-NET, PROFIBUS and WorldFIP. While the first was originally intended for use as an in-vehicle network, its very interesting characteristics make it also appealing for DCCS applications. CAN is a priority bus, hence its real-time characteristics can be easily evaluated and improved by the traditional analysis for the scheduling of non pre-emptive and independent tasks in single processor environments. This is also the reason why CAN has been the focus of research teams from the real-time systems area. However, its physical limitations prevent its use in most part of the typical factory-floor applications.

For typical factory-floor applications, the profiles encompassed in the EN 50170 standard (P-NET, PROFIBUS and WorldFIP) are strong contenders, since they all offer deterministic access.

In FIP, the determinism is guaranteed by a bus arbitrator, which, for periodic traffic, controls data transfers according a static scanning table. PROFIBUS adopts a simplified version of the timed-token protocol (Grow, 1982). Despite some differences to the timed-token protocol, it is still possible to guarantee real-time behaviour with PROFIBUS networks. P-NET also offers a deterministic access, since it is based on a virtual token passing (VTP) mechanism. The determinism is not achieved by means of controlling the token rotation time, as it happens in networks based on the timed-token protocol. Instead, the bounded access delay is implicitly guaranteed by the fact that at each token visit only one message request may be performed.

In this chapter, we describe the main characteristics of the fieldbus protocols encompassed in the EN 50170 standard. We also survey the most relevant results concerning their ability to support real-time communications, identifying some of the open research issues.

3.7. References

Agrawal, G., Chen, B., Zhao, W. and Davari, S. (1994). Guaranteeing Synchronous Message Deadline with the Timed Token Medium Access Control Protocol. In *IEEE Transactions on Computers*, Vol. 43, No. 3, pp. 327-339.

Almeida, L., Pasadas, R. and Fonseca, J. (1999). Using a Planning Scheduler to Improve the Flexibility of Real-Time Fieldbus Networks. In *Control Engineering Practice*, 7, pp. 101-108.

ASI. (1996). Actuator and Sensor Interface, Low Voltage Switchgear and Controlgear. CENELEC TC17B8secretariat) 146.

BSI DD 238. (1996). Fieldbus Foundation. Draft for Development.

Day, J. and Zimmermann, H. (1983). The OSI Reference Model. In *Proceedings of the IEEE*, Vol. 71, No. 12, pp. 1334-1340.

DIN 19245. (1990). PROFIBUS, Process FieldBus. German Standard.

DIN 19258. (1995). Interbus-S, Sensor and Actuator Network for Industrial Control Systems. German Draft Standard.

DIN 19245-4. (1996). PROFIBUS-PA, Profibus for Process Automation. German Standard.

DS 21906. (1990). P-Net, multi-master, multi-net fieldbus for sensor, actuator and controller communications. Danish Standard.

Echelon. (1993). LONTalk Protocol. In *LONWorksEngineering Bulletin*, Part Number 005-0017-01 Rev. C.

EN 50170. (1996). General Purpose Field Communication System. EN 50170-1 (P-NET), EN 50170-2 (PROFIBUS), EN 50170-3 (WorldFIP). CENELEC.

EN 50254. (1996). High Efficiency Communications Subsystems for Small Data Packages. CLC TC/65CX, CENELEC EN 50254 Project.

Grow, R. (1982). A Timed Token Protocol for Local Area Networks. In *Proceedings of Electro'82*, Token Access Protocols, Paper 17/3.

Haverty, N. (1986). MIL-STD 1553 - A Standard for Data Communications. In *Communication and Broadcasting*, Vol. 10, No. 1, pp. 29-33.

IEC 1158-2. (1993). IEC Standard 1158-2, Fieldbus Standard for Use in Industrial Control Systems - Part2 Physical Layer Specification and Service Definition.

IEEE 802.4. (1985). IEEE Standard 802.4: Token Passing Bus Access Method and Physical Layer Specification, IEEE.

Intel (1984). The Bitbus Interconnect Serial Control Bus Specifications. In Distributed Control Modules. Also (1991) IEEE 1118, Standard Microcontroller System Serial Control Bus.

ISO 9314-2. (1989). Information Processing Systems - Fibre Distributed Data Interface (FDDI) - Part 2: Token Ring Media Access Control (MAC). ISO.

ISO 11519. (1995). Road Vehicles - Low-Speed Serial Data Communication (Part 1 - Part 4). ISO.

ISO 11898. (1993). Road Vehicle - Interchange of Digital Information - Controller Area Network (CAN) for High-Speed Communication. ISO.

Jenkins, C. (1997). P-NET as a European Fieldbus Standard EN 50170 Vol. 1. In *Institute of Measurement + Control Journal*, Vol. 30, April, pp. 75-79.

Kim, Y., Jeong, S. and Kown, W. (1998). A Pre-Run-Time Scheduling Method for Distributed Real-Time Systems in a FIP Environment. In *Control Engineering Practice*, Vol. 6, pp. 103-109.

Laine, T. (1991). Modélisation d'Application Réparties pour la Configuration Automatique d'un Bus de Terrain. (in french), PhD Thesis, CRIN, Nancy, France.

Lenhart, G. (1993). A Fieldbus Approach to Local Control Networks. In *Advances in Instrumentation and Control*, Vol. 48, No. 1, pp. 357-365.

Li, M. and Stoeckli, L. (1994). The Time Characteristics of Cyclic Service in Profibus. In *Proceedings of the EURISCON'94*, Vol. 3, pp. 1781-1786.

Montuschi, P., Ciminiera. L. and Valenzano, A. (1992). Time Characteristics of IEE802.4 Token Bus Protocol. In *IEE Proceedings*, Vol. 139, No. 1, pp. 81-87.

NF C46. (1989). FIP Bus for Exchange of Information Between Transmitters, Actuators and Programmable Controllers. French Standard.

NF C46-638. (1996). Système de Communication Haute Performance pour Petits Modules de Donnés. French Standard.

Noonen, D., Siegel, S. and Maloney, P. (1994). DeviceNet Application Protocol. In *Proceedings of International CAN Conference*.

Pedro, P. and Burns, A. (1997). Worst Case Response Time Analysis of Hard Real-time Sporadic Traffic in FIP Networks. In *Proceedings of the 9th Euromicro Workshop on Real-Time Systems*, pp. 3-10.

Pimentel, J. (1990). Communication Networks for Manufacturing. Prentice-Hall International Editions, Englewood Cliffs, New Jersey.

Raja, P. and Noubir, G. (1993). Static and Dynamic Polling Mechanisms for Fieldbus Networks. In *ACM Operating Systems Review*, Vol. 27, No. 3, pp. 34-45.

Raja, P., Noubir, G., Ruiz, L., Hernandez, J. and Decotignie, J.-D. (1993). Analysis of Polling Protocols for Fieldbus Networks. In *Computer Communication Review*, Vol. 23, No. 3, pp. 69-90.

Rockwell Automation (1997). DeviceNet Produc Overview. Publication DN-2.5, Rockwell.

Rosemount Inc. (1991). HART Smart Communications Protocol Specification. Rev. 5.1.4, Rosemount.

Tindell K., Hansson, H. and Wellings, A. (1994). Analysing Real-Time Communications: Controller Area Network (CAN). In *Proceedings of the IEEE Real-Time Systems Symposium*, pp. 259-263.

Tindell K., Burns, A. and Wellings, A. (1995). Calculating Controller Area Network (CAN) Message Response Time. In *Control Engineering Practice*, Vol. 3, No. 8, pp. 1163-1169.

Thomesse, J. (1997). The Fieldbuses. In *Proceedings of the International Symposium on Intelligent Components and Instruments for Control Applications*, pp. 13-23.

Tovar, E. and Vasques, F. (1998). Cycle Time Properties of the PROFIBUS Timed Token Protocol. Polytechnic Institute of Porto, Technical Report HURRAY-TR-9811, August 1998, to appear in *Computer Communications*, Elsevier Science.

Vasques, F. and Juanole, G. (1994). Pre-run-time Schedulability Analysis in Fieldbus Networks. In *Proceedings of the Annual Conference of the IEEE Industrial Electronics Society*, pp. 1200-1204.

Vasques, F. (1996). Sur l'Integration de Mecanismes d'Ordonnacement et de Communication dans la Sous-couche MAC de Reseaux Locaux Temps-reel. PhD Thesis (*in French*), available as Technical Report LAAS No. 96229.

Zheng, Q. and Shin, K. (1995). Synchronous Bandwidth Allocation in FDDI Networks. In *IEEE Transactions on Parallel and Distributed Systems*, Vol. 6, No. 12, pp. 1332-1338.

Zuberi, K, and Shin, K. (1995). Non-Preemptive Scheduling of Messages on Controller Area Network for Real-Time Control Applications. In *Proceedings of the IEEE Real-Time Systems Symposium*, pp. 240-249.

Zuberi, K, and Shin, K. (1997). Scheduling Messages on Controller Are Network for Real-Time CIM Applications. In *IEEE Transactions on Robotics and Automation*, Vol. 13, No. 2, pp. 310-314.

Chapter 4

Real-Time Communications with P-NET Networks: Contributions to the State-of-the-Art

In this chapter we develop a methodology for the worst-case response time analysis of P-NET messages. This chapter is largely drawn from the following published work: "A Communication Support for Real-Time Distributed Computer Controlled Systems" (Tovar and Vasques, 1998a); "Communication Response Time in P-NET Networks: Worst-Case Analysis Considering the Actual Token Utilisation" (Tovar *et al.*, 1999); "Supporting Real-Time Distributed Computer-Controlled Systems with Multi-hop P-NET Networks" (Tovar *et al.*, 1998b).

4.1. Introduction

The P-NET medium-access-control (MAC) protocol is based on a virtual token-passing (VTP) procedure, used by master stations to grant bus access to each other, and a master-slave procedure, used by master stations to communicate with slave stations. Please refer to Section 3.3.1 for a detailed description of the main characteristics of this protocol.

The remainder of this chapter is organised as follows. In Section 4.2 we introduce the network and message models, which will be used throughout the rest of the chapter. In Section 4.3 we provide a basic response time analysis considering the worst-case token rotation time. In Section 4.4 we extend the basic analysis to consider the actual token utilisation. Finally, in Section 4.5 we assess the real-time characteristics of multi-hop P-NET networks.

4.2. Network and Message Models

Consider a network with *n* masters, with addresses ranging from 1 to *n*. Each master accesses the network according to the VTP scheme; hence, first master 1, then masters 2, 3, … until master *n*, and then again masters 1, 2, … . Slaves will have network addresses higher than *n*.

We assume the following message stream model:

$$S_i^k = (C_i^k, T_i^k, D_i^k) \tag{4.1}$$

S_i^k defines a message stream *i* in master *k* (*k* = 1, ..., *n*). A message stream is a temporal sequence of message cycles concerning, for instance, the remote reading of a specific process variable. C_i^k is the longest message cycle duration of stream S_i^k. This duration includes both the longest request and response transmission times, and also the worst-case slave's turnaround time. T_i^k is the periodicity of stream S_i^k requests. In order to have a subsequent timing analysis independent from the model of the tasks at the application process level, we assume that this periodicity is the minimum interval between two consecutive arrivals of S_i^k requests to the outgoing queue. Finally, D_i^k is the relative deadline of a message cycle; that is, the maximum admissible time interval between the instant when the message request is placed in the outgoing queue and the instant at which the related response is completely received at the master's incoming queue. Finally, ns^k denotes the number of message streams associated with a master k.

4.3. Basic Analysis for the Worst-Case Response Time

In our model, the relative deadline of a message can be equal or smaller than its period ($D_i^k \leq T_i^k$). Thus, if in the outgoing queue there are two message requests from the same message stream, this means that a deadline for the first of the requests was missed. Actually, we can be more precise saying that deadlines will be missed if a new request appears, in the outgoing queue, before the completion of a previous message cycle for the same request. Therefore, the maximum number of pending requests in the outgoing queue will be, in the worst-case, ns^k.

We denote the worst-case response time of a message stream S_i^k as R_i^k. This time is measured starting at the instant when the request is placed in the outgoing queue, until the instant when the response is completely received at the incoming queue. Basically, this time span is made up of the two following components.

1. The time spent by the request in the outgoing queue, until gaining access to the bus (queuing delay);
2. The time needed to process the message cycle, that is, to send the request and receive the related response (transmission delay). As the bit rate in P-NET is 76800bps, the propagation delay can be neglected, even for P-NET networks with a length of some kilometres.

Thus,

$$R_i^k = Q_i^k + C_i^k \quad \textbf{(4.2)}$$

where Q_i^k is the worst-case queuing delay of a message request from S_i^k.

A basic analysis for the worst-case response time can be performed if the worst-case token rotation time is assumed for all token cycles. Assume also that C_M is the maximum transmission duration of a message cycle. If a master uses the token to perform a message cycle, we can define the token holding time as:

$$H = r + C_M + t \quad \textbf{(4.3)}$$

It is not usual to include the token passing time in the token holding time. However, due to the specificity of the Virtual Token Passing scheme, we decided to associate the token holding time with the state of the P-NET access counter.

In equation (4.3), t (= 40 bit periods) corresponds to the time to pass the token after a message cycle has been performed. r ($\leq$ 7 bit periods) denotes the worst-case master's reaction time. If a station does not use the token to perform a message cycle, the bus will be idle during s (= 10 bit periods) before all access counters are incremented (please refer to Section 3.3.1).

As the token rotation time is the time interval between two consecutive visits of the token to a particular station, the worst-case token rotation time, denoted as V, is:

$$V = n \times H \tag{4.4}$$

with H as defined in equation (4.3). The value V represents the worst-case time interval between two consecutive token arrivals to any master k (k = 1, ..., n).

In P-NET, the outgoing queue is implemented as a FCFS queue. Therefore, a message request can be in any position within the ns^k pending requests. ns^k is also the maximum number of requests which, at any time, can be pending in the master k outgoing queue. This results from the adopted message stream model, which considers $D_i^k \leq T_i^k$. Hence, the maximum number of token visits needed to process a message request in master k, is ns^k.

The worst-case queuing delay occurs if ns^k requests are placed in the outgoing queue just after a message cycle was completed (at the beginning of the token passing interval: t) and the token is fully utilised in the next ns^k consecutive token cycles. We denote this time instant as t_c. We consider that a message cycle was just completed since the token passing time is t (= 40 *bp*) instead of s (= 10 *bp*). Considering the value t leads to the largest time span till the next visit of the token to that same master k. Only then master k will be able to process the first of the ns^k requests that were placed in the outgoing queue at time instant t_c (please refer to Fig. 4.1 for a specific example).

Definition 4.1 – Master's Critical Instant – We define the critical instant in master k, as the time instant when ns^k requests are placed in its outgoing queue just after it has completed a previous message cycle.

Note that we can not consider releasing ns^k new requests while master k is processing a message cycle, since that would mean a deadline violation in master k (a new request released before the completion of a previous one from the same stream). If there was no message cycle being processed, there was no point in considering an earlier release time, since one of those ns^k requests would be processed in the preceding visit.

Due to both the deadline restriction and the FCFS behaviour of the outgoing queue, no additional request may appear in master k till the time instant (t_e), when the last of the ns^k requests (made at t_c) is completely processed, otherwise, a message deadline would be missed. Therefore, we introduce Definition 4.2 and Theorem 4.1.

Definition 4.2 – Master's Busy Period – We define the busy period in master k, as the time interval between the critical instant, t_c, and the time instant t_e, when the last of the ns^k requests is completely processed.

Theorem 4.1 In P-NET networks, the worst-case response time of a master's message request corresponds to the longest busy period in that master.

Proof: The busy period starts when a critical instant occurs. From the critical instant definition, ns^k requests are placed in the outgoing queue at the earliest possible instant. As the end of the busy period is defined as being the time instant t_e when the last of those ns^k requests is completely processed, the difference $t_c - t_e$ gives the worst-case response time for a message request in master k, since at time instant t_c, a message request can be in any position, from 1st to ns^k-nd, due to the FCFS behaviour of the outgoing queue. ❑

Theorem 4.2 In P-NET networks, assuming that the token is fully utilised, the worst-case response time of a message request in a master k is:

$$R^k = ns^k \times V \tag{4.5}$$

Proof: Assuming that the token is fully utilised, it will take $t + (n - 1) \times H$ from instant t_c until the next visit to master k. At the first visit, the token arrives at $t_2 = t_c + t + n + (n - 1) \times H$, and only then the master will be able to process the first of the ns^k pending requests. As only one of the ns^k message requests is processed per token visit, the token will arrive at master k only at instant $t_3 = t_2 + (ns^k - 1) \times V$ to process the last of the ns^k requests. The time elapsed since t_c is then $t_3 - t_c = t + (n - 1) \times H +$ $+ (ns^k - 1) \times V$. As the worst-case reaction time of a master is r, the last one of the ns^k message requests will start to be transmitted with a queuing delay $Q^k = t +$ $+ (n - 1) \times H + (ns^k - 1) \times V + r$. Note that as we are assuming $C_i^k = C_M$, $\forall_{i,k}$, the worst-case queuing delay is equal for all message requests in the same master ($Q_i^k = Q^k$, $\forall_i$). As $R_i^k = Q_i^k + C_M$, the worst-case response time for a message stream i in master k is (note that R_i^k is also equal to R^k): $R^k = t + (n - 1) \times H + (ns^k - 1) \times V + r + C_M$, which, considering that $H{=}r{+}C_M{+}t$, can be re-written as follows:

$$R^k = n \times H + (ns^k - 1) \times V = V + (ns^k - 1) \times V = ns^k \times V$$

❑

Corollary 4.1 In P-NET networks, assuming that the token is fully utilised, the worst-case queuing delay of a message request in a master k is:

$$Q^k = t + (n-1) \times H + (ns^k - 1) \times V + r = ns^k \times V - C_M \tag{4.6}$$

To illustrate Theorem 4.1 and Theorem 4.2, assume a network scenario with $n = 3$ and $ns^1 = 2$. Fig. 4.1 shows both Q^1 and R^1 values for such scenario. Note that at t_c, the ns^1 requests are placed in the outgoing queue in any arbitrarily order. Whichever the ordering, the busy period corresponds to R^1, and therefore, the worst-case response time for a message request in master 1 is (6): $ns^1 \times V = 2 \times V = 2 \times 3 \times H = 6 \times H$.

Having found the value for the worst-case response time of a message request in each master k, a pre-run-time schedulability test results:

$$D_i^k \geq R^k, \forall_{i,k} \tag{4.7}$$

That is, the worst-case response time ($R_i^k = R^k$) of a message stream S_i^k must be equal or smaller than its relative deadline (D_i^k).

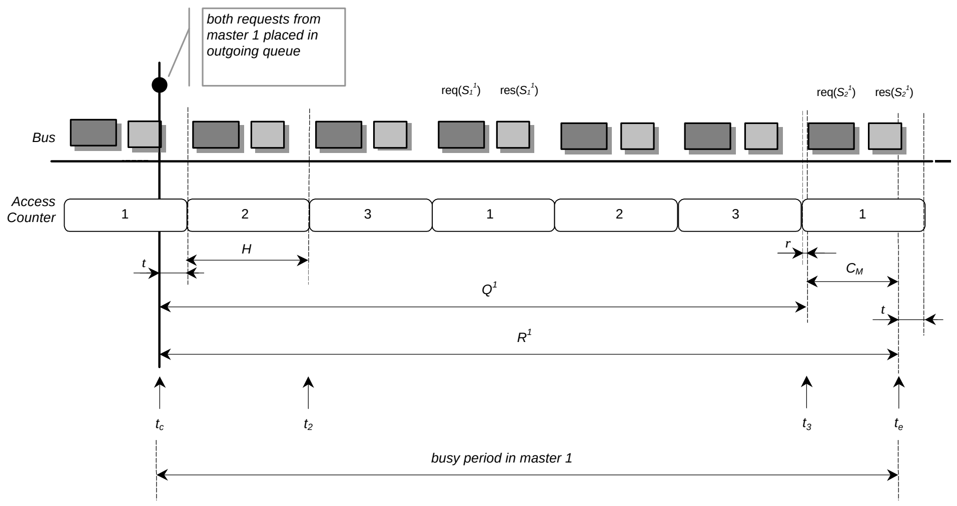


Fig. 4.1 Queuing and response times of a P-NET message

4.4. Schedulability Analysis Considering the Actual Token Utilisation

In the previous section we derived a basic timing analysis for the evaluation of the worst-case message response time. Such analysis may however be very pessimistic, since we assumed the token as being fully utilised in the ns^k consecutive token cycles of the busy period. However, the token can only be fully utilised during that interval if:

$$ns^y \geq ns^k, \forall_{y \neq k} \tag{4.8}$$

as, only in such case, the number of pending requests, in each master y, may be greater than ns^k. Otherwise, if $\exists_{y \neq k}$: $ns^y < ns^k$, the token utilisation depends on the periodicity of message streams for those masters y.

Definition 4.3 – Master's Eligible Requests – We define the eligible requests of master $y \neq k$, as the maximum number of requests generated in that master that will be pending[1] within the busy period of master k.

If the number of eligible requests of master y (Er^y) is smaller than ns^k, then master y will not use all ns^k token visits to process message cycles. Therefore, the evaluation of the Er^y is of paramount importance for the worst-case response time analysis considering the actual token utilisation. We will use the following equation as the starting basis for the evaluation of Er^y:

[1] Even they are processed during the busy period of master k, for a while, they were pending.

$$Er^y(t) = ns^y + \sum_{i=1}^{ns^y} \left\lfloor \frac{t}{T_i^y} \right\rfloor \quad \textbf{(4.9)}$$

Equation (4.9) gives the maximum number of requests generated by a master y within a time interval t: ns^y requests are made at the beginning of the interval, and then, new requests are made at their maximum rate. This is also known as the *asap* (as soon as possible) pattern (Liu and Layland, 1973). By tailoring equation (4.9) to encompass the P-NET MAC characteristics, we will be able to perform a worst-case message response time analysis which considers the actual token utilisation.

4.4.1. Concept of P-NET Logical Ring Request Jitter

From equation (4.9), it is obvious that the larger the considered time interval is, the higher is the value for Er^y. Note, however, that this equation is a step function, hence it varies only for multiples of T_i^y.

Consider that in each master $y \neq k$, ns^y requests are simultaneously made at the critical instant (t_c) of master k. Remembering that since the busy period is defined as $[t_c, t_e]$, it would be reasonable to consider as the eligible requests of master y, all those requests given by $E_r^y = ns^y + \Sigma_{i=1,..,ns^y} \lfloor (t_e - t_c) / T_i^y \rfloor$. In the following analysis, we will show that the worst-case situation appears when the ns^y requests are not simultaneously made in all masters $y \neq k$, and that a quantity before t_c must be considered for each master y, depending on its position in the logical ring.

Assume that the critical instant, which must be considered for each master y, is denoted as t_r^y, with $t_r^y < t_c$, $\forall_{y \neq k}$. Basically we need to analyse how much earlier t_r^y can be made, increasing the number of master y eligible requests $E_r^y = ns^y + \Sigma_{i=1,...,ns^y} \lfloor (t_e - t_c)/T_i^y \rfloor$, without any of the initial ns^y requests being able to be processed in an earlier token visit, prior to the critical instant in master k.

For master k^{-1}, which denotes the predecessor of master k, the instant t_r can be shifted back by $C_M + r + t$, being coincident with the starting of a busy period in master k^{-1}. The critical instant t_r cannot be shifted further back, since it would imply a deadline violation in master k^{-1} or one of the initial ns^y requests would be processed prior to the busy period in master k. Considering t as the token passing time implies that a message cycle was just completed at instant t_r. Otherwise, the token passing time would be reduce to s (please refer to section 3.3.1). Consequently, the total amount that t_r may be shifted back for the case of master k^{-1}, is $C_M + r + t = H$.

Considering master k^{-2}, and following a similar analysis, t_r could be shifted back up to $2 \times H$. Thus, a different value for t_r, denoted as t_r^y, must be considered for each master $y \neq k$, and its value only depends on the relative logical ring position of master y with respect to master k.

Definition 4.4 – Logical Ring Request Jitter – We define the logical ring request jitter of master y, as the difference $Jr^y = t_r^y - t_c$, being t_r^y how much earlier than the critical instant in k, a master y can made its ns^y requests, without violating a deadline, nor processing any of those ns^y requests prior to the critical instant in master k.

Resulting from the previous analysis, Jr^y can be expressed as follows:

$$Jr^y = \sum_{i=y,y^{+1},\dots,k^{-1}} H \tag{4.10}$$

which is equivalent to:

$$Jr^y = [(n+k-y) \bmod n] \times H \tag{4.11}$$

To illustrate this definition, assume the following network scenario (Table 4.1). For simplification, the periodicity of streams is expressed in multiples of *H*.

Table 4.1: Stream Set Scenario 1

Master		**(*C, T, D*)**		
1	$ns^1 = 3$	(C_M, 14, 14)	(C_M, 20, 20)	(C_M, 20, 20)
2	$ns^2 = 1$	(C_M, 8, 8)		
3	$ns^3 = 3$	(C_M, 14, 14)	(C_M, 20, 20)	(C_M, 20, 20)
4	$ns^4 = 3$	(C_M, 14, 14)	(C_M, 20, 20)	(C_M, 20, 20)

The time-line for the master 1 busy period will result as illustrated in Fig 4.2, where $(t_r^2 - t_c)$, $(t_r^3 - t_c)$ and $(t_r^4 - t_c)$ represent the logical ring request jitter, respectively of masters 2, 3 and 4. Note that for master 2, its number of eligible requests is greater than ns^2.

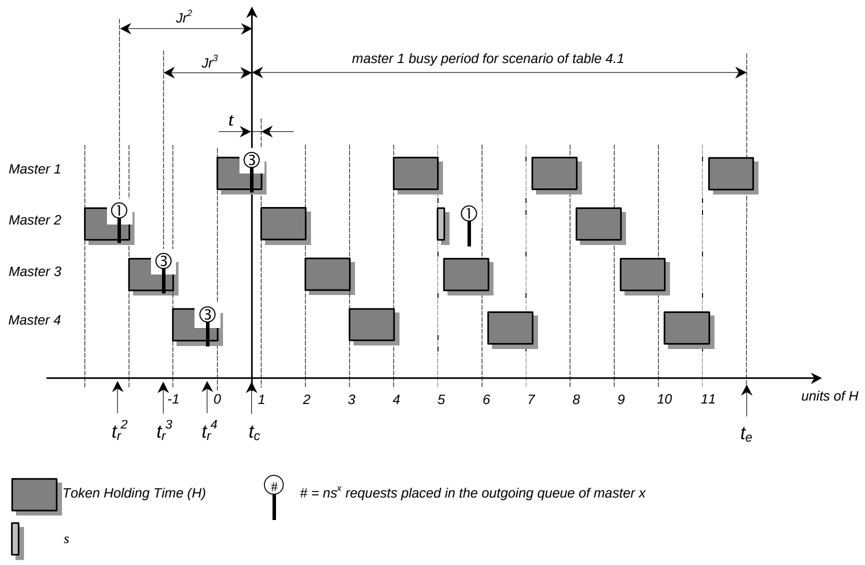


Fig. 4.2 Busy Period of Master 1 with the Scenario of Table 4.1

Theorem 4.3 In P-NET networks, the longest busy period of master *k* occurs when all its predecessors started their busy periods in the token cycle previous to the busy period in master *k*.

Proof: The worst-case length busy period in master *k*, results if all the eligible requests of each master *y* are considered for transmission during the interval $t_e - t_c$. Considering that in each master *y*, ns^y requests are placed in each master's outgoing queue at the instant $t_c - Jr^y$, then for each master *y* the number of eligible requests is $E_r^y = ns^y + {}$ $+ \Sigma_{i=1,\ldots,ns^y} \lfloor (t_e - t_c + t_r^y) \; / \; T_i^y \rfloor$.

Using the definitions of busy period and logical ring request jitter, if ns^y requests are placed in the outgoing queues at $t_1 - Jr^y$, then, in the token cycle prior to the busy period in master *k*, busy periods have started in all predecessors of *k*. ❑

Considering P-NET's logical ring request jitter concept, the number of eligible requests (4.9) can now be updated to:

$$Er^y = ns^y + \sum_{i=1}^{ns^y} \left\lfloor \frac{t_e - t_c + t_r^y}{T_i^y} \right\rfloor \quad \textbf{(4.12)}$$

which, by using Theorem 4.1, Theorem 4.3 and Definition 4.4, can be re-written as:

$$Er^y = ns^y + \sum_{i=1}^{ns^y} \left\lfloor \frac{R^k + Jr^y}{T_i^y} \right\rfloor \quad \textbf{(4.13)}$$

with Jr^y as defined in (4.11).

4.4.2. Concept of P-NET Logical Ring Visit Jitter

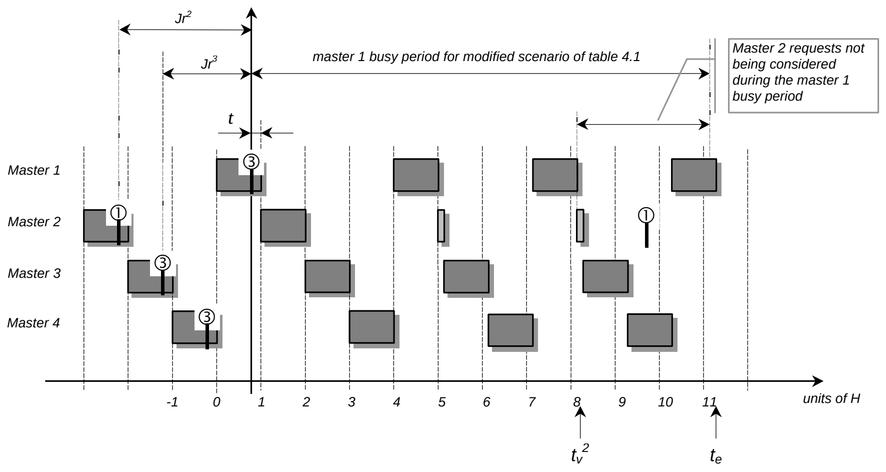
Equation (4.13) is still pessimistic, since not all the eligible requests will be able to be processed within the busy period of master *k*. The reason is obvious. If ns^y is greater than ns^k, only a maximum of ns^k requests will be processed by master *y* within the busy period. However, for our analysis, the relevant case is when $ns^y < ns^k$, since it leads to a scenario where the token is not fully utilised. For this case, even if Er^y (as given by equation (4.13)) is larger than ns^k, it might happen that a number smaller than ns^k requests could be processed during the busy period.

Intuitively we can show that if a new request appears in the outgoing queue of master *y* at time instant t_2 ($t_c < t_2 < t_e$), this request may not be processed before t_e, even if the outgoing queue was empty. This is the case of all the requests made in master *y* after the last token visit (to master *y*) prior to the completion of the busy period in master *k*.

Assume the following example, where we modify, in Table 4.1, the periodicity of stream S_1^2 from 8 to 12. The time-line for the busy period in master *k* would be as shown in Fig. 4.3, instead of that shown in Fig. 4.2. Note that a new request for master 2 appearing before t_e, can not be processed during the busy period of master 1.

Definition 4.5 – Processing Window of Master's Busy Period – We define the processing window of master *k* busy period, as the time span between t_c and t_v^y ($t_v^y < t_e$), within which, a first-positioned pending request in master *y* will assuredly be processed.

Definition 4.6 – Logical Ring Visit Jitter – We define the logical ring visit jitter (Jv^y) of master *y*, as the difference $t_e - t_v^y$.



Jr^2
Jr^3
master 1 busy period for modified scenario of table 4.1
Master 2 requests not being considered during the master 1 busy period
t
Master 1
Master 2
Master 3
Master 4
-1
0
1
2
3
4
5
6
7
8
9
10
11
units of H
t_v^2
t_e

Fig. 4.3 This figure illustrates the concept of P-NET logical ring visit jitter

It becomes obvious that the worst-case response time of a message request in a master *k* corresponds to processing windows in masters $y \neq k$ as large as possible, since this is the case where more eligible requests would be processed during the busy period of master *k*. Thus, the worst-case response time of a message request in a master *k* corresponds to the minimum logical ring visit jitter in masters *y*.

Such minimum logical ring visit jitter Jv^y can be evaluated considering that none of the masters *y* processed any message request in the last token visit prior to the completion of the busy period in master *k*.

Therefore,

$$Jv^y = [((n+k-y) \bmod n) - 1] \times s + r + C_M + (s - r) \qquad \textbf{(4.14)}$$

where $r + C_M$ corresponds to the processing time of the last of the ns^k requests in master *k* (see Fig. 4.1), $s + r$ corresponds to master *y*, and $[((n + k - y) \bmod n) - 1]$ corresponds to the number of masters between *y* and *k*.

There is a certain level of pessimism in considering that none of the masters *y* processed any message request in the last token visit prior to the completion of the busy period in master *k*. In fact, if for some of those masters $ns^y \geq ns^k$, then, they will assuredly use the token in all ns^k consecutive cycles of the busy period in master *k*. For those masters, we may consider *H* instead of s, hence diminishing the length of the busy period processing window.

Therefore, equation (4.14) can be updated to:

$$Jv^y = [((n+k-y) \bmod n)] \times s + C_M + \sum_{\substack{i=y^{+1},\dots,k^{-1} \\ \text{with} \\ ns^y \geq ns^k}} (H - s) \qquad \textbf{(4.15)}$$

For the previous example (Fig. 4.3), the number of eligible requests of master 2 would be 2. However, as, for that master, the logical ring visit jitter is $Jv^2 = [((4 + 1 - 2) \bmod 4) - 1] \times s + C_M + \Sigma_{i=3,4 \text{ with } ns^i \geq ns^1}(H - s) = 3 \times s + C_M +$ $+ 2 \times (H - s) = 2 \times H + C_M + s$, only one of those two eligible requests is able to be processed within the busy period of master 1.

4.4.3. Number of Unused Tokens during the Longest Busy Period

Using the previous analysis, we are now able to evaluate the maximum number of eligible requests from each master y that may be processed during the busy period of master k. Such maximum number will lead to the worst-case response time of a message request in master k.

Definition 4.7 – Master's Logical Ring Aggregate Jitter – We denote $Ja^y = Jr^y - Jv^y$ as the logical ring aggregate jitter of master y.

Definition 4.8 – Minimum Number of Unused Tokens During a Busy Period – We define the minimum number of unused tokens by a master y (Ut^y) during the busy period in master k, as the minimum number of times that a master y receives the token and does not have any pending requests, during that period.

Theorem 4.4 The minimum number of unused tokens by a master y within a busy period of master k, is $Ut^y = ns^y - \min\{ns^k, ns^y + \Sigma_{i=1,\dots,ns^y} \lfloor (R^k + Ja^y) / T_i^y \rfloor\}$.

Proof: By Theorem 4.3, the maximum number of eligible requests of master y is $ns^y + \Sigma_{i=1,\dots,ns^y} \lfloor (R^k + Jr^y) / T_i^y \rfloor$. From these requests, only those which arrive within the master k processing window, will be able to be processed within the busy period. Therefore, the evaluation interval for the *asap* pattern is $R^k + Jr^y - Jv^y = R^k + Ja^y$.

In a master k, the number of token cycles during the busy period is, by Definition 4.1, equal to ns^k. Thus, the actual token utilisation by a master y, during the busy period of master k, is $\min\{ns^k, ns^y + \Sigma_{i=1,\dots,ns^y} \lfloor (R^k + Ja^y) / T_i^y \rfloor\}$.

As a consequence, the number of times master y does not use the token during the busy period of master k is:

$$Ut^y = ns^k - \min\left\{ ns^k, ns^y + \sum_{i=1}^{ns^y} \left\lfloor \frac{\left(R^k + Ja^y\right)}{T_i^y} \right\rfloor \right\} \qquad \textbf{(4.16)}$$

❑

Theorem 4.5 The minimum number of unused tokens during a busy period of master k, is

$$Ut = \sum_{y=1}^{n} Ut^y \qquad \textbf{(4.17)}$$

Proof: Since $y = k$, $Ut^y = 0$, the proof for this theorem is obvious.

❑

4.4.4. Analysis of the Worst-Case Response Time

Considering that the token is fully utilised (Section 4.3), the worst-case response time of a message request in a master *k* (equation (4.5)) is $R^k = ns^k \times V$.

It is now possible to update equation (4.5) to incorporate the actual token utilisation, considering that, for each unused token we must subtract the corresponding value of the token holding time (*H*), and add a s corresponding to the token passing time for the case of an unused token:

$$R^k = ns^k \times V - Ut \times (H - s) \qquad \textbf{(4.18)}$$

Using the results obtained along this sub-section, the worst-case response time of a message request in a master *k* is:

$$R^k = ns^k \times V - \left[\sum_{y=1}^{n} \left(ns^k - \min\left\{ ns^k, ns^y + \sum_{i=1}^{ns^y} \left\lceil \frac{R^k + Ja^y}{T_i^y} \right\rceil \right\} \right) \right] \times (H - s) \qquad \textbf{(4.19)}$$

As expected, this equation embodies a mutual dependence, since R^k appears in both sides of the equation. In fact, all the previous analysis underlay this mutual dependence, since in order to evaluate R^k, *Ut* must be found, and *vice-versa*.

We can use the same approach as described in sub-section 2.4.3, and form the following recurrence relationship:

$$W^{m+1} = ns^k \times V - \left[\sum_{y=1}^{n} \left(ns^k - \min\left\{ ns^k, ns^y + \sum_{i=1}^{ns^y} \left\lceil \frac{W^m + Ja^y}{T_i^y} \right\rceil \right\} \right) \right] \times (H - s) \qquad \textbf{(4.20)}$$

The set of values $\{W^0, W^1, W^2, \ldots, W^m, \ldots\}$ is monotonically non decreasing, since as W evolves, less unused tokens are being considered. Starting with $W^0 = 0$, when $W^m = W^{m+1}$, the solution for equation (4.19) has been found.

4.4.5. Pre-Run-Time Schedulability Condition

For obvious reasons, inequality (4.7) holds for this analysis considering the actual token utilisation, and thus, a pre-run-time schedulability condition is:

$$D_i^k \geq ns^k \times V - Ut \times (H - s), \ \forall_{i,k} \qquad \textbf{(4.21)}$$

Note that inequality (4.21) constrains D_i^k to be larger than a value that is proportional the number of message streams in master *k*. This aspect may be very restrictive for masters dealing with a large number of remote I/O points. In Chapter 7, we will present some solutions to overcome this problem, which is due to the FCFS characteristics of the P-NET outgoing queue.

4.4.6. Pre-Run-Time Schedulability Tool

As for the case of the response time analysis in a single processor environment, the communication response time analysis imposes that each message stream must be individually tested. However, as the proposed schedulability condition is to be done prior to run time, no major cost exists, provided that a software analysis tool is available. In Appendix A.1, we detail an algorithm which may be the basis for the implementation of such a software tool.

4.4.7. Numerical Example

Assume the following message stream set:

Table 4.2: Another Message Stream Set Example

Master		**(*C, T, D*)**		
1	***ns***^{1} **= 3**	(C_M, 14, 14)	(C_M, 20, 20)	(C_M, 40, 40)
2	***ns***^{2} **= 1**	(C_M, 12, 12)		
3	***ns***^{3} **= 3**	(C_M, 14, 14)	(C_M, 20, 20)	(C_M, 20, 20)
4	***ns***^{4} **= 2**	(C_M, 14, 14)	(C_M, 20, 20)	

Applying equation (4.19) by using the recurrence relationship given by equation (4.20), we will be able to find the worst-case response time for master 1.

As the number of streams in master 3 is equal to the number of streams in master 1, we need only to focus on the unused tokens of masters 2 and 4.

Therefore, the network aggregate release jitter for master 2 will be:

$$Ja^2 = 3\times H - Jv^2 = 3\times H - (3\times s + C_M + 1\times(H - s)) = 2\times H - C_M - 2\times s$$

and the network aggregate release jitter for master 4 will be:

$$Ja^4 = 1\times H - Jv^2 = H - (s + C_M + 0) = H - C_M = r + t$$

For $W^0 = 0$, then, W^1 is,

$$= 3\times 4\times H - nut\times(H - s) = 3\times 4\times H - (2+1)\times(H - s) = 9\times H + 3\times s$$

For $W^1 = 9\times H + s$, W^2 is:

$$= 3\times 4\times H - (2+1)\times(H - s) = 9\times H + 3\times s$$

The iterations stop at this point, as $W^2 = W^1$. This corresponds to the time-line illustrated in Fig 4.4.

If we consider that the longest message cycle is composed by 67 P-NET frame bytes (request + response), then this corresponds to 67 × 11= 670 bits. Including 30 *bp*, for the worst-case reaction time of a slave, then C_M = 767 / 76800 =10 ms.

Therefore, H = (7 + 767+ 40) / 76800 = 10.6ms.

This means that the worst-case response time for a message request in master 1 is: 9 ×10.6 + 3 × (10 / 76800) = 95.79ms.

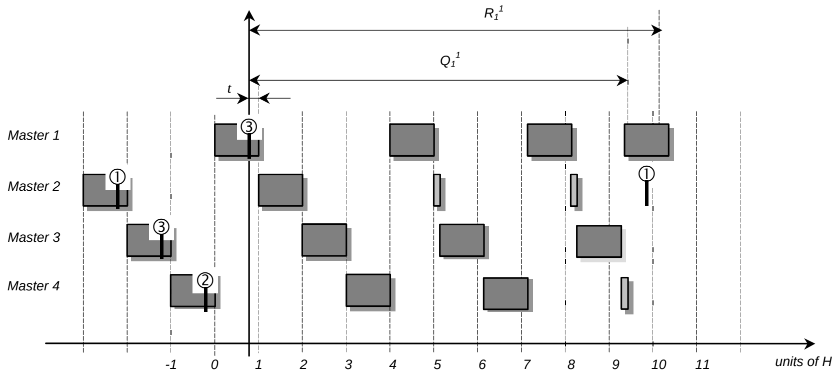


Fig. 4.4 Busy Period of Master 1 within the Scenario of Table 4.2

4.4.8. Considering the Actual Transmission Time for Message Cycles

In the previous analysis, the message cycles' length was considered, for simplification, to be constant ($C_i^k = C_M$, $\forall_{i,k}$). The results can now be updated, considering the actual message cycles' length, or at least, the longest (smallest) message cycle in each master.

Considering $M^k = \max_{i=1,\dots,ns^k}\{C_i^k\}$ as the longest message cycle in a master k and that the token is fully utilised then, the worst-case queuing delay of a message request in master k (updating (4.6)) is (note that now $Q_i^k = Q^k$, $\forall_i$ is not valid any more):

$$Q_i^k = ns^k \times \sum_{l=1}^{n}\left(r + M^l + t\right) - C_i^k \qquad \textbf{(4.22)}$$

The logical ring request jitter (4.10), can be updated to:

$$Jr^y = \sum_{i=y,y^{+1},\dots,k^{-1}}\left(r + M^i + t\right) \qquad \textbf{(4.23)}$$

The logical ring visit jitter (4.15), can be updated to:

$$Jv^y = \sum_{l=y,\dots,k^{-1}} s + \min_{i=1,\dots,ns^k}\left\{C_i^k\right\} + \sum_{\substack{l=y^{+},\dots,k^{-1}\\ ns^l \geq ns^k}}\left(L^l - s\right) \qquad \textbf{(4.24)}$$

where L^l is defined as the smallest message cycle in a master l: $L^l = r + \min_{i=1,\dots,ns^l}\{C_i^l\} + t$.

The worst-case response time of a message request in a master k (4.19) can be updated to (we need now to consider the shortest holding time in each master):

$$R_i^k = ns^k \times \sum_{l=1}^{n}\left(r + M^l + t\right) - C_i^k - \sum_{y=1}^{n}\left[\left(ns^k - \min\left\{ns^k, ns^y + \sum_{j=1}^{ns^y}\left\lceil \frac{R_i^k + Ja^y}{T_j^y} \right\rceil\right\}\right) \times \left(L^y - s\right)\right] \qquad \textbf{(4.25)}$$

Finally, as R_i^k may be different from stream to stream in each master, the pre-run-time schedulability condition (4.7) must be updated to:

$$D_i^k \geq R_i^k, \forall_{i,k} \tag{4.26}$$

4.5. Extending the Analysis to Multi-Hop P-NET Networks

P-NET hopping devices (labelled as gateways in the standard, but termed hopping devices in this work) allow the interconnection of different network segments, each one with independent logical virtual token-passing schemes. In P-NET, the function of a gateway is to isolate two or more bus segments, and to automatically route a frame between the connected buses. In our opinion, and according to ISO-OSI definitions (Perleman et al., 1988), the P-NET gateways combine techniques used in bridges and routers, and thus the term "hopping devices" is preferred.

The P-NET multi-segment feature allows for routing through up to ten hopping devices. These multi-hopping capabilities are based on simple rules for address conversion inside the hopping devices. P-NET supports four types of addresses: simple, complex, extended and response address types (see Section 3.3.1). The complex address can contain up to 24 bytes. P-NET uses the complex addressing scheme to route frames through hopping devices. This complex address explicitly addresses each intermediate device.

In P-NET, hopping devices isolate traffic between P-NET segments. If the different segments group inter-related masters and slaves, the overall real-time capabilities are improved, as the virtual token cycle time in each single segment becomes smaller. However, if a particular stream relates a master and a slave in two distinct segments, that stream will have a higher response time. We denote message streams that are relayed through at least one hopping device as multi-hop message streams. In this section, an analysis for deriving the upper bound of the response time for multi-hop message streams is provided.

4.5.1. Motivation

Suppose a P-NET network composed of four masters (M1, M2, M3, M4) and four slaves (e1, e2, e3, e4), all connected to the same network segment. Each one of the masters deals with two message streams, as shown in Fig. 4.5.

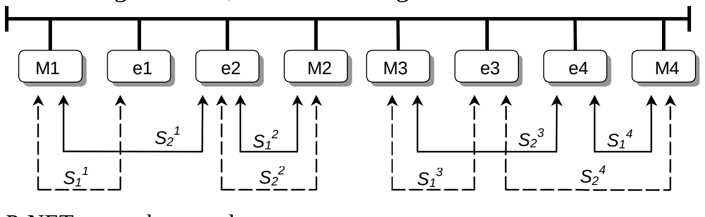


Fig. 4.5 A P-NET network example

Considering that the maximum token holding time in each station is $H = 250 \times bp$ (this means that the longest message cycles for each master is: $\max\{C^k\}=203\times bp, \forall_k$), then the pre-run-time schedulability condition (4.7) is: $D \geq 2\times 4\times 250\times bp = 2000\times 1/76800 = 26$ms.

The example in Fig. 4.5 illustrates, on a reduced scale, the advantages of segmentation. In fact, the whole network could be composed of two segments, grouping M1, M2, e1 and e2 in one segment, and M3, M4, e3 and e4 in another segment (Fig. 4.6). For simplification, any of the existing masters (M1, M2, M3 or M4) is used to implement the multi-hopping features.

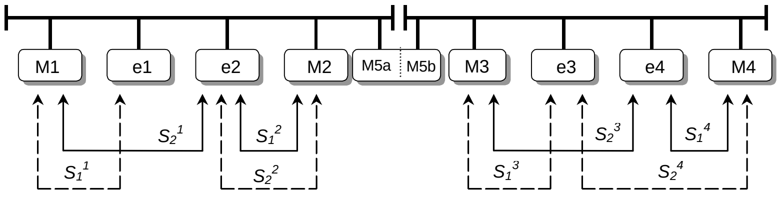


Fig. 4.6 Using one hopping device

As none of the message streams is to be relayed through the hopping device, the pre-run-time schedulability condition becomes: $D \geq 2 \times 3 \times 250 \times bp = 1500 \times 1/76800 = 19.5$ms.

However, in more complex systems, it is unlikely that all the message streams can be restricted to their own segments. As real implementations of slave nodes group several I/O points in racks, it is possible that specific information flows will demand inter-operation between masters and slaves in different network segments.

4.5.2. Sequence of Transactions in Multi-Hop Message Streams

Apart from having a longer address field, multi-hop message streams will have additional queuing delays. Fig. 4.7 illustrates the message sequence corresponding to master/slave transactions through two hopping devices. It is important to notice that each hopping device embodies two masters (and in the general case, as many masters as the number of segments that it interconnects).

If master M1 (in network segment 1) requires the reading of a sensor associated with slave e3 (in network segment 3), hopping devices M5 and M4 are used to relay the message stream. The sequence of message transactions is as follows.

1. When M1 gains access to the network (segment 1), and the message is the first one in the outgoing queue, M1 sends a request and M5a responds immediately with an “answer due later”.
2. When M5b gains access to segment 2, and the message is the first one in the outgoing queue, M5b sends the request and M4a responds immediately with an “answer due later”.
3. When M4b gains access to segment 3, and the message is the first one in the outgoing queue, M4b sends the request and slave e3 responds immediately with the requested information.

4. When M4a gains access to segment 2, and the message containing the required information is the first one in the outgoing queue, M4a sends a request without a response to M5b.
5. When M5a gains access to segment 1, and the message is the first one in the outgoing queue, M5a sends a request (containing the required information) without a response to M1.

So, in general, if *h* represents the number of intermediate hopping devices through which a message stream is to be relayed, there will be 2 × *h* + 1 queuing delays to be considered.

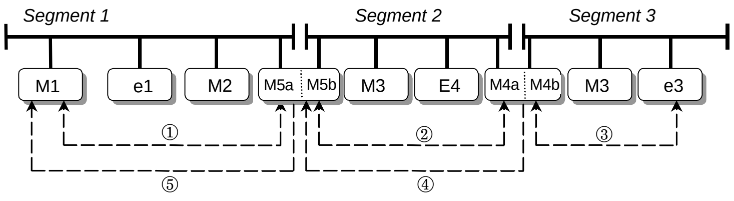


Fig. 4.7 An example of a multi-hop transaction

4.5.3. Pre-run-time Schedulability Condition for Multi-hop Message Streams

Each P-NET segment has its own virtual token-rotation procedure. Thus, the maximum virtual token cycle time in a segment *x* can be defined as:

$$V_x = \sum_{\substack{i=1 \\ i \in x}}^{n} H \tag{4.27}$$

The upper bound for the response time of a message from stream S_i^k can be derived as follows. If the message stream is to be relayed through *0* hopping devices, then R_i^k is:

$$R_i^k = ns^k \times V(k) \tag{4.28}$$

where *V(k)* corresponds to the upper bound for the virtual token rotation time of the network segment to which station *k* belongs.

If the message stream is to be relayed through *1* hopping device, then R_i^k is:

$$R_i^k = \left(ns^k + ns^{r_1}\right) \times V(k) + ns^{r_2} \times V(r_2) + 2 \times f \tag{4.29}$$

where r_1 is the hopping master in the same segment as master *k*, and r_2 is the hopping master in the other segment. The symbol *f* stands for the time needed by the hopping device to transfer frames between communication stacks.

If the message stream is to be relayed through *h* hopping devices (with $h \geq 2$), then R_i^k is:

$$R_i^k = \left(ns^k + ns^{r_1}\right) \times V(k) + \sum_{j=1}^{h-1}\left[\left(ns^{r_{2\times j}} + ns^{r_{2\times j+1}}\right) \times V\left(r_{2\times j}\right)\right] + ns^{r_{2\times h}} \times V\left(r_{2\times h}\right) + \\ + 2 \times h \times f \qquad \textbf{(4.30)}$$

where r_1 is the hopping master in the same segment as master *k* and r_2, r_3, …, $r_{2'h}$ are the hopping masters which relay the message from master *k* to its destination. For example, for the scenario shown in Fig. 4.7, r_1 = M5a, r_2 = M5b, r_3 = M4a and r_4 = M4b. As a consequence, in equation (4.30) r_i ($i = 1, \ldots, 2 \times h$) identifies the masters in hopping devices according to the physical sequence from master *k* to the addressed slave devices and not according to the sequence of transactions.

As for the non-segmented case (equations (4.28) or (4.5)), the same sufficient P-NET pre-run-time schedulability condition (inequality (4.7) – $D_i^k \geq R_i^k$) can be used to guarantee that real-time multi-hop message streams are processed before their deadlines. Depending on the number of hopping devices a message is relayed through, equation (4.28), (4.29) or (4.30) is used to evaluate the upper bound of the message response time.

4.5.4. Numerical Results

In this section, a numerical example is provided, which exemplifies what a user of P-NET can obtain from the proposed pre-run-time schedulability conditions. Although a limited number of message streams per node is assumed, some useful information can be obtained from this example:

1. how the maximum upper bound of each message-stream's response time, in both a non-segmented and segmented P-NET network can be evaluated;
2. how a P-NET network can be segmented in order to reduce the maximum upper bound for the message response time.

In this specific example, a comparison is also made between the response time's upper bound in a non-segmented and a segmented P-NET network, clearly demonstrating the impact of network segmentation.

Assume that a DCCS should be implemented using eight master networks. Also assume that all message cycle lengths are bounded to 200×*bp* (2.6ms at 76800bps). The number of streams related to each master is shown in Table 4.3 (a total of twenty-eight message streams, distributed by eight masters).

Table 4.3: Number of Message Streams Related to Each Master

Master	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
ns^k	3	4	3	2	1	4	5	6
max{C_j^k**}**	200*bp*	200*bp*	200*bp*	200*bp*	200*bp*	200*bp*	200*bp*	200*bp*

In this case, the upper bound for the virtual token cycle is $V = 8 \times 247 \times bp$ = 25.7ms. Therefore, using equation (4.28), the upper bound for the message response times is as shown in Table 4.4 (all streams in the same master will have the same upper bound for their response times). Note that the generation and delivery delays at the application process level are ignored, and must be evaluated at the level of the application process software. However, ignoring such delays is not of major importance, and as P-NET operates at 76800bps they will be usually much smaller than the transmission and queuing delays.

Table 4.4: Upper Bound for the Message Response Times

Master	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
R_i^k (ms)	77.1	102.1	77.1	51.4	25.7	102.8	128.5	154.2

If, for example, the application imposes deadlines smaller than 102.1ms for the message streams of master 2, or less than 25.7ms for the message streams of master 5, then the message stream set would not be schedulable.

Suppose that by re-organising the network into three network segments, as shown in Fig. 4.8, only two message streams are multi-hop streams: S_1^1 and S_2^8. Then, tighter deadlines can be supported for all but those two message streams.

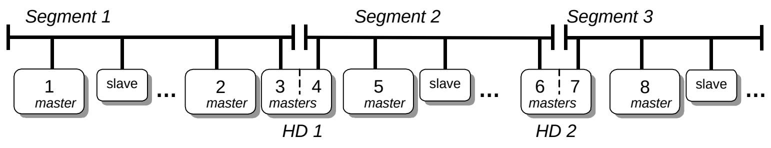


Fig. 4.8 Proposed segmentation of the network

Assume that streams S_1^1 and S_2^8 correspond to remote accesses, to slaves in segment 2 and segment 8, respectively. Table 4.5 illustrates the routing sequence for these streams.

Table 4.5: Routing Sequence (Master IDs) for the Multi-Hop Streams

	h	r_1	r_2	r_3	r_4
S_1^1	1	3	4	-	-
S_2^8	2	7	6	4	3

These two streams will impose two additional message streams (resulting from messages being relayed through the hopping device) in masters 3 and 4, and one additional message stream in masters 6 and 7. Table 4.6 reflects the aggregate number of message streams per master station.

Table 4.6: Aggregate number of message streams related to each master

Master	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
ns^k	3	4	5	4	1	5	6	6

If, for simplification, the components f in equations (4.29) and (4.30) are ignored, as well as the additional byte addresses in the multi-hop streams (thus maintaining $200 \times bp$ as the value for the longest message cycle in each master), then the results, shown in Table 4.7, are obtained.

Table 4.7: Upper Bound for the Virtual Token Cycle in Each Segment (Eq. (4.27))

Segment	**1**	**2**	**3**
V (ms)	9.65	9.65	6.43

By implementing the proposed network segmentation, an important reduction of worst-case response times can be achieved, as illustrated in Tables 4.8 and 4.9.

Table 4.8: Upper Bound for the Response Times (Single-Segment Streams Eq. (4.28))

Master	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
$\boldsymbol{R^*{}_i^k}$ (ms)	28.9	38.6	48.2	38.6	9.6	48.2	38.6	38.6

Table 4.9: Response Time Reduction as Compared to Table 4.4

Master	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
$\boldsymbol{R^*{}_i^k/R_i^k}$	36.9%	37.5%	62.6%	75.1%	37.5%	46.9%	30.0%	24.5%

Obviously, for the multi-hop message streams S_1^1 and S_2^8 the upper bounds for their response times increase, as compared to the figures given in Table 4.6 (77.1ms and 154.2ms, respectively).

The values are as follows. For R_1^1 equation (4.29) is used and Table 4.6 gives the number of streams for the masters. Its value is: $R_1^1 = (ns^1 + ns^3) \times V(1) + ns^4 \times V(4) =$ $= (3+5) \times 9.65 + 4 \times 9.65 = 115.72$ms.

For R_2^8 equation (4.30) is used and Table 4.6 gives the number of streams for the masters. Its value is: $R_2^8 = (ns^8 + ns^7) \times V(8) + (ns^6 + ns^4) \times V(6) + ns^3 \times V(3) = 12 \times 6.43 + 9 \times 9.65 + 5 \times 9.65 = 212.36$ms.

The upper bound for the response time of message stream S_1^1 (for which $h = 1$) becomes 150% higher, whereas message stream S_2^8 (for which $h = 2$) becomes 137.7% higher.

The impact of network segmentation on the response time of both single-segment and multi-hop message streams is therefore clear.

4.5.5. Considering the Actual Token Utilisation

The pre-run-time schedulability conditions presented in Section 4.5.3 may now be updated to consider the actual token utilisation (see Section 4.4).

Consider that $R^*{}_i^k$ denotes the worst-case response time for a message stream S_i^k, considering the actual token utilisation. $R^*{}_i^k$ is defined as follows:

$$R*_i^k = \left(\sum_{z=0}^{2\times h} R*_i^{r_z} \right) + 2 \times h \times f \tag{4.31}$$

where $r_0, r_1, ..., r_{2\times h}$. In Section 4.5.3, we used the notation r_i ($i \in \aleph$) to denote the master stations which relay message stream S_i^k. In this section, r_i is extended such that r_0 denotes master k itself (thus r_i: $i \in \aleph_0$).

In equation (4.31), $R^*{}_i^{r_z}$ denotes the worst-case response time of the individual transaction concerning message stream S_i^k in master r_z, and is defined as follows (see equation (4.19) for an analogy):

$$R*_i^{r_z} = ns^{r_z} \times V(r_z) - \left[\sum_{\substack{y=1 \\ y \in x_{r_z}}}^{n} \left(ns^{r_z} - \min \left\{ ns^{r_z}, ns^y + \sum_{\substack{j=1 \\ S_j^y \xleftrightarrow{\text{not relates to}} S_i^k}}^{ns^y} \left\lfloor \frac{R*_i^{r_z} + Ja^y}{T_j^y} \right\rfloor \right\} \right) \right] \times (H - s) \quad \textbf{(4.32)}$$

In equation (4.32), $y \in x_{r_z}$ denotes all masters y belonging to the same network segment as master r_z. Also in the same equation, (S_j^y <u>not relates to</u> S_i^k) denotes all streams in master y except an eventual extrinsic stream related to the relaying of stream S_i^k. $Ja^y = Jr^y - Jv^y$ is the logical ring aggregate jitter of master y as related to master r_i, as defined in Section 4.4.3. Equation (4.10), for the logical ring request jitter, and equation (4.15), for the logical ring visit jitter, must be restricted to the considered segment. Thus, equation (4.10) must be updated to:

$$Jr^y = \sum_{\substack{l=y,y^{+1},\ldots,r_z^{-1} \\ l \in x_{r_z}}} H \quad \textbf{(4.33)}$$

and equation (4.15) must be updated to:

$$Jv^y = \sum_{\substack{l=y,y^{+1},\ldots,r_z^{-1} \\ l \in x_{r_z}}} s + C_M + \sum_{\substack{l=y^{+1},\ldots,r_l^{-1} \\ l \in x_{r_z} \\ ns^{y*} \geq ns^{r_z}}} (H - s) \quad \textbf{(4.34)}$$

In equation (4.34), $ns^{y}*$ is defined as follows:

$$ns^{y*} = \begin{cases} ns^y, \text{if } y \text{ do not relay } S_i^k \\ ns^y - 1, \text{ if } y \text{ is used to relay } S_i^k \end{cases} \quad \textbf{(4.35)}$$

The consideration of the actual token utilisation for multi-hop message streams is now compared to the previous analysis in the following example. Consider again the same multi-hop example (Fig. 4.8), where the timing characteristics of the message streams are as shown in Table 4.10. The only streams that are multi-hop are streams S_1^1 and S_2^8, with a relaying table as shown in Table 4.5.

The worst-case response times for the single segmented streams, considering the actual token utilisation are as shown in Table 4.10. We have used a software tool based on the algorithm given in Appendix A.1. Therefore, we have excluded the eventual extrinsic streams related with the relaying of stream S_i^k, and also the algorithm for the evaluation of the visit jitter was based on equation (4.15) and not on equation (4.34).

Table 4.10: Characterisation of the Message Stream Set

Master	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
$\boldsymbol{Ns^k}$	3	4	3	2	1	4	5	6
Max{C_j^k}	200*bp*	200*bp*	200*bp*	200*bp*	200*bp*	200*bp*	200*bp*	200*bp*
$\boldsymbol{T_i^k}$ **(ms)**	300	200	500	200	100	500	500	500

Table 4.11: Upper Bound for the Response Times (Single-Segment Streams Eq. (4.5))

Master	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
$\boldsymbol{R^{\S}{}_i^k}$ (ms)	28.9	35.5	38.9	29.3	9.6	32.8	38.6	38.6

For stream S_1^1, $R*_1^1$ = 28.9 + 29.3 + 38.9 = 97.1ms. For stream S_2^8, $R*_2^8$ = 38.6 + 32.8 + 38.9 + 29.3 + 38.6 = 178.2ms.

The results shown in Tables 4.2, 4.8 and 4.11, for a non-segmented network, a segmented network (as shown in Fig. 4.6) and a segmented network considering the actual token utilisation, respectively, are compared in the graph of Fig. 4.9.

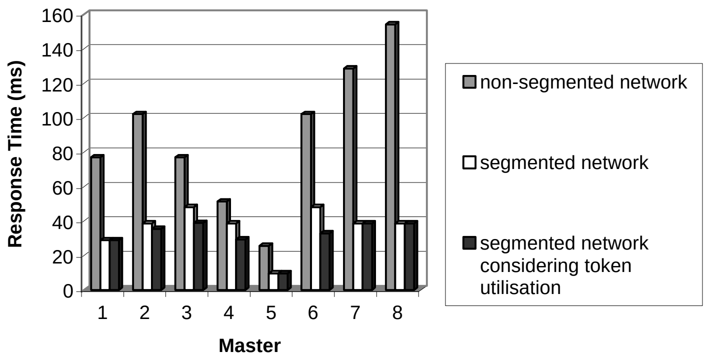


Fig. 4.9 This figure compares the worst-case response times for the example, resulting from the different analysis drawn in this chapter

4.6. Summary

P-NET networks aim at the interconnection of field devices such as sensors, actuators and small controllers. Therefore, they may be a privileged basis upon which Distributed Computer-Controlled Systems (DCCS) are built. DCCS impose strict timeliness requirements to the communication network; that is, they impose that traffic must be sent and received within a bounded interval, otherwise a timing fault is said to occur.

P-NET is a multi-master fieldbus standard based on a virtual token passing scheme. In P-NET each master is allowed to transmit only one message cycle per token visit. This means that, in the worst-case, the communication response time of a message can be derived considering that the token is fully utilised by all stations. Based on this consideration, in Section 4.3, we introduce the concept of worst-case token rotation time, and we provide a simple analytical formulation for expressing the response time of a P-NET message (equation (4.5)). This equation allows the implementation of a simple feasibility test (equation (4.7)) for P-NET message streams.

However, such analysis was proved to be quite pessimistic. Therefore, in Section 4.4, we propose a more sophisticated P-NET model, which considers the actual token utilisation by the different network masters. The major contribution of this model is to provide a less pessimistic, and thus more accurate, analysis for the evaluation of the worst-case communication response time of P-NET messages (equation (4.18)). This equation allows for the implementation of a more accurate feasibility test (equation (4.21)) for P-NET message streams.

Finally, in Section 4.5 we extended the analysis performed in Sections 4.3 and 4.4 to the case of multi-hop P-NET networks. We showed how, by using P-NET hopping devices, a significant reduction on response times could be achieved for most of the message transactions. In such way, tighter message deadlines can be supported. However, we stress that the system designer must clearly understand that such reductions are not possible for inter-segment message transactions. Therefore, care should be taken to group masters and slaves involved in message transactions with stringent deadlines in the same network segment.

4.7. References

Liu, C. and Layland, J. (1973). Scheduling Algorithms for Multiprograming in Hard-Real-Time Environment. In *Journal of the ACM*, Vol. 20, No. 1, pp. 46-61.

Perleman, R., Harvey, A., and Varghese, G. (1988). Choosing the Appropriate ISO Layer for LAN Interconnection. In *IEEE Network*, Vol. 2, No. 1, pp. 81-86.

Tovar, E. and Vasques, F. (1998a). A Communication Support for Real-Time Distributed Computer Controlled Systems. In *Proceedings of the IEE International Workshop on Discrete Event Systems*, pp.178-183. Published by IEE.

Tovar, E., Vasques, F. and Burns, A. (1998b). Supporting Real-Time Distributed Computer-Controlled Systems with Multi-hop P-NET Networks. Polytechnic Institute of Porto, Technical Report HURRAY-TR-9813, September 1998, to appear in *Control Engineering Practice*, Pergamon Publishers.

Tovar, E., Vasques, F. and Burns, A. (1999). Communication Response Time in P-NET Networks: Worst-Case Analysis Considering the Actual Token Utilisation. Department of Computer Science, University of York, Technical Report YCS 312, January 1999, submitted for publication to the *Journal of Real-Time Systems*, Kluwer Academic Publishers.

Chapter 5

Real-Time Communications with PROFIBUS Networks: Contributions to the State-of-the-Art

In this chapter we develop a methodology for the worst-case response time analysis of PROFIBUS messages. This chapter is largely drawn from the following published work: "Real-Time Fieldbus Communications Using PROFIBUS Networks" (Tovar and Vasques, 1998a); "Guaranteeing Real-Time Message Deadlines in PROFIBUS Networks" (Tovar and Vasques, 1998b); "Cycle Time Properties of the PROFIBUS Timed Token Protocol" (Tovar and Vasques, 1998c) and "Setting Target Rotation Time in PROFIBUS Based Real-Time Distributed Applications" (Tovar and Vasques, 1998d).

5.1. Preliminary Protocol Analysis

The PROFIBUS medium access control (MAC) protocol is based on a token passing procedure, used by master stations to grant bus access to each other, and a master-slave procedure used by master stations to communicate with slave stations. Please refer to Section 3.4.1 for a detailed description of the main characteristics of this protocol.

The remainder of this chapter is organised as follows. In Section 5.2 we introduce the network and message models, which will be used throughout the rest of the chapter. Considering a profile where the low-priority traffic is unconstrained, the worst-case response time of PROFIBUS messages depends on the accurate definition of the maximum token cycle time. This response time analysis is outlined in Section 5.3, and, due to its relevance, the token cycle time properties are assessed in Section 5.4. Finally, in Section 5.5, a constrained low-priority traffic profile is considered, where, by controlling the number of low-priority message transfers, all pending real-time traffic is transmitted at each token arrival. A worst-case response time analysis is therefore provided for this profile and some implementation issues are also discussed.

5.2. Network and Message Models

Consider a bus topology containing n master stations. A special frame (the token) circulates around the logical ring formed by the masters. We denote t as the logical ring latency (token walk time, including node latency delay, media propagation delay, etc.).

Message cycles generated in the system at run-time may be placed in a high-priority outgoing queue (real-time messages) or in a low-priority outgoing queue (non real-time messages). We denote the i^{th} (i = 1, 2, … nh^k) high-priority message stream associated to a master k as Sh_i^k, which is characterised as follows:

$$Sh_i^k = (Ch_i^k, Th_i^k, Dh_i^k) \tag{5.1}$$

A message stream is a temporal sequence of message cycles concerning, for instance, the remote reading of a specific process variable. Ch_i^k is the longest message cycle duration of stream Sh_i^k. This duration includes the time needed to transmit the request frame and completely receive the related response, and also the time needed to perform the allowed number of message retries. Th_i^k is the periodicity of stream Sh_i^k requests. In order to have a subsequent timing analysis independent from the model of the tasks at the application process level, we assume that this periodicity is the minimum interval between two arrivals of Sh_i^k requests to the outgoing queue. Dh_i^k is the relative deadline of the message cycle; that is, the maximum admissible time interval between the instant when the message request is placed in the outgoing queue and the instant when the related response is completely received at the master's incoming queue. Finally, nh^k denotes the number of high-priority message streams associated with a master k.

We assume only one low-priority message stream per master k, which gathers all the non real-time traffic issued by that master (see Section 3.4.1). Thus, a low-priority message stream Sl^k is characterised as:

$$Sl^k = (Cl^k, nlp^k) \tag{5.2}$$

where Cl^k is the maximum amount of time required to perform a low-priority message cycle in master k. nlp^k is the maximum number of low-priority message cycles that a master k is allowed to perform at each token arrival, which is a meaningful parameter to constrain the low-priority traffic.

5.3. Analysis for the Worst-Case Response Time

In the PROFIBUS MAC protocol, and as long as there are pending high-priority messages, a master station is guaranteed to transmit, at least, one high-priority message per received token (even if there is not enough token holding time left). Assuming the worst-case scenario (token always arriving late), if there are m pending high-priority messages, it will take m token visits to execute all those high-priority message cycles. The worst-case is when the message with the shortest deadline becomes the last one in the FCFS outgoing queue. Fig. 5.1 illustrates this scenario.

In this scenario, when master *1* sends the token to master *2*, messages from all the nh^1 streams arrive at the FCFS outgoing queue in the following order: Sh_4^1, Sh_3^1, Sh_2^1, Sh_1^1. Considering that the message deadlines are as follows: $Dh_4^1 > Dh_3^1 > Dh_2^1 > Dh_1^1$, the most stringent message (Sh_1^1) will wait until the fourth token visit to be transmitted.

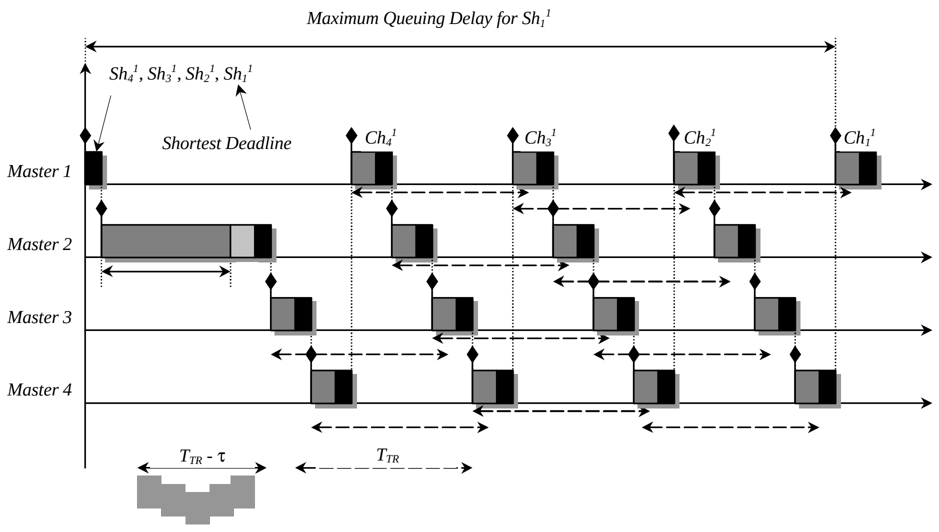


Fig. 5.1 Example of the maximum queuing delay for a PROFIBUS message

For the queuing delay analysis, it is important to note that the maximum number of pending messages will be nh^k, corresponding to one message per each Sh_i^k stream. Indeed, if at any time there are two pending message requests from the same stream, then a deadline for that message stream was missed.

We denote the worst-case response time of a stream Sh_i^k as R_i^k. This time is measured starting at the instant when the request is placed in the outgoing queue, until the instant when the response is completely received at the incoming queue. Basically, this time interval is made up of the two following components.

1. The time spent by the request in the outgoing queue, until gaining access to the bus (queuing delay);
2. The time needed to process the message cycle.

Thus,

$$R_i^k = Q_i^k + Ch_i^k \tag{5.2}$$

where Q_i^k is the worst-case queuing delay of a message request from Sh_i^k.

It is clear that, assuming that message deadlines are not missed (thus the maximum number of high-priority pending messages is nh^k), the upper bound for the message queuing delay in a master k is

$$Q^k = nh^k \times T_{cycle}^k \tag{5.3}$$

where T^k_{cycle} is the upper bound for the token inter-arrival time at a station k, hence the upper bound for the real token rotation time (T^k_{RR}). Note that, under our assumptions, the queuing delay for a message request in one station is independent of the message stream ($Q_i^k = Q^k$, $\forall_{i=1,..., nh^k}$).

Combining equations (5.2) and (5.3), the worst-case response time of a PROFIBUS high-priority message is:

$$R_i^k = nh^k \times T_{cycle}^k + Ch_i^k \qquad \textbf{(5.4)}$$

Therefore, a pre-run-time schedulability condition for the high-priority message stream set is:

$$Dh_i^k \geq nh^k \times T_{cycle}^k + Ch_i^k \qquad \textbf{(5.5)}$$

5.4. PROFIBUS Token Cycle Time Analysis

From the proposed pre-run-time schedulability condition (5.5), it follows that an accurate evaluation of the T^k_{cycle} parameter is of paramount importance for the response time analysis in PROFIBUS networks. Therefore, such evaluation is the main focus of this section.

5.4.1. Analysis of the PROFIBUS Token Lateness

In PROFIBUS networks, the real token rotation time (T^k_{RR}) is always smaller than T_{TR}, except when one or more masters in the logical ring induce the token to be late. Two reasons justify a late token arrival at a master *k*.

1. As once a message cycle is started, it is always completed, even if the T^k_{TH} timer has expired during its execution, a late token may be transmitted to the following masters. We define this occurrence as an overrun of the T^k_{TH} overrun timer.
2. If a master receives a late token, it will still be able to transmit one high-priority message, which may further increase the token lateness in the following masters. This case is not considered to be an overrun of the T^k_{TH} timer.

In this section, we analyse causes and consequences of the token lateness. We will introduce and prove three theorems. Theorem 5.1 states that the token is never late unless an overrun of T_{TH} occurs in one of the masters that form the logical token-passing ring. Theorem 5.2 states that even if more than one master overruns its T_{TH} in a token cycle, only the last one (as seen from the master for which T_{RR} is being measured) will contribute to the token delay. Finally, Theorem 5.3 states that, in a specific situation, all masters may contribute to the token lateness. These three theorems are the basis for the evaluation of T^k_{cycle} (an upper bound for T^k_{RR}).

Theorem 5.1 In PROFIBUS networks, if the master holding the token releases it before the expiration of T^k_{TH}, then, the following master in the logical ring will receive an early token.

Proof: We denote $A^k(l)$ as the time instant when the token arrives to the master *k* for its l^{th} visit, and $R^k(l)$ as the time instant when master *k* releases the token in that l^{th} visit.

If master *k* releases the token before the expiration of T^k_{TH} then, $R^k(l)-A^k(l-1)<T_{TR}$. Note that the real token rotation time is measured between token arrivals. Therefore, at the time instant $A^k(l)$, T^k_{TH} is given the value $T_{TR}-T^k_{RR}$, a positive value (the token does

not arrive late). If at the time instant $R^k(l)$ master k releases the token and the T^k_{TH} timer has not yet reached 0, then $R^k(l)$-$A^k(l$-1)<T_{TR}. This equation is the starting assumption for the remainder of the proof.

We denote the successor of master k as master k^{+1} (with k^{+1}=(k+1) mod n). We want to prove that if $R^k(l)$-$A^k(l$-1)<T_{TR} is true, then $A^{k+1}(l)$-$A^{k+1}(l$-1)<T_{TR} is also true; that is, after releasing the token at time instant $R^k(l)$, the successor of k will receive an early token.

As $A^{k+1}(l$-1)=$R^k(l$-1)+t/n (since t denotes the total logical ring latency, k^{+1} will receive the token t/n time after the release of the token in k) and as $A^{k+1}(l)$=$R^k(l)$+t/n then, combining these two expressions, it follows that $A^{k+1}(l)$-$A^{k+1}(l$-1)=$R^k(l)$-$R^k(l$-1).

The starting assumption is that $R^k(l)$-$A^k(l$-1)<T_{TR}. As $R^k(l$-1)=$A^k(l$–1)+t/n, it is true saying that $R^k(l$-1)>$A^k(l$-1). Therefore, if $R^k(l)$-$A^k(l$-1)<T_{TR} then $R^k(l)$-$R^k(l$-1)<T_{TR}.

As $A^{k+1}(l)$-$A^{k+1}(l$-1)=$R^k(l)$-$R^k(l$-1), if $R^k(l)$-$R^k(l$-1)<T_{TR} then $A^{k+1}(l)$-$A^{k+1}(l$-1)<T_{TR} is true; that is, the successor of k receives an early token. ❑

Fig. 5.2 illustrates Theorem 5.1, where master 2 releases the token before the expiration of $T^2_{TH}(l)$, and so master 3 receives an early token.

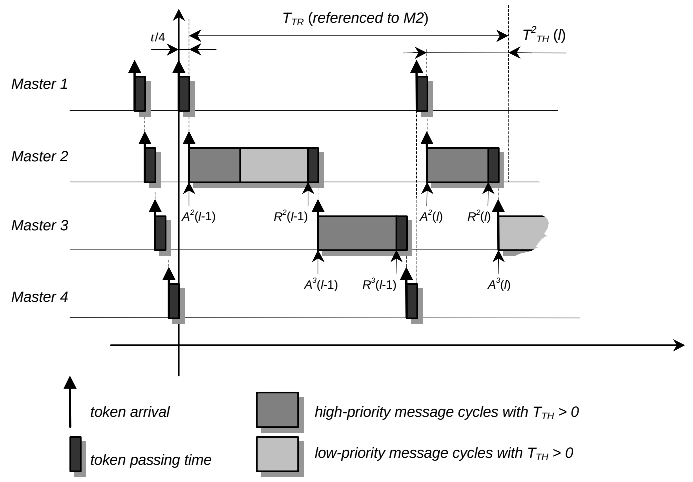


Fig. 5.2 An illustrative example for Theorem 5.1

From Theorem 5.1, two corollaries result.

Corollary 5.1 In PROFIBUS, a master k receives an early token if master k^{-1} releases it before the T^{k-1}_{TH} expiration, even if there was any overrun of a T_{TH} in masters k^{-2}, k^{-3}, …

Corollary 5.2 In PROFIBUS, if none of the masters overrun their T_{TH}, the token will never be late.

Theorem 5.2 In a PROFIBUS network, in a specific token cycle, only one overrun of T_{TH} contributes to the token lateness.

Proof: Assume that a token delay is induced in the l^{th} token cycle. Hence, the token will arrive late in the next token cycle. Consider the analysis focused on master k, and the measurement of the time elapsed between $A^k(l)$ and $A^k(l+1)$, that is, between two consecutive token arrivals to master k (T^k_{RR}).

If $k\neq1$, then the masters that may induce a delay in the token are, in sequence of token holding, the master k itself and all other masters up to master n, in the l^{th} token rotation, and master 1 and all masters up to master k^{-1} in the $(l+1)^{th}$ token rotation. If k=1, then the masters that may induce a delay in the token are, in sequence of token holding, the master k itself and all other masters up to master n, all in the l^{th} token rotation.

For simplification of this proof, and without loss of generality, we assume that k=1. In this case, the last master, before the $(l+1)^{th}$ visit of the token to master 1, which may produce an overrun of the T_{TH}, is master n, hence an overrun in $T^n_{TH}(l)$.

If in the l^{th} visit to master n an overrun of T^n_{TH} occurs, then $A^n(l)$-$A^n(l$-1$)\leq T_{TR}$; that is, the token arrived early to master n. If we denote $b^n(l)$ as the time instant when T^n_{TH} expires during the l^{th} visit to master n, then, as $A^n(l)>A^n(l$-1), it follows that $b^n(l)$-$A^k(l)\leq T_{TR}$, no matter if other overruns have occurred in the l^{th} rotation of the token in any of the predecessors of master n. Thus, only one overrun may contribute to the token lateness. ❑

Fig. 3 illustrates Theorem 2, where n is set to 4 and k is set to 1. In this illustrative example, two overruns occur in the l^{th} token rotation, in master 1 and in master 4. Only the last one before $A^1(l+1)$, the one that took place in master 4, contributes to the lateness of the token arriving to master 1 at $A^1(l+1)$.

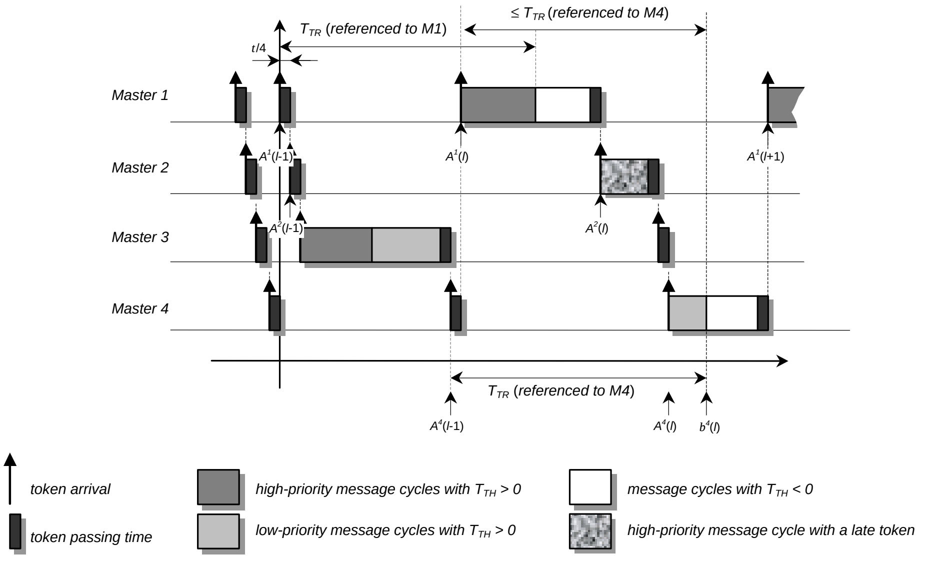

≤ T_{TR} (referenced to M4)
T_{TR} (referenced to M1)
t/4
Master 1
Master 2
Master 3
Master 4
$A^1(l$-1)
$A^1(l)$
$A^1(l$+1)
$A^2(l$-1)
$A^2(l)$
T_{TR} (referenced to M4)
$A^4(l$-1)
$A^4(l)$
$b^4(l)$
token arrival
token passing time
high-priority message cycles with T_{TH} > 0
low-priority message cycles with T_{TH} > 0
message cycles with T_{TH} < 0
high-priority message cycle with a late token

Fig. 5.3 An illustrative example for Theorem 5.2

Theorem 5.3 If a PROFIBUS master k holds the token for an interval greater than T_{TR}-t, all the following masters up to master k^{-1} will receive a late token.

Proof: Due to the token-passing time and other network latencies, it follows that $A^k(l)-A^{k+1}(l-1)\geq((n-1)/n)\times t$; that is, the difference between the token arrival to a master k and the token arrival to its successor in the previous token cycle is at least $((n-1)/n)\times t$ (corresponding to n-1 token-passing times).

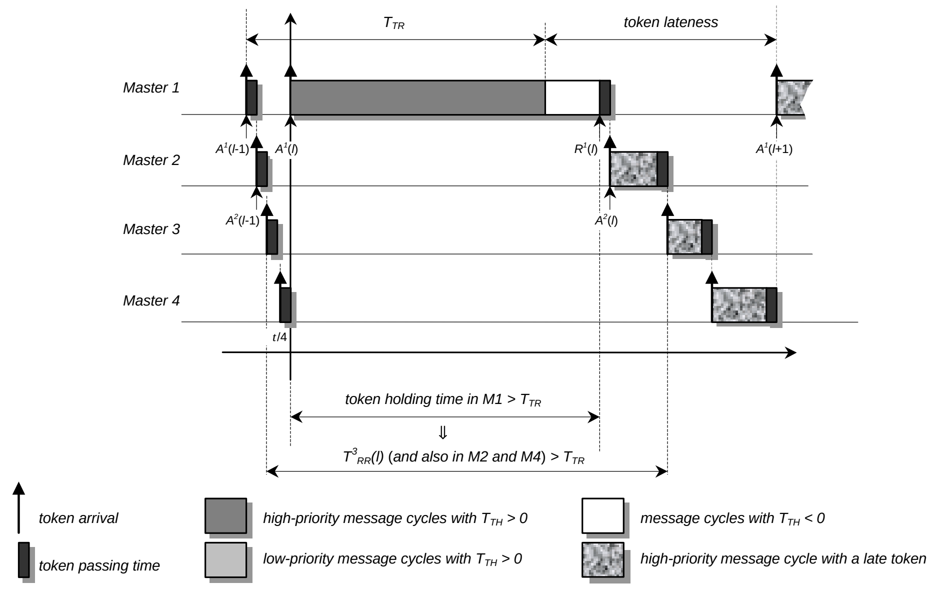
$A^k(l)-A^{k+1}(l-1)\geq((n-1)/n)\times t$ can be re-written as $A^{k+1}(l-1)\leq A^k(l)-((n-1)/n)\times t$. As the master k holds the token for an interval greater than $T_{TR}-t$, then $R^k(l)>A^k(l)+T_{TR}-t$.

It is also evident that the arrival of the token to master k^{+1} occurs at $A^{k+1}(l)=R^k(l)+t/n$, that is at the time the token is released in k added to the time to pass the token to k^{+1}.

Thus, if we replace $R^k(l)$ in the equation $(A^{k+1}(l)=R^k(l)+t/n)$ with the inequality $(R^k(l)>A^k(l)+T_{TR}-t)$, it follows that $A^{k+1}(l)>A^k(l)+T_{TR}-((n-1)/n)\times t$. Hence, using this last inequality and knowing that $A^k(l)-A^{k+1}(l-1)\geq((n-1)/n)\times t \Leftrightarrow A^{k+1}(l-1)\leq A^k(l)-((n-1)/n)\times t$, it follows that $A^{k+1}(l)-A^{k+1}(l-1)>A^k(l)+T_{TR}-((n-1)/n)\times t-A^k(l)+((n-1)/n)\times t=T_{TR}$.

Obviously this result extends to all the following masters which range from master k^{+2} up to master k^{-1}. The starting assumption is that the token holding time in master k is $R^k(l)-A^k(l)>T_{TR}-t$. As the token arrived in the other masters before $A^k(l)$, and after its release from master k at time instant $R^k(l)$ it will arrive at the other masters after $R^k(l)$, the token rotation time as measured in the other masters will give, for all of them, a value greater than T_{TR}. ❑

Fig. 5.4 illustrates Theorem 5.3, where k is set to 1.



T_{TR}
token lateness
Master 1
Master 2
Master 3
Master 4
$A^1(l-1)$
$A^1(l)$
$R^1(l)$
$A^1(l+1)$
$A^2(l-1)$
$A^2(l)$
t/4
token holding time in M1 > T_{TR}
⇓
$T^3_{RR}(l)$ (and also in M2 and M4) > T_{TR}
token arrival
token passing time
high-priority message cycles with T_{TH} > 0
low-priority message cycles with T_{TH} > 0
message cycles with T_{TH} < 0
high-priority message cycle with a late token

Fig. 5.4 An illustrative example for Theorem 5.3

5.4.2. Evaluation of the Token Cycle Time

By using Theorem 5.3, we can define the token lateness in a master k (T^k_{del}) as the maximum excess to T_{TR} at the token arrival to the master k. The token cycle time is then given by:

$$T^k_{cycle} = T_{TR} + T^k_{del} \tag{5.6}$$

Assuming, for simplification, that all the message cycles have the same duration (represented by C_s) then, the worst-case token lateness in a master k would result from the simultaneous occurrence of the three following conditions.

1. The time interval during which the master k actually holds the token is greater than T_{TR}-t.
2. The master k itself causes an overrun of T^k_{TH}, and this overrun starts with a residual value of T^k_{TH}.
3. All the following masters (until the master k^{-1}) transmit, each one, one high-priority message cycle, having received a late token.

Observed these three conditions, in the next token cycle, T^k_{RR} reaches its upper bound, which is T^k_{cycle}. In the case of equal length for all the message stream cycles, T^k_{del}=n×C_s, and thus:

$$T^k_{cycle} = T_{TR} + n \times C_s\ ,\ \forall_{master\ k} \tag{5.7}$$

This evaluation of the worst-case token cycle time improves the previous available results from (Vasques, 1996; Vasques and Juanole, 1994) presented in equation 3.6. This evaluation of T^k_{cycle} will now be improved to consider message cycles with different lengths.

In the general case (message cycles with different duration), the worst-case token lateness may result not from an overrun of the T_{TH} in master k but from one occurring in one of the following masters (k^{+1} until k^{-1}).

Using Fig. 5.5 as an illustrative example, assume that master 1 does not overrun its T^1_{TH} (Fig. 5.5b). Then, master 2 may use its available token holding time and produce an overrun of its T^2_{TH} overrun timer. If this overrun is longer than the sum of the longest overrun of T^1_{TH} added to the longest Ch^2_i (Fig. 5.5a), then this would lead to a higher value for T^1_{del}. Similarly, if the longest overrun of T^3_{TH} is longer than the longest overrun of T^2_{TH} added to the longest Ch^3_i, then this would lead to a higher value for T^1_{del}.

Note that by Theorems 5.2 and 5.3, for the evaluation of T^k_{del}, we can only consider one overrun in master j (with j ranging from master k to master k^{-1}), and one high-priority message cycle per each master whose address is between j and k^{-1}. Consequently, we conclude that the evaluation of T^k_{cycle} depends on which master produces the worst-case overrun of its T_{TH} and on its relative position within the logical ring sequence of token-passing.

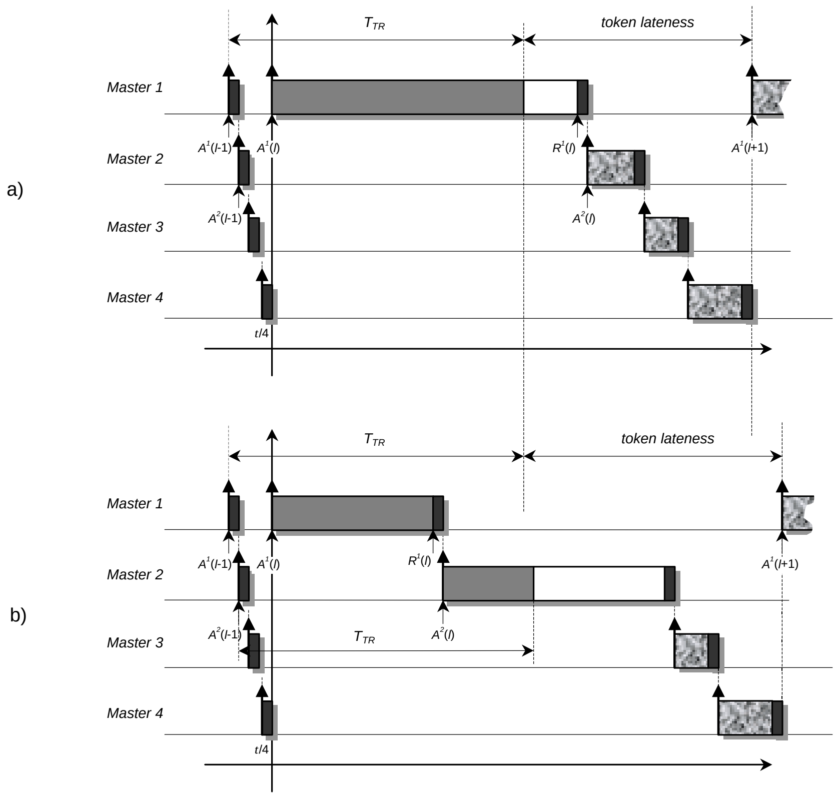


Fig. 5.5 A comparison between two scenarios with overrunning of T_{TH}

For the evaluation of T^k_{del}, we introduce the following parameters: H^k, L^k and A^k. H^k is the longest high-priority message cycle that can be requested by a master k:

$$H^k = \max_{i=1,..,nh^k} \left\{ Ch_i^k \right\} \qquad \textbf{(5.8)}$$

L^k is the longest low-priority message cycle that can be requested by a master k:

$$L^k = \max_{i=1,..,nl^k} \left\{ Cl_i^k \right\} \qquad \textbf{(5.9)}$$

Finally, A^k is the longest message cycle that can be requested by a master k (including both low and high-priority message cycles):

$$A^k = \max \left\{ H^k, L^k \right\} \qquad \textbf{(5.10)}$$

Using the analysis outlined in this section, we can thus define the maximum token lateness in a PROFIBUS master k (T^k_{del}) as being:

$$T_{del}^{k} = \max_{j \in f_1} \left\{ A^{j} + \sum_{i \in f_2} H^{i} \right\} \tag{5.11}$$

where f_1 is defined as being the following set of values:

$$f_1 = \begin{cases} \{k, \dots, n\}, & \text{if } k = 1 \\ \{k, \dots, n, 1, \dots, k-1\}, & \text{if } k > 1 \end{cases} \tag{5.12}$$

and f_2 is defined as being the following set of values:

$$f_2 = \begin{cases} \{j+1, \dots, n\}, & \text{if } k = 1 \\ \{j+1, \dots, n, 1, \dots, k-1\}, & \text{if } k > 1 \end{cases} \tag{5.13}$$

Consequently, the worst-case token cycle time in a PROFIBUS fieldbus network may be defined as follows:

$$T_{cycle}^{k} = T_{TR} + \max_{j \in f_1} \left\{ A^{j} + \sum_{i \in f_2} H^{i} \right\} \tag{5.14}$$

In Appendix B.1 we give the pseudo code details of an algorithm used for evaluating T_{del}^{k}.

5.4.3. Setting the Target Token Rotation Time

Based on the evaluation of the token cycle time (equations (5.6) and (5.14)), the pre-run-time schedulability condition (5.5) for the high-priority message stream set, can be re-written as follows:

$$Dh_i^k \geq nh^k \times \left(T_{TR} + T_{del}^{k} \right) + Ch_i^k, \forall_{master\ k,\ stream\ Sh_i^k} \tag{5.15}$$

and, the following condition for setting the T_{TR} parameter can be used:

$$0 \leq T_{TR} \leq \frac{Dh_i^k - Ch_i^k}{nh^k} - T_{del}^{k}, \forall_{master\ k,\ stream\ Sh_i^k} \tag{5.16}$$

Note that that inequality (5.15) constrains D_i^k to be larger than a value which is proportional to the number of high-priority message streams in master *k*. This may be very restrictive for stations dealing with a large number of I/O points.

In Chapter 7, we will present some solutions to overcome this problem, which, like in P-NET networks, results from the FCFS characteristics of the PROFIBUS outgoing queues.

5.4.4. Numerical Example

Consider a PROFIBUS network with 3 masters, each one with the following message streams:

Table 5.1: A Numerical Example

Master 1	Master 2	Master 3
Ch_1^1=8 ms	Ch_1^2=8 ms	Ch_1^3=8 ms
Ch_2^1=6 ms	Ch_2^2=15 ms	Ch_2^3=18 ms
Ch_3^1=7 ms	-	-
Cl_1^1=10 ms	Cl_1^2=30 ms	-
-	Cl_2^2=18 ms	-

For this numerical example, the results for each T^k_{del} are (using equation 5.11 - detailed in an algorithmic form in Appendix B.1):

Table 5.2: T^k_{del} Evaluation for the Numerical Example (case of $T_{TR} > t$)

Master 1	Master 2	Master 3
H^1=8 ms	H^2 = 15 ms	H^3 = 18 ms
A^1=10 ms	A^2 = 30 ms	A^3 = 18 ms
T^1_{del}=A^2+H^3=48 ms	T^2_{del}=A^2+H^3+H^1=56 ms	T^3_{del}=A^3+H^1+H^2=41 ms

Consider that t=1ms. If we assume that the minimum value for T_{TR} should be marginally greater than t (otherwise low-priority traffic would not be transferred at all), then inequality (5.16) can be re-written as:

$$t \le T_{TR} \le \frac{Dh_i^k - Ch_i^k}{nh^k} - T_{del}^k, \forall_{master\ k,\ stream\ Sh_i^k} \quad \textbf{(5.17)}$$

Then, to evaluate the smaller value for each message's deadline we can use the following inequality:

$$\frac{Dh_i^k - Ch_i^k}{nh^k} - T_{del}^k > t \quad \textbf{(5.18)}$$

which can be re-written as:

$$Dh_i^k > \left(t + T_{del}^k\right) \times nh^k + Ch_i^k \quad \textbf{(5.19)}$$

Using inequality (5.19), the minimum deadline supported for each high-priority message stream (Table 5.1) would be as follows:

Table 5.3: Minimum Admissible Deadlines (case of $T_{TR} = t$)

Master 1	Master 2	Master 3
Dh_1^1>155 ms	Dh_1^2>122 ms	Dh_1^3>92 ms
Dh_2^1>153 ms	Dh_2^2>129 ms	Dh_2^3>102ms
Dh_3^1>154 ms	-	-

From inequality (5.16), it is obvious that T_{TR} can be set to a value as small as *0*. In this case however the low-priority traffic would not be transferred at all. It also follows that in this non-realistic situation, the low-priority traffic would not be considered for the evaluation of T^k_{del}. In fact, if T_{TR} is smaller than t, then equation (5.20) must replace equation (5.11) for the evaluation of T^k_{cycle}.

$$T^k_{del} = \sum_{i=1}^{n} H^i \tag{5.20}$$

It follows that T^k_{cycle} will have the same value for all masters. Using the same scenario as shown in Table 5.1, each T^k_{del} would then be:

Table 5.4: T^k_{del} Evaluation for the Numerical Example (case of $T_{TR} = 0$)

Master 1	**Master 2**	**Master 3**
H^1=8 ms	H^2=15 ms	H^3=18 ms
T^1_{del}=41 ms	T^2_{del}=41 ms	T^3_{del}=41 ms

and the minimum deadline supported for each high-priority stream, would be as shown in Table 5.5.

Table 5.5: Minimum Admissible Deadlines (case of $T_{TR} = 0$)

Master 1	**Master 2**	**Master 3**
Dh_1^1>131 ms	Dh_1^2>90 ms	Dh_1^3>90 ms
Dh_2^1>129 ms	Dh_2^2>97 ms	Dh_2^3>100 ms
Dh_3^1>130 ms	-	-

5.5. Constraining Low-Priority Traffic in PROFIBUS Networks

In this section, a constrained low-priority traffic profile is considered for PROFIBUS networks, where by controlling the number of low-priority message transfers, all pending high-priority (real-time) traffic is transmitted at each token arrival.

5.5.1. Pre-Run-Time Schedulability Condition

Considering that all high-priority messages are to be sent at each token visit, it is a sufficient pre-run-time schedulability condition to guarantee the deadlines of the high-priority message stream set that:

$$\min_{i=1,\dots,nh^k} \left\{ Dh_i^k \right\} \geq T_{cycle}, \forall_{\text{master } k} \tag{5.21}$$

Furthermore, as each master must be able to transmit its high-priority traffic and, if possible, its allowed low-priority traffic, an upper bound for the token cycle time is:

$$T_{cycle} = \sum_{k=1}^{n}\sum_{i=1}^{nh^k} Ch_i^k + \sum_{k=1}^{n}\left(nlp^k \times Cl^k\right) + t \qquad \textbf{(5.22)}$$

where $nlp^k \times Cl^k$ (see Section 5.2) corresponds to the maximum ammount of low-priority traffic that can be transmitted at each token visit to a master *k*. Therefore, the pre-run-time schedulability condition (5.21) can be re-written as follows:

$$\min_{i,k}\left\{Dh_i^k\right\} \geq \sum_{k=1}^{n}\sum_{i=1}^{nh^k} Ch_i^k + \sum_{k=1}^{n}\left(nlp^k \times Cl^k\right) + t \qquad \textbf{(5.23)}$$

5.5.2. Setting the T_{TR} Parameter

The T_{TR} parameter must be set in order to guarantee that, at the token arrival, there will be always enough time to execute all pending high-priority traffic. This means that, at the token arrival, $T_{TH} = T_{TR} - T_{RR}$ must be enough to transmit all high-priority traffic, i.e.:

$$T_{TR} \geq Tcycle + \max_{k=1..n}\left\{\sum_{i=1}^{nh^k} Ch_i^k\right\} \qquad \textbf{(5.24)}$$

since T_{cycle} is the upper bound for T_{RR}.

Combining inequalities (5.22) and (5.24), a lower bound for T_{TR} is given by:

$$T_{TR} \geq \sum_{k=1}^{n}\sum_{i=1}^{nh^k} Ch_i^k + \sum_{k=1}^{n}\left(nlp^k \times Cl^k\right) + t + \max_{k=1..n}\left\{\sum_{i=1}^{nh^k} Ch_i^k\right\} \qquad \textbf{(5.25)}$$

Inequalities (5.23) and (5.25) are the basis for setting the T_{TR} parameter, when considering the constrained low-priority traffic profile. It can also be seen that an implicit upper bound for the T_{TR} parameter is imposed by the deadline constraint (5.21). In fact, we can denote as T_{TRmin} the minimum value of T_{TR} that satisfies inequality (5.25):

$$T_{TR_{\min}} = \sum_{k=1}^{n}\sum_{i=1}^{nh^k} Ch_i^k + \sum_{k=1}^{n}\left(nlp^k \times Cl^k\right) + t + \max_{k=1..n}\left\{\sum_{i=1}^{nh^k} Ch_i^k\right\} \qquad \textbf{(5.26)}$$

which, by using equation (5.22), can be re-written as:

$$T_{TR_{\min}} = T_{cycle} + \max_{k=1..n}\left\{\sum_{i=1}^{nh^k} Ch_i^k\right\} \qquad \textbf{(5.27)}$$

Therefore, the deadline constraint (5.21) can be re-defined to integrate such lower bound for T_{TR} as follows:

$$\min_{i=1,\ldots,nh^k}\left\{Dh_i^k\right\} \geq T_{TR_{\min}} - \max_{k=1..n}\left\{\sum_{i=1}^{nh^k} Ch_i^k\right\}, \forall_k \qquad \textbf{(5.28)}$$

which imposes an upper bound for the for T_{TR} parameter:

$$T_{TR_{\min}} \leq \min_{i=1,\dots,nh^k}\left\{Dh_i^k\right\} + \max_{k=1..n}\left\{\sum_{i=1}^{nh^k} Ch_i^k\right\}, \forall_k \tag{5.29}$$

Inequality (5.29) gives an upper bound for the T_{TR} parameter, while inequality (5.25) gives a lower bound for the same parameter. This allows the formulation of a pre-run-time schedulability condition, which gives the range of T_{TR} values satisfying the real-time requirements of a constrained low-priority PROFIBUS profile:

$$\sum_{k=1}^{n}\sum_{i=1}^{nh^k} Ch_i^k + \sum_{k=1}^{n}\left(nlp^k \times Cl^k\right) + t + \max_{k=1..n}\left\{\sum_{i=1}^{nh^k} Ch_i^k\right\} \leq T_{TR} \leq \min_{i=1,\dots,nh^k}\left\{Dh_i^k\right\} + \max_{k=1..n}\left\{\sum_{i=1}^{nh^k} Ch_i^k\right\}, \forall_k \tag{5.30}$$

By further elaboration over (5.30) it also turns out that:

$$\min_{i=1,\dots,nh^k}\left\{Dh_i^k\right\} - \sum_{k=1}^{n}\sum_{i=1}^{nh^k} Ch_i^k - \sum_{k=1}^{n}\left(nlp^k \times Cl^k\right) - t \geq 0,\ \forall_k \tag{5.31}$$

which is obviously equivalent to inequality (5.21)

Fig. 5.6 clarifies the analysis underlying this Section 5.5, exploring the example set given in Table 5.6.

Table 5.6: Example Scenario

Master 1	Master 2	Master 3
$nh^1 = 2$	$nh^2 = 2$	$nh^3 = 2$
$nlp^1 = 3$	$nlp^2 = 1$	$nlp^3 = 4$

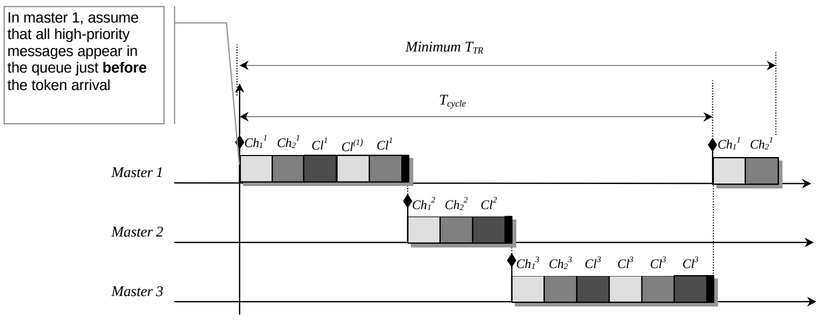

In master 1, assume that all high-priority messages appear in the queue just **before** the token arrival
Minimum T_{TR}
T_{cycle}
Ch_1^1 Ch_2^1 Cl^1 $Cl^{(1)}$ Cl^1
Ch_1^1 Ch_2^1
Ch_1^2 Ch_2^2 Cl^2
Ch_1^3 Ch_2^3 Cl^3 Cl^3 Cl^3 Cl^3
Master 1
Master 2
Master 3

Fig. 5.6 This figure illustrates the upper-bound for the token cycle time

5.5.3. Implementation Issues

We propose two different alternatives to constrain the low-priority traffic at a master k.

1. The first is based on the implementation of a low-priority message counter at the MAC level, intended to control the number of transferred low-priority messages per token visit.

2. The second is based on the application level control of low-priority services, such as application layer non-cyclical low-priority services and remote management services.

The first approach may be implemented as follows. We define at each master *k* the maximum number of low-priority messages to be transferred (nlp^k), per token visit. The low-priority traffic is then controlled by means of a low-priority message counter (*nlp_c*). Fig. 5.7b illustrates such implementation, whereas 5.7a illustrates the traditional PROFIBUS implementation (Fig. 5.7a details the shaded block of Fig. 3.8 in Section 3.4.1).

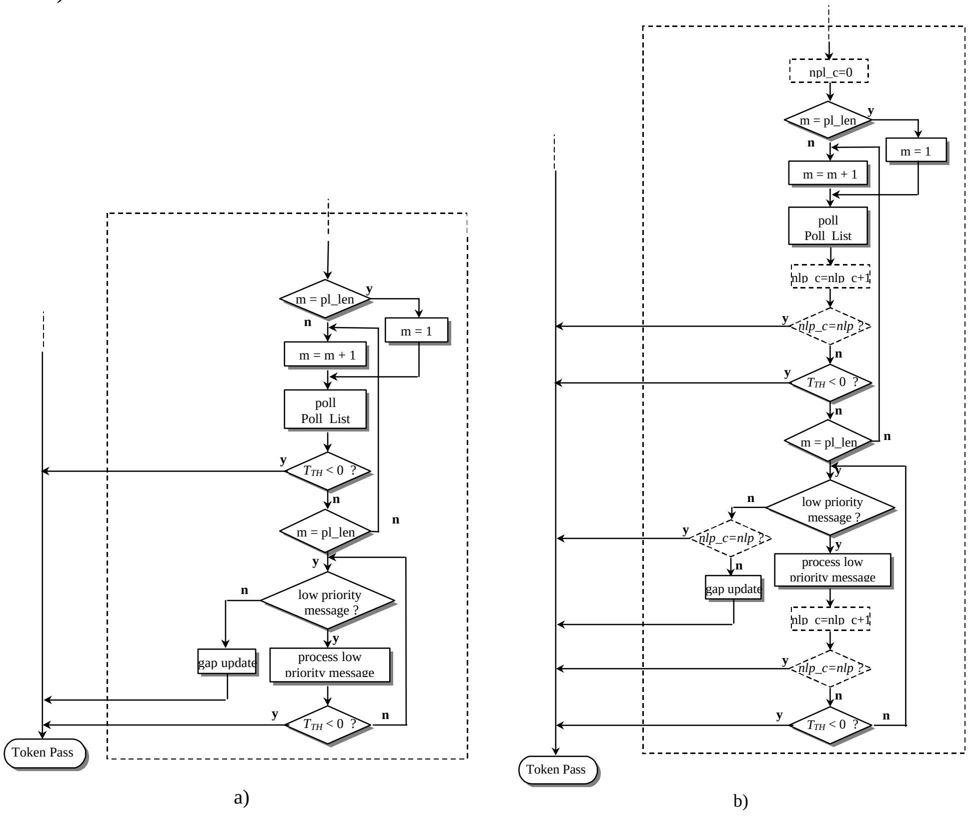


Fig. 5.7 Handling procedures for the non high-priority messages. In a) the traditional implementation and b) the proposed implementation given in the first approach

The second approach is based on the control of low-priority services. Concerning the traffic generated explicitly by the user, it is not advisable to use the Live List management service. The Live List service requests the FDL status of all stations (masters and slaves). It will generate multiple frames in the network. If, in the worst-case, every master station requests a Live List, expression (5.22) would then be as follows:

$$T_{cycle} = \sum_{k=1}^{n} \sum_{i=1}^{nh^k} Ch_i^k + \sum_{k=1}^{n} \left(nlp^k \times Cl^k \right) + t + n \times \sum_{k=1}^{n+s} C_{live} \tag{5.32}$$

where C_{live} stands for a request status message cycle length and ($n+s$) corresponds to the aggregate number of master and slave stations. This would lead to an excessively long T_{cycle}.

Concerning low-priority non-cyclical services, the application process must be able to accept a T_{cycle} parameter, in order to control the number of low-priority messages (nlp^k) generated in master k.

Equation (5.22) must also consider the influence of the Gap updating (see Section 3.4.1 and Fig. 5.7b). We denote C_{gap} as the length of a Gap maintenance message cycle. Therefore, equation (5.33) updates equation (5.22) to include the influence in T_{cycle} resulting from the Gap maintenance, considering that, in the worst-case, each master executes one Gap maintenance message cycle in each token visit.

In order to support Poll List, an additional term must be added to (5.22). The user must be careful while using Poll Lists, since Poll List messages are FDL triggered. Thus, the whole list length must be considered for the evaluation of T_{cycle}:

$$T_{cycle} = \sum_{k=1}^{n} \sum_{i=1}^{nh^k} Ch_i^k + \sum_{k=1}^{n} \left(nlp^k \times Cl^k \right) + t + n \times C_{gap} + \sum_{i=1}^{n} C_{poll}^k \tag{5.33}$$

where C_{poll}^k stands for the master k Poll List length.

5.5.4. Numerical Example

In this profile, as each master must be able to execute all its pending high-priority traffic at each token visit, the token cycle time will be much longer than in the case of the unconstrained low-priority traffic profile. Therefore, as each master will need to wait more time to transmit its high-priority messages, the supported message deadlines are necessarily looser than those supported by the unconstrained low-priority traffic profile.

However, with the constrained low-priority traffic profile, there is more available time transmit low-priority traffic, which allows for an increased non real-time traffic throughput.

Consider a PROFIBUS network with 6 master stations, with timing requirements as shown in Table 5.7.

Table 5.7: Another Example Scenario

	Master 1	**Master 2**	**Master 3**	**Master 4**	**Master 5**	**Master 6**
Dh_1^k	50ms (Dmin)	90ms	120ms	60ms	60ms	80ms
Dh_2^k	100ms	80ms	130ms	200ms	100ms	80ms
Dh_3^k	--------	140ms	110ms	140ms	100ms	100ms
C	2ms	2ms	2ms	2ms	2ms	2ms
nlp^k	3	3	3	3	3	3

For simplification, we assume that the maximum message length is, in all cases, equal to 2ms. Using a 1Mbps network, and if request and response frames total 400 bits, the frame duration is 400μs. Considering 260μs for communication stack and propagation delay, each message cycle will take 660μs. Configuring each master to support up to 2 message replies, we get the 2ms figure for the total length of the message cycle. We also assume that t = 0.1ms.

In order to assess the responsiveness of each proposed profile, the minimum relative deadline of message stream Sh_1^1 will be evaluated.

For the unconstrained low-priority traffic, it follows that (using equation (5.11))

$$T_{del}^{k} = 6 \times 2 = 12\text{ms}, \quad \forall_k$$

In this case, the upper bound for T_{TR} (inequality (5.16)) is imposed by master 4 (or master 5) and is given by:

$$T_{TR_{/4}} \leq \frac{60-2}{3} - 12\text{ms} = 7.33\text{ms}$$

Using inequality (5.15), the minimum the minimum relative deadline for Sh_1^1 is either

$$D\min_{/T_{TR}=7.33} = 2 \times 19.33 + 2 = 40.66\,\text{ms}$$

for T_{TR} = 7.33ms, or

$$D\min_{/T_{TR}=0} = 2 \times 12 + 2 = 26\,\text{ms}$$

for T_{TR} = 0ms (in this case it would not be possible to transmit low-priority messages).

For the constrained low-priority traffic profile, it follows that the lower bound for T_{TR} (inequality 5.25) is:

$$T_{TR} \geq 5 \times 3 \times 2 + 2 \times 2 + 6 \times 3 \times 2 + 0.1 + 6 = 76.1\,\text{ms}$$

and the minimum value for the relative deadlines of high-priority streams is (inequality (5.28)):

$$D\min_{/nlp=3} \geq 76.1 - 6 = 70.1\,\text{ms}$$

This means that the message stream set shown in table 1 is not schedulable using the constrained low-priority profile. In fact, with this profile, the most stringent deadline must be at least 70.1 ms, which is larger than both Dh_1^1, Dh_1^4 and Dh_1^5.

5.6. Summary

PROFIBUS networks aim at the interconnection of field devices such as sensors, actuators and small controllers. Therefore, they may be a privileged basis upon which Distributed Computer-Controlled Systems (DCCS) are built. DCCS impose strict timeliness requirements to the communication network; that is, they impose that traffic must be sent and received within a bounded interval, otherwise a timing fault is said to occur.

The PROFIBUS MAC protocol is based on a token passing procedure used by master stations to grant the bus access to each one of them, and a master-slave procedure used by master stations to communicate with slave stations. The PROFIBUS token passing procedure uses a simplified version of the timed-token protocol (Grow, 1982).

It is possible to support real-time communications with PROFIBUS networks, considering one of the two following approaches (Vasques, 1996; Vasques and Juanole, 1994).

1. If the low-priority traffic is unconstrained, then the real-time traffic requirements may be satisfied, considering that, at least, one pending high-priority message is transmitted per token visit.
2. If the low-priority traffic can be constrained (controlling the number of low-priority message transfers at each master station), then, by an appropriate setting of the T_{TR} parameter, all pending real-time traffic is guaranteed to be transmitted at each token visit.

In this chapter we significantly improve the previous analysis made by Vasques and Juanole.

1. We provide a simple worst-case response time for PROFIBUS messages (equation (5.4)), which reflects the FCFS behaviour the PROFIBUS queues. This new formulation overcomes the problems identified in Chapter 3 for equation (3.9).
2. The evaluation of the worst-case response-time is highly dependent on the accurate definition of the token cycle time. In Section 5.4, we provide an accurate analysis of the PROFIBUS token cycle time (equation (5.14)), which significantly improves the simple and inaccurate result presented in previous analysis (equation (3.6)).
3. Finally, to implement the second approach it is necessary to correctly identify the low-priority traffic supported by PROFIBUS. It is important to note that some of the low-priority traffic cannot be controlled at the user level. In Section 5.5.3 we discuss how to implement such approach.

5.7. References

Grow, R. (1982). A Timed Token Protocol for Local Area Networks. In *Proceedings of Electro'82*, Token Access Protocols, Paper 17/3.

Tovar, E. and Vasques, F. (1998a). Real-Time Fieldbus Communications Using PROFIBUS Networks. Polytechnic Institute of Porto, Technical Report HURRAY-TR-9803, April 1998, to appear in *IEEE Transactions on Industrial Electronics*.

Tovar, E. and Vasques, F. (1998b). Guaranteeing Real-Time Message Deadlines in PROFIBUS Networks. In *Proceedings of the 10th Euromicro Workshop on Real-Time Systems*, pp. 79-86, Published by IEEE Computer Society Press.

Tovar, E. and Vasques, F. (1998c). Cycle Time Properties of the PROFIBUS Timed Token Protocol. Polytechnic Institute of Porto, Technical Report HURRAY-TR-9811, August 1998, to appear in *Computer Communications*, Elsevier Science.

Tovar, E. and Vasques, F. (1998d). Setting Target Rotation Time in PROFIBUS Based Real-Time Distributed Applications. In *Proceedings of the 15th IFAC Workshop on Distributed Computer Control Systems*, pp. 1-6, Published by Pergamon, an Imprint of Elsevier Science.

Vasques, F. and Juanole, G. (1994). Pre-run-time Schedulability Analysis in Fieldbus Networks. In *Proceedings of the Annual Conference of the IEEE Industrial Electronics Society*, pp. 1200-1204.

Vasques, F. (1996). Sur l'Integration de Mecanismes d'Ordonnacement et de Communication dans la Sous-couche MAC de Reseaux Locaux Temps-reel. PhD Thesis (*in French*), available as Technical Report LAAS No. 96229.

Chapter 6

Real-Time Communications with WorldFIP Networks: Contributions to the State-of-the-Art

In this chapter we develop methodologies to guarantee the real-time behaviour of WorldFIP periodic and aperiodic buffer transfers. This chapter is largely drawn from the following published work: "Distributed Computing for the Factory-Floor: A Real-Time Approach using WorldFIP Networks" (Tovar and Vasques, 1999a); "Factory Communications: on the Configuration of the WorldFIP Bus Arbitrator Table" (Tovar and Vasques, 1999b); "Contributions for the Worst-Case Response-Time Analysis of Sporadic Traffic in WorldFIP Networks" (Tovar and Vasques, 1999c).

6.1. Introduction

The WorldFIP protocol is based on a centralised medium access control mechanism, where a specific station, the bus arbitrator (BA), controls all data transfers between the different stations. At configuration time, the BA is given a list of process variables to scan along with their corresponding periods. This piece of information is known as the bus arbitrator table.

WorldFIP supports two basic types of transmission services: exchanges of identified variables and exchanges of messages. In this work we address WorldFIP networks supporting only exchanges of identified variables, since they are the basis of WorldFIP real-time services. As it was explained in Section 3.5.1, in WorldFIP identified variables may be exchanged in a periodic or aperiodic basis. While for the periodic traffic, end-to-end communication deadlines can be easily guaranteed, since for them the WorldFIP BAT (Bus Arbitrator Table) implements a pre-defined static schedule, for the aperiodic traffic more complex analysis must be performed.

The remainder of this chapter is organised as follows. In Section 6.2 we introduce the network and message models, which will be used throughout the rest of the chapter. In Section 6.3 we introduce the basic HCF/LCM methodology for building the bus arbitrator table. In Section 6.4 and Section 6.5 we describe detailed algorithms for obtaining the BAT schedule, based on RM and EDF approaches, respectively. Moreover, in these two sections we provide feasibility tests, adapted from previous results on the schedulability analysis. Sections 6.2 up to 6.6 form the ground basis upon we build (in Section 6.6) a worst-case response time analysis for the WorldFIP aperiodic buffer transfers. This integrated methodology is the more important since the analysis for the aperiodic traffic depends for the most part on the schedule for the periodic traffic.

6.2. Network and Buffer Models

Consider a bus network with *n* stations. One of these *n* stations performs the role of active bus arbitrator (BA), while the others are simple producer/consumer stations.

Associated to this set of stations there is a set of *np* periodic buffer transfer streams (Sp_i). Each buffer transfer stream is a temporal sequence of periodic buffer transfer concerning variables with the same identifier. Contrarily to the case of P-NET (Section 4.2) and PROFIBUS (Section 5.2), these streams are not related with any particular station. Sp_i streams are characterised as follows:

$$Sp_i = (Cp_i, Tp_i, Dp_i) \qquad \textbf{(6.1)}$$

with $i = 1, ..., np$. In the rest of this chapter, we will refer to Sp_i streams as periodic streams. Cp_i represents the maximum amount of time to perform a periodic buffer transfer, and its value is given by (refer to Fig. 3.12 and Fig. 3.13, in Section 3.5.1):

$$Cp_i = \frac{len(\mathrm{id\_dat\_Sp_i}) + len(\mathrm{rp\_dat\_Sp_i})}{bps} + 2 \times t_r \qquad \textbf{(6.2)}$$

where *bps* stands for the network data rate (in bits per second) and *len*(`<frame>`) is the length, in bits, of frame `<frame>`. Tp_i is the required periodicity for the buffer transfer, and its value is a multiple of the microcycle value. For simplification, throughout this chapter we assume that all periods are multiples of 1ms. Dp_i is the deadline of the periodic stream, which we assume that can be equal to its period; that is, a buffer transfer concerning a periodic variable can have a communication jitter of up to (Tp_i-Cp_i), and this conforms with the requirements of the distributed application (Fig. 6.1).

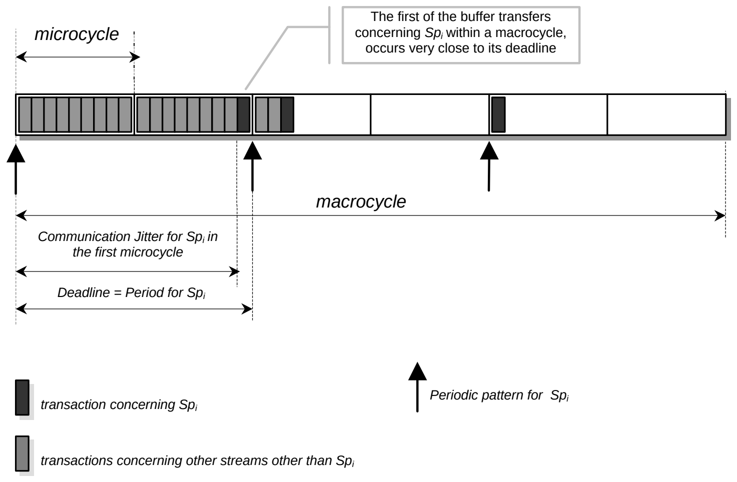

microcycle
The first of the buffer transfers concerning Sp_i within a macrocycle, occurs very close to its deadline
macrocycle
Communication Jitter for Sp_i in the first microcycle
Deadline = Period for Sp_i
transaction concerning Sp_i
Periodic pattern for Sp_i
transactions concerning other streams other than Sp_i

Fig. 6.2 Illustration of deadline and period in periodic streams (equation (6.1))

Additionally, we consider a set of aperiodic buffer transfer streams (Sa_i^k) associated with each station k (k = 1, ..., n):

$$Sa_i^k = \left(Ca_i^k, Ta_i^k, Da_i^k\right) \tag{6.3}$$

Similarly to the case of periodic streams, in the rest of this chapter we will refer to Sa_i^k streams as aperiodic streams. If na^k is the number of aperiodic streams requested at a station k, na denotes the sum of all na^k in the overall network.

Note (see Section 3.5.1) that a station k can only have aperiodic streams if it produces a variable related with a periodic stream. It is also important to note that two different stations can have aperiodic streams related to a same variable identifier.

Ca_i^k will later be addressed in more detail in Section 6.6. Ta_i^k is the minimum time interval between any two consecutive requests for Sa_i^k being placed in the queue for aperiodic requests of the requesting station (refer to Fig. 3.15 in Section 3.5.1). The maximum admissible time interval between the time instant when the request is placed in the local queue (requesting station) and the completion of the transfer of the identified variable is denoted as Da_i^k.

6.3. Using the HCF/LCM Methodology for Setting the BAT

In WorldFIP networks, the bus arbitrator table (BAT) imposes the schedule of periodic buffer transfers, and also regulates the aperiodic buffer transfers.

Following the HCF/LCM methodology to build the BAT (refer to Section 3.5.1), the value of the microcycle (μCy) must be chosen as:

$$\mu Cy = \underset{i=1,..,np}{HCF}\left(Tp_i\right) \tag{6.4}$$

where HCF stands for the highest common factor and corresponds to the following value:

$$\mu Cy = \max\{\Omega\} \wedge \Omega \in \aleph, \ \text{ with } \frac{Tp_i}{\Omega} = \left\lfloor \frac{Tp_i}{\Omega} \right\rfloor, \forall_i \tag{6.5}$$

In Appendix C.1, we give the pseudo-code details of an algorithm for the evaluation of the microcycle.

The macrocycle (MCy) is defined as:

$$MCy = N \times \mu Cy \tag{6.6}$$

where N is the number of microcycles that compose a macrocycle. Using the LCM (Least Common Multiple) rule, N can be evaluated as follows:

$$N = \min\{\Phi\} \wedge \Phi \in \aleph, \ \text{ with } \frac{\Phi}{Tp_i / \mu Cy} = \left\lfloor \frac{\Phi}{Tp_i / \mu Cy} \right\rfloor, \forall_i \tag{6.7}$$

In Appendix C.2, we give the pseudo-code details of an algorithm for the evaluation of the macrocycle is suggested.

6.4. Setting the WorldFIP BAT: a Rate Monotonic Approach

6.4.1. Algorithmic Approach for Building the BAT

After defining the values for the microcycle and macrocycle, a schedule can easily be built according to the rate monotonic (RM) algorithm (Section 2.2.3) as follows:

1. From variable with the shortest period until variable with the longest period
    - 1.1. If the current load in a microcycle added to Cp_i is still smaller than the value of the microcycle, then schedule Sp_i for each one of the microcycles (of a macrocycle) corresponding to the periodic pattern of Sp_i. Update the value of the load in each concerned microcycle.
    - 1.2. If the load in some of the microcycles does not allow to schedule a scan for that stream, try to schedule it in the next microcycles up to the microcycle in which a new scan for Sp_i would be required. If this is not possible, the stream set is not schedulable.

Assume an example where all variables have a data field with 4 bytes (all *RP_DAT* have 92 bits), tr = 20ms and the network data rate is 2.5Mbps. Then, the duration of any elementary transaction will be (64+80)/2.5+2×20=97.6μs (equation (6.2)). Given the example of Table 3.3 (with Cp_i = 0.0976ms, $\forall_i$) and considering the RM algorithm, the BAT will result as shown in Table 6.1.

Table 6.1: BAT (using RM) for Example of Table 3.3

	Microcycle											
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
bat[*A*,*cycle*]	1	1	1	1	1	1	1	1	1	1	1	1
bat[*B*,*cycle*]	1	0	1	0	1	0	1	0	1	0	1	0
bat[*C*,*cycle*]	1	0	0	1	0	0	1	0	0	1	0	0
bat[*D*,*cycle*]	1	0	0	0	1	0	0	0	1	0	0	0
bat[*E*,*cycle*]	1	0	0	0	1	0	0	0	1	0	0	0
bat[*F*,*cycle*]	1	0	0	0	0	0	1	0	0	0	0	0

where *bat*[*i*, *j*] is a table of booleans with *i* ranging from 1 up to *np*, and *j* ranging from 1 up to *N* (number of microcycles in a macrocycle).

In Appendix C.3, a detailed algorithm for building the BAT using the RM algorithm is presented. The algorithm indicates whether or not all traffic is schedulable (line 22). In the algorithm, the vector *load*[ ] is used to store the load in each microcycle as the traffic is scheduled. It also assumes that the array *Vp*[,] is ordered from the variable with the smallest period (*Vp*[1,]) to the variable with the longest period (*Vp*[*np*,]).

The HCF/LCM/RM approach for building a WorldFIP BAT has the following characteristic: the variables are not scanned at exactly regular intervals. For the given example, only variables *A* and *B* are scanned exactly in the same "slot" within the microcycle. All other variables suffer from a slight communication jitter. For instance, concerning variable *F*, the interval between microcycles 1 and 7 is (1-5×0.0976)+5+(3×0.0976)=5.8048ms, whereas the interval between microcycles 7 and 13 is (1-3×0.0976)+5+(5×0.0976)=6.1952ms.

Note that by using the RM algorithm some of the variables with larger periods can be scheduled for the next microcycles, thus inducing an increased communication jitter for those variables. For example, if the network data rate is 1Mbps instead of 2.5Mbps (Cp_i=(64+80)/1+2×20=184μs), the BAT would be as shown in Table 6.2, since a microcycle is only able to schedule up to 5 periodic buffer transfers (Fig 6.2).

Table 6.2: BAT (using RM) for Modified Example of Table 3.3

	Microcycle											
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
bat[*A*,*cycle*]	1	1	1	1	1	1	1	1	1	1	1	1
bat[*B*,*cycle*]	1	0	1	0	1	0	1	0	1	0	1	0
bat[*C*,*cycle*]	1	0	0	1	0	0	1	0	0	1	0	0
bat[*D*,*cycle*]	1	0	0	0	1	0	0	0	1	0	0	0
bat[*E*,*cycle*]	1	0	0	0	1	0	0	0	1	0	0	0
bat[*F*,*cycle*]	0	1	0	0	0	0	1	0	0	0	0	0

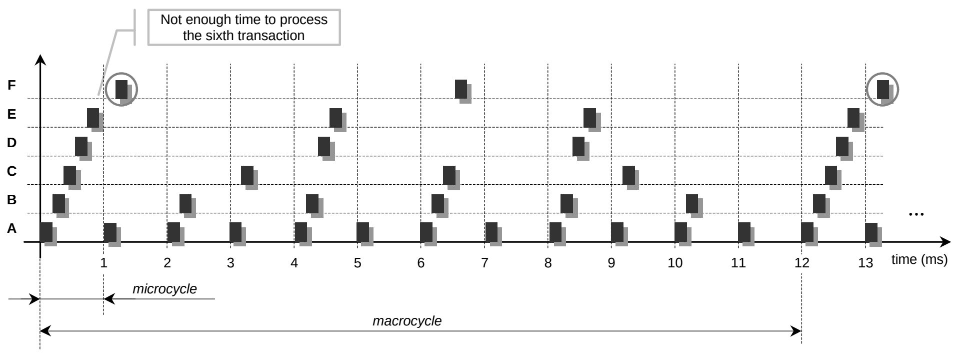


Fig. 6.2 Schedule (RM Approach) for the set example of Table 3.3

6.4.2. A Feasibility Test Based on the Response Time Techniques

Although the *rm_bat* algorithm indicates whether all traffic is schedulable (line 22), a simple pre-run-time schedulability test can be provided to check the schedulability of the periodic stream set. The following analysis adapts to the WorldFIP case, the worst-case response time analysis of tasks in a single processor environment presented in Section 2.4.3.

For simplicity of analysis we consider that all transactions have a length of *Cp*, with $Cp = \max\{Cp_i\}, \forall_i$. In most of the cases this is a valid assumption, since the data field for the periodic streams will concern process data which typically has a number of bytes ranging from 2 to 4.

The interference that a periodic stream Sp_i suffers in NR_i microcycles is given by:

$$I_i = \sum_{j \in hp(i)} \left\lceil \frac{NR_i \times \mu Cy}{Tp_j} \right\rceil \qquad \textbf{(6.8)}$$

where I_i corresponds to the number of "requests" for higher-priority periodic streams. Therefore, the number of requests to process during NR_i microcycles (including the request for Sp_i) is:

$$1 + I_i = 1 + \sum_{j \in hp(i)} \left\lceil \frac{NR_i \times \mu Cy}{Tp_j} \right\rceil \tag{6.9}$$

The maximum number of buffer exchanges that fit in a microcycle is given by:

$$\left\lfloor \frac{\mu Cy}{Cp} \right\rfloor \tag{6.10}$$

which is a constant. Therefore, the number of microcycles (NR_i) needed to process the request for Sp_i is given by:

$$NR_i = \min\{\Psi\} \wedge \Psi \in \aleph \wedge \Psi \le \frac{Tp_i}{\mu Cy}, \quad \text{with} \quad 1 + \sum_{j \in hp(i)} \left\lceil \frac{\Psi \times \mu Cy}{Tp_j} \right\rceil \le \Psi \times \left\lfloor \frac{\mu Cy}{Cp} \right\rfloor \tag{6.11}$$

In (6.11), inequality $1 + I_i \le Y \times \lfloor \mu Cy / Cp \rfloor$ is tested in successive iterations, starting with $Y = 1$. If the solution (if any) gives $Y > Tp_i / \mu Cy$ then the periodic stream Sp_i is not schedulable.

6.4.3. Numerical Example

Consider a WorldFIP network, with the following set of periodic streams.

Table 6.3: Set of Periodic Streams (Cp = 0.21ms)

Identifier	**A**	**B**	**C**	**D**	**E**
Periodicity (ms)	1	1	1	1	3

Stream Sp_E is RM schedulable if the related feasibility test (6.11) holds:

$\Psi = 1$:

$$1 + \left\lceil \frac{1}{1} \right\rceil + \left\lceil \frac{1}{1} \right\rceil + \left\lceil \frac{1}{1} \right\rceil + \left\lceil \frac{1}{1} \right\rceil = 5 \le 1 \times 4, \qquad \text{FALSE,}$$

and thus Sp_E is not schedulable in the first microcycle,

$\Psi = 2$:

$$1 + \left\lceil \frac{2}{1} \right\rceil + \left\lceil \frac{2}{1} \right\rceil + \left\lceil \frac{2}{1} \right\rceil + \left\lceil \frac{2}{1} \right\rceil = 9 \le 2 \times 4, \qquad \text{FALSE,}$$

and thus Sp_E is not schedulable in the second microcycle,

$\Psi = 3$:

$$1 + \left\lceil \frac{3}{1} \right\rceil + \left\lceil \frac{3}{1} \right\rceil + \left\lceil \frac{3}{1} \right\rceil + \left\lceil \frac{3}{1} \right\rceil = 13 \le 3 \times 4, \qquad \text{FALSE,}$$

which means that Sp_E is not schedulable at all, as Y must be smaller or equal than $Tp_E / \mu Cy = 3$.

6.5. Setting the WorldFIP BAT: a Earliest Deadline Approach

6.5.1. Algorithmic Approach for Building the BAT

An alternative approach for the BAT schedule can be the earliest deadline first (EDF) approach (Section 2.2.3). With the EDF approach, periodic streams are scheduled according to the earliest deadline (the microcycle when a new "request" appears). If several streams have the same deadline, priority is given to the periodic stream with the earliest request.

In Appendix C.4 we give a detailed description of an algorithm that can be used for building the BAT using the EDF approach. In that algorithm, the array *disp*[,] is used to store in *disp*[*i*,1] if there is a pending request for variable *i*, in *disp*[*i*,2] the deadline (multiple of the microcycle) and in *disp*[*i*,3] the microcycle at which the request is made. Note that for algorithmic convenience, the first requests (for all the variables) appear in *cycle* = 0, and from those, some will be scheduled in the first microcycle (*cycle* + 1).

The advantage of the EDF approach over the DM approach can be easily depicted from the following example. Consider the periodic stream set example shown in Table 6.4. In this example, the macrocycle is made of 6 microcycles, and the maximum number of buffer exchanges that fit in a microcycle is $\lfloor 1/0.3 \rfloor = 3$

Table 6.4: Example Set of Periodic Streams (Cp = 0.30ms)

Identifier	**A**	**B**	**C**	**D**	**E**	**F**
Periodicity (ms)	1	2	2	3	3	3

Considering the RM approach, the first request for Sp_F would miss a deadline (Table 6.5).

Table 6.5: BAT (using RM) for Example of Table 6.4

	Microcycle					
	1	**2**	**3**	**4**	**5**	**6**
bat[*A*,*cycle*]	1	1	1	1	1	1
bat[*B*,*cycle*]	1	0	1	0	1	0
bat[*C*,*cycle*]	1	0	1	0	1	0
bat[*D*,*cycle*]	0	1	0	1	0	0
bat[*E*,*cycle*]	0	1	0	1	0	0
bat[*F*,*cycle*]	**X**	**X**	**X**	0	0	1

In fact, considering the RM approach, there is no empty slot for Sp_F in the first 3 microcycles. For the same periodic stream set (Table 6.4), considering the EDF approach, the BAT schedule would be as shown in Table 6.6.

Table 6.6: BAT (using EDF) for Example of Table 6.4

	Microcycle					
	1	**2**	**3**	**4**	**5**	**6**
bat[*A*,*cycle*]	1	1	1	1	1	1
bat[*B*,*cycle*]	1	0	1	0	1	0
bat[*C*,*cycle*]	1	0	0	1	1	0
bat[*D*,*cycle*]	0	1	0	1	0	0
bat[*E*,*cycle*]	0	1	0	0	0	1
bat[*F*,*cycle*]	0	0	1	0	0	1

As it can be seen, with the EDF approach this stream set is now schedulable, which was not possible with the RM approach. However, with the EDF approach the communication jitter is increased. This results from the following: some of the buffer transfers with more stringent relative deadlines are occasionally "delayed" by buffer transfers with less stringent relative deadlines. For instance, variable *C* is scheduled for the 4th microcycle, whereas with the RM approach it would be scheduled for the 3rd.

6.5.2. A Feasibility Test Based on the Response Time Techniques

Similarly to the RM case, a simple pre-run-time schedulability test can be derived to check the schedulability of the periodic stream set, also based on the classic task's response time analysis (Section 2.5.4). Note that the *edf_bat* algorithm (Appendix C.4) gives (line 10) whether the periodic traffic is schedulable or not. For the simplification of the analysis we consider that all transactions have a length of Cp, with $Cp = \max\{Cp_i\}$, $\forall_i$.

The interference that a periodic stream Sp_i suffers in NR_i microcycles is given by:

$$I_i = \sum_{\substack{j \neq i \\ D_j \leq D_i}} \left(\min\left\{ 1 + \left\lfloor \frac{NR_i \times \mu Cy}{Tp_j} \right\rfloor, 1 + \left\lfloor \frac{D_i - D_j}{Tp_j} \right\rfloor \right\} \right) \tag{6.12}$$

This expression considers that, although in the $NR_i \times \mu Cy$ time interval there is a number of requests for streams that have relative deadlines smaller than the one for Sp_i, they can only be considered if they have absolute deadlines which are smaller than the deadline for stream Sp_i.

Therefore, the number of requests to process during NR_i microcycles (including the request for periodic stream Sp_i) is:

$$1 + \sum_{\substack{j \neq i \\ D_j \leq D_i}} \left(\min\left\{ 1 + \left\lfloor \frac{NR_i \times \mu Cy}{Tp_j} \right\rfloor, 1 + \left\lfloor \frac{D_i - D_j}{Tp_j} \right\rfloor \right\} \right) \tag{6.13}$$

The number of microcycles (NR_i) needed to process the request for periodic stream Sp_i is then given by:

$$NR_i = \min\{\Psi\} \wedge \Psi \in \aleph \wedge \Psi \leq \frac{Tp_i}{\mu Cy},$$

$$\text{with} \quad 1 + \sum_{\substack{j \neq i \\ D_j \leq D_i}} \left(\min\left\{ 1 + \left\lfloor \frac{\Psi \times \mu Cy}{Tp_j} \right\rfloor, 1 + \left\lfloor \frac{D_i - D_j}{Tp_j} \right\rfloor \right\} \right) \leq \Psi \times \left\lfloor \frac{\mu Cy}{Cp} \right\rfloor \qquad \textbf{(6.14)}$$

In (6.14), inequality $1 + I_i \leq Y \times \lfloor \mu Cy/Cp \rfloor$ is tested in successive iterations, starting with $Y = 1$. If the solution (if any) gives $Y > Tp_i / \mu Cy$, then Vp_i is not schedulable.

6.5.3. Numerical Example

Consider a WorldFIP network with a set of periodic streams as shown in Table 6.4, where the maximum number of buffer exchanges that fit in a microcycle (6.10) is $\lfloor 1 / 0.3 \rfloor = 3$. The periodic stream Sp_F will be schedulable by the EDF algorithm if the related feasibility test (6.14) holds.

$\Psi = 1$:

$1 + (\min\{2,3\} + \min\{1,1\} + \min\{1,1\} + \min\{1,1\} + \min\{1,1\}) = 7 \leq 1 \times 3,$ FALSE,

and thus Sp_F is not schedulable in the first microcycle,

$\Psi = 2$:

$1 + (\min\{3,3\} + \min\{2,1\} + \min\{1,1\} + \min\{1,1\} + \min\{1,1\}) = 8 \leq 2 \times 3,$ FALSE,

and thus Sp_F is not schedulable in the second microcycle,

$\Psi = 3$:

$1 + (\min\{4,3\} + \min\{2,1\} + \min\{1,1\} + \min\{1,1\} + \min\{1,1\}) = 8 \leq 3 \times 3,$ TRUE,

which means that a buffer exchange for Sp_F is schedulable in the third microcycle.

6.6. Worst-Case Response Time for the Aperiodic Traffic

Concerning the aperiodic streams, we consider that they are all generated by the use of the application layer service `L_FREE_UPDATE.req(ID_Ap, urgent)`, thus we only consider the urgent queues (both at the requesting station and at the BA) (refer to Fig. 3.15 for clarification).

It is important to stress that the urgent queue in the BA is only processed if, and only if, the BA's ongoing aperiodic queue is empty, as detailed in Fig 6.3.

As illustrated in Fig. 6.3, traffic concerning the aperiodic buffer transfers (transactions `ID_RQ` / `RP_RQ` or `ID_DAT` /`RP_DAT`) can only be processed if there is still enough time in a specific microcycle to completely process each one of them. That is, they are atomically processed.

Therefore, we define the worst-case response time (Ra_i^k) for an aperiodic transfer Sa_i^k as the time interval between the arrival of the `L_FREE_UPDATE.req(ID_Ap,` `urgent)` to the urgent queue of station *k* (instant t_0) and the completion of the buffer transfer concerning the aperiodic stream Sa_i^k.

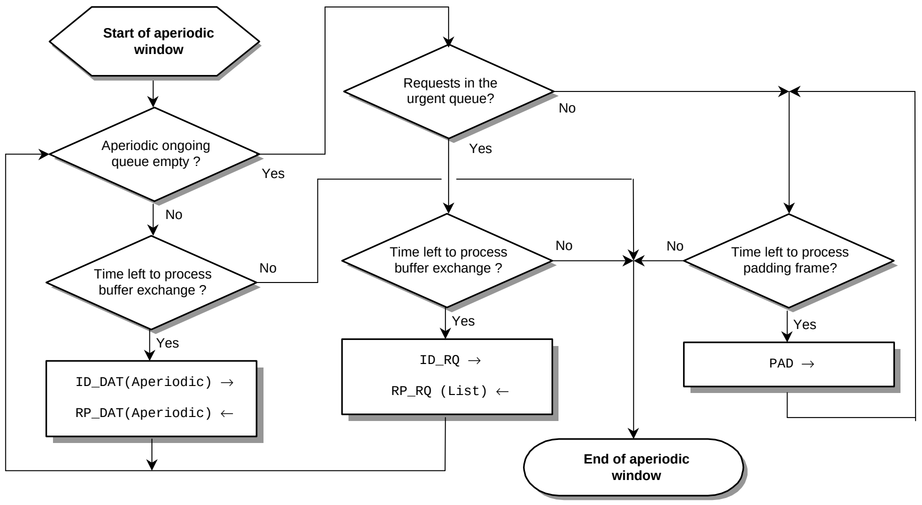

Start of aperiodic window
Aperiodic ongoing queue empty ?
Yes
No
Time left to process buffer exchange ?
No
Yes
ID_DAT(Aperiodic) →
RP_DAT(Aperiodic) ←
Requests in the urgent queue?
No
Yes
Time left to process buffer exchange ?
No
Yes
ID_RQ →
RP_RQ (List) ←
No
Time left to process padding frame?
Yes
PAD →
End of aperiodic window

Fig. 6.3 Sequence for aperiodic transfers

The response time associated to an aperiodic stream includes the following three components (Fig. 6.4):

1. the time elapsed between t_0 and the time instant when the requesting station is able to indicate the BA (via *RP_DAT*, with the request bit set) that there is a pending aperiodic request. We define this time interval as the *dead interval* of a producer station;
2. the time interval during which the request indication stays in the BA's urgent queue until the related *ID_RQ* / *RP_RQ* pair of frames is processed in an aperiodic window;
3. the time interval during which the Sa_i^k stays in the BA's ongoing aperiodic queue until the related *ID_DAT_Ap* / *RP_DAT* pair of frames is processed in an aperiodic window.

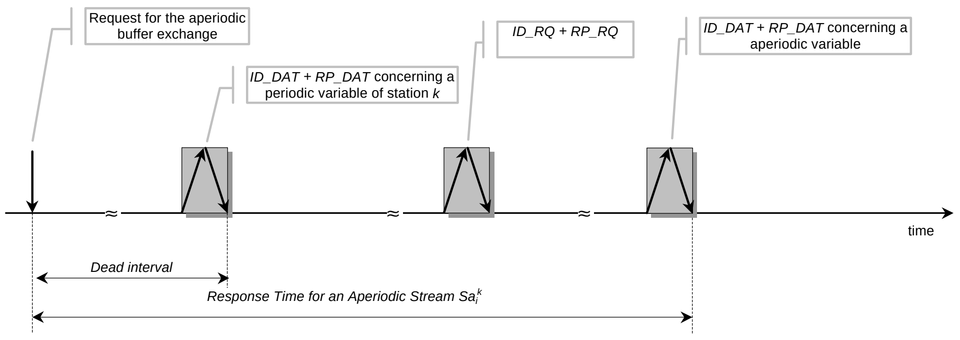

Request for the aperiodic buffer exchange
ID_DAT + RP_DAT concerning a periodic variable of station k
ID_RQ + RP_RQ
ID_DAT + RP_DAT concerning a aperiodic variable
time
Dead interval
Response Time for an Aperiodic Stream Sa_i^k

Fig. 6.4 Timings for the transactions associated with the processing of an aperiodic variable

6.6.1. Upper Bound for the Dead Interval

The upper bound for the dead interval in a station *k* is related to the smallest scanning rate of a produced periodic variable in that station. It is important to note that a periodic stream (Sp_i) is not polled at regular intervals, since there is a communication jitter inherent to the BAT setting. Therefore, the previous result for the evaluation of the dead interval (refer to Section 3.5.2) in a station *k* must be updated to:

$$s^k = Tp_j + J_{Sp_j} + Cp_j, \text{ with } Sp_j : Tp_j = \min_{Sp_i \text{ produced in } k} \{Tp_i\} \qquad \textbf{(6.15)}$$

where J_{Sp_j} is the maximum communication jitter of a periodic stream Sp_j. For example, considering the periodic stream set of Table 3.3 (Section 3.5.1), if variable *F* is the only produced periodic variable at station *k*, then s^k=6+0.1952+0.0976= 6.2928ms (see Section 6.4.1 for the evaluation of the communication jitter for variable *F*).

For the evaluation of the dead interval (6.15) it is also considered that a local aperiodic request is only processed (setting the request bit in the *RP_DAT* frame) if it arrives before the start of the related *ID_DAT*. Hence, the term Cp_j is included in equation (6.15).

In Appendix C.5 we describe a detailed algorithm for the evaluation of the communication jitter associated to a periodic variable. This algorithm is the basis for the evaluation of the dead interval in a specific *k* station.

Considering again the periodic stream set of Table 3.3, the value for the communication jitter associated to each periodic variable is as shown in Table 6.7.

Table 6.7: Communication Jitter for the Periodic Stream Set of Table 3.3 (Cp_i = 0.21ms)

Identifier	**A**	**B**	**C**	**D**	**E**	**F**
Communication Jitter (ms)	0	0	0.21	0.21	0.58	0.79

Thus, the periodic traffic schedule, according to the RM algorithm would be as follows:

Table 6.8: BAT (using RM) for Example of Table 3.3 (Cp_i = 0.21ms)

	Microcycle											
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
bat[*A*,*cycle*]	1	1	1	1	1	1	1	1	1	1	1	1
bat[*B*,*cycle*]	1	0	1	0	1	0	1	0	1	0	1	0
bat[*C*,*cycle*]	1	0	0	1	0	0	1	0	0	1	0	0
bat[*D*,*cycle*]	1	0	0	0	1	0	0	0	1	0	0	0
bat[*E*,*cycle*]	0	1	0	0	1	0	0	0	1	0	0	0
bat[*F*,*cycle*]	0	1	0	0	0	0	1	0	0	0	0	0

Note that considering all Cp_i = 0.21ms, only 4 periodic variables can be accommodated in each microcycle (1ms), and thus the jitter associated to the transfer of the lower priority variables is not negligible.

6.6.2. Aperiodic Busy Interval

The worst-case response time for an aperiodic variable transfer occurs if, when the request arrives at the BA's urgent queue (s^k after t_0), this queue is already full of requests from all the other aperiodic streams in the network.

We consider that:

1. for each aperiodic stream, a request for identification must be made, and thus the network load is maximised. In practice this case only occurs if each station has only one aperiodic stream, since the BA is able to manage the urgent queue to avoid redundancy in *ID_RQs* addressed to a same station. This is the case which generates the heavier network load, hence which leads to the worst-case response time for aperiodic streams;
2. and that those aperiodic stream requests will start to contend for the medium access when the BA is starting the macrocycle (with the RM scheduling policy used to construct the BAT, this means maximum periodic load). This is defined as the critical instant.

We also consider that all aperiodic traffic has a minimum inter-arrival time between requests, which is greater than its worst-case response time. Therefore, no other aperiodic request appears before the completion of a previous one. Hence, the maximum number of aperiodic requests pending in the BA is *na*, with *na* being the number of aperiodic requests (two different station can require an aperiodic buffer transfer of a same variable) that can be made in the network.

We define the time interval between the critical instant and the time instant when all the aperiodic requests (which were pending at the critical instant) have been processed as the aperiodic busy interval (ABI), since all aperiodic windows within the microcycles are used to process aperiodic traffic.

It is also clear that to process all those *na* requests, the aperiodic windows will perform alternately sequences of (*ID_RQ* / *RP_RQ*) and (*ID_DAT* / *RP_DAT*), as the BA gives priority to the ongoing aperiodic queue (refer to Fig 6.3).

If all the aperiodic variables have a similar length, Ca^* may be defined as:

$$Ca^* = \max_{i=1,..,na}\left\{\frac{len(\text{id\_dat}) + len(\text{rp\_dat}_\text{i})}{bps} + 2\times t_r, \frac{len(\text{id\_rq}) + len(\text{rp\_rq})}{bps} + 2\times t_r\right\} \quad \textbf{(6.16)}$$

which gives the longest transaction in an aperiodic window.

Therefore, the maximum number of transactions to be processed during the ABI is $2\times na$, corresponding to the set of *ID_RQ* / *RP_RQ* transactions and the set of *ID_DAT* / *RP_DAT* transactions.

Considering these assumptions, the analysis for the worst-case response time for the aperiodic traffic is as follows.

6.6.3. Worst-Case Response Time for Aperiodic Buffer Transfers

The worst-case response time of aperiodic buffer transfers is a function of the network periodic load during the ABI, since it bounds the length of the available aperiodic windows.

The length of the aperiodic window in the l^{th} cycle (l = 1, .., N, N + 1, ..) may be evaluated as follows:

$$aw(l^*) = \mu Cy - \sum_{i=1}^{np} \left(bat[i, l^*] \times Cp_i \right) \qquad \textbf{(6.17)}$$

where $bat[i,l^*]$ is a matrix representing the schedule of the periodic traffic and $l^* = [(l\text{-}1) \bmod N] + 1$. Therefore, the number of aperiodic transactions that fit in the l^{th} aperiodic window is:

$$nap(l) = \left\lfloor \frac{aw(l^*)}{Ca^*} \right\rfloor \qquad \textbf{(6.18)}$$

where Ca^* is as given by (6.16). It follows that the number of microcycles (N') in an ABI is:

$$N' = \min\{\Psi\}, \text{ with } \Psi = \sum_{l=1}^{N'} nap(l^*) \wedge \Psi \geq 2 \times na \qquad \textbf{(6.19)}$$

that is, the minimum number of microcycles within which the number of available "slots" (each "slot" with the length of Ca^*) is at least $2 \times na$.

In Appendix C.6 we describe a detailed algorithm for the evaluation of the number of microcycles in the aperiodic busy interval.

Knowing the number of microcycle in a ABI (N'), the length of the aperiodic busy interval (*len_abi*) may be evaluated as follows:

$$len\_abi = (N'-1) \times \mu Cy + \sum_{i=1}^{np} \left(bat[i, N^*] \times Cp_i \right) + \left(2 \times na - \sum_{l=1}^{N'-1} nap(l^*) \right) \times Ca^* \qquad \textbf{(6.20)}$$

where $\sum_{i=1,..,np}(bat[i,N^*] \times Cp_i)$ gives the length of the periodic window in microcycle N', with $N^* = [(N'\text{-}1) \bmod N] + 1$ and $(2 \times na - \sum_{l=1,..,N'-1} nap(l^*)) \times Ca^*$ gives the length of the aperiodic window, included in the aperiodic busy interval, also in microcycle N'.

In appendix C.7 a detailed algorithm for evaluating the length of the ABI is provided.

Therefore, the worst-case response time for an aperiodic buffer transfer requested at station k is:

$$Ra^k = s^k + len\_abi \qquad \textbf{(6.21)}$$

and thus, the minimum inter-arrival time between any two consecutive aperiodic requests of the same aperiodic variable in a station k is:

$$Ta_i^k \geq Ra^k = s^k + len\_abi \qquad \textbf{(6.22)}$$

6.6.4. Numerical Example

Consider a WorldFIP network with a set of periodic streams as shown in Table 3.3. Assume that this system must also support 9 aperiodic buffer exchanges. Assume also that the Cp_i=Cp=0.0976ms, $\forall_i$, and that for aperiodic traffic Ca^* = 0.1ms.

If the BAT is built as shown in Table 6.1, and the urgent aperiodic request queue is full at the critical instant, a schedule for the aperiodic traffic is as represented in Fig. 6.5. Obviously, as the BA urgent queue is a FCFS queue, the order of the aperiodic transfers is arbitrary.

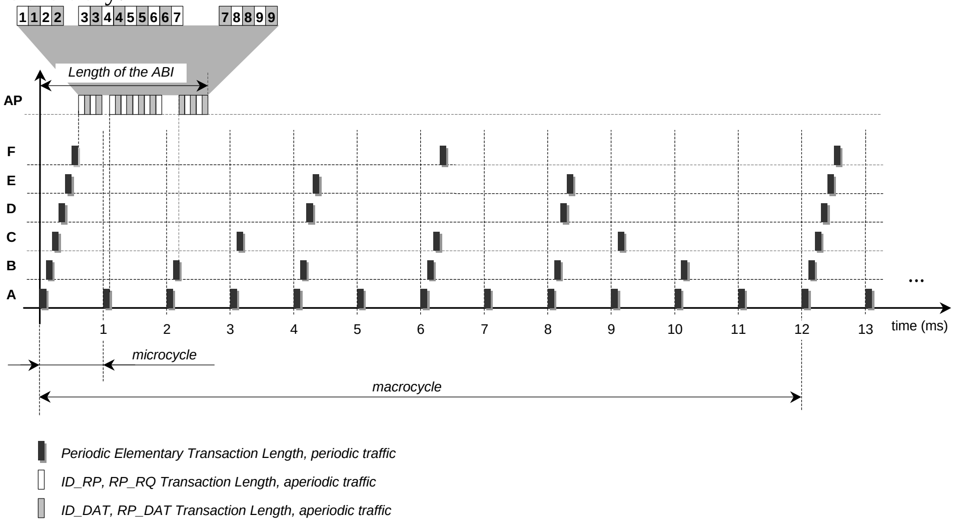

1 1 2 2 3 3 4 4 5 5 6 6 7 7 8 8 9 9
Length of the ABI
AP
F
E
D
C
B
A
...
1 2 3 4 5 6 7 8 9 10 11 12 13
time (ms)
microcycle
macrocycle
Periodic Elementary Transaction Length, periodic traffic
ID_RP, RP_RQ Transaction Length, aperiodic traffic
ID_DAT, RP_DAT Transaction Length, aperiodic traffic

Fig. 6.5 Schedule for the aperiodic traffic, considering the example of Table 3.3 (using RM to schedule the periodic traffic)

In Fig. 6.5, the time available for the aperiodic traffic in the first microcycle is 1-6×0.0976=0.414ms. This time allows for the processing of four Ca^* transactions. In the second microcycle the aperiodic window has the length of 1-0.097=0.902ms, which allows for nine Ca^* transactions. Finally, in the third microcycle the length of the aperiodic window is 0.804ms, and all the remaining five Ca^* transactions are processed. Note that each aperiodic transfer corresponds to, in the worst-case, two Ca^* transactions (*ID_RQ* / *RP_RQ* followed by *ID_DAT* / *RP_DAT*). It follows that the length of the aperiodic busy interval (ABI) is: $2\times\mu Cy+2\times0.0976+ +5\times0.1=2.695$ms.

For an aperiodic request made at the station that produces periodic variable F (assume that F is the only periodic variable produced in that station), the dead interval (given by equation (6.15)) is 6.2928ms. Therefore, the worst-case response time of an aperiodic variable requested in station k is: Ra^k=6.2928+2.695=8.9879ms.

6.7. Summary

In WorldFIP networks, the bus arbitrator table (BAT) regulates the scheduling of all buffer transfers. In practice, two types of buffer transfers can be considered: periodic and aperiodic (sporadic). The BAT imposes the schedule of the periodic buffer transfers, and also regulates the aperiodic buffer transfers.

In this Chapter we provided a comprehensive study on how to configure a WorldFIP bus arbitrator table (BAT), in order to guarantee that periodic data transfers are performed before their deadlines. Important contributions were made in the definition of simple feasibility tests for the periodic traffic scheduled according to both the RM and EDF approaches. We showed that while allowing for an increased utilisation level of the network, the EDF scheduling induces an increased communication jitter.

Concerning the aperiodic traffic, we showed how some previous works (Pedro and Burns, 1997; Vasques and Juanole, 1994) proved to be quite pessimistic. We significantly improved previous results by approaching the analysis for the aperiodic traffic in an integrated manner with the methodologies used to build the bus arbitrator table. Thus, we reduce the pessimism considering the actual length of the periodic window in each microcycle. We also considered the effect of both the communication jitter and the padding window.

6.8. References

Pedro, P. and Burns, A. (1997). Worst Case Response Time Analysis of Hard Real-time Sporadic Traffic in FIP Networks. In *Proceedings of the 9th Euromicro Workshop on Real-Time Systems*, pp. 3-10.

Tovar, E. and Vasques, F. (1999a). Distributed Computing for the Factory-Floor: a Real-Time Approach Using WorldFIP Networks. Polytechnic Institute of Porto, Technical Report HURRAY-TR-9908, March 1999, submitted to *Computers in Industry*, Elsevier Science.

Tovar, E. and Vasques, F. (1999b). Factory Communications: on the Configuration of the WorldFIP Bus Arbitrator Table. Polytechnic Institute of Porto, Technical Report HURRAY-TR-9909, March 1999, submitted to the *7th IEEE International Conference on Emerging Technologies and Factory Automation*.

Tovar, E. and Vasques, F. (1999c). Contributions for the Worst-Case Response Time Analysis of Real-Time Sporadic Traffic in WorldFIP Networks. Polytechnic Institute of Porto, Technical Report HURRAY-TR-9910, March 1999, to be presented at the *WIP Session of the 11th Euromicro Workshop on Real-Time Systems*.

Tovar, E. and Vasques, F. (1999d). Engineering Real-Time Applications with WorldFIP: Analysis and Tools. Polytechnic Institute of Porto, Technical Report HURRAY-TR-9912, April 1999.

Vasques, F. and Juanole, G. (1994). Pre-run-time Schedulability Analysis in Fieldbus Networks. In *Proceedings of the Annual Conference of the IEEE Industrial Electronics Society*, pp. 1200-1204.

Chapter 7

Adding Local Scheduling Mechanisms to P-NET and PROFIBUS Masters

In this chapter we propose a methodology for the implementation of priority-based scheduling mechanisms at the application process level of P-NET and PROFIBUS masters. This chapter is partially drawn from the following published work: "Adding Local Priority-Based Dispatching Mechanisms to P-NET Networks: a Fixed Priority Approach" (Tovar *et al.*, 1998) and "From Task Scheduling in Single Processor Environments to Message Scheduling in a Profibus Fieldbus Network" (Tovar and Vasques, 1999).

7.1. Introduction

The worst-case response time of messages in P-NET and PROFIBUS networks, and thus the pre-run-time schedulability condition, is highly dependent on the number of message streams in each master station (please refer to Section 4.4.5 and Section 5.4.3 for the P-NET and PROFIBUS cases, respectively). This problem, which is a consequence of the first-come-first-served (FCFS) nature of the communication queues (Fig. 7.1a), may preclude the use of P-NET and PROFIBUS networks, in applications involving master stations dealing with a large number of I/O points (resulting in a large number of message streams per master station).

The impact of the FCFS behaviour in the message response time, motivated us to consider priority-based queuing mechanisms implemented at the masters' application process (AP) level (Fig. 7.1b). The priority-based queuing mechanisms must be implemented at the AP level, to preserve compliance with the standards.

In the proposed architecture, requests generated at the AP level are stored in an AP queue. Priority-based mechanisms are provided to guarantee that at each token visit, the highest-priority message request will be in the MAC communication queue (which has a length of one message). Priorities can be assigned using one of the following priority assignment schemes: RM/DM or EDF. Note that in the case of PROFIBUS masters, these priority-based mechanisms are only intended for message requests of the high-priority type.

The remainder of this chapter is organised as follows. In Section 7.2 we discuss the analogy between non pre-emptive task scheduling in single processor environments and message scheduling in token-based networks. In Section 7.3, we provide (token) utilisation-based tests for both types of token-passing networks. In Section 7.4 we propose worst-case response time tests for both RM/DM and EDF dispatched messages in P-NET and PROFIBUS networks. We also show how the (token) utilisation-based tests can be quite pessimistic. In Section 7.5, we extend the proposed analysis considering the actual token utilisation (presented in Section 4.4) in P-NET networks.

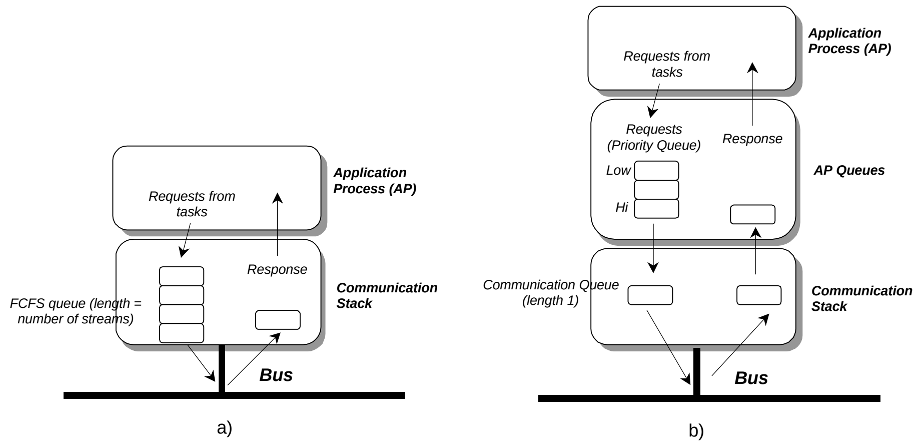


Fig. 7.1 This figure illustrates compares the original architecture of both P-NET and PROFIBUS masters (a) with the proposed architecture (b)

7.2. From Task to Message Scheduling: Analogies and Adaptations

In this section we discuss the analogy between task scheduling in a single processor environment and message scheduling in token-passing networks (considering, at most, one processed message per token visit). This analogy will later enable the formulation of feasibility tests for P-NET and PROFIBUS message stream sets.

7.2.1. Homogenisation of Notations

Firstly, in order to apply the proposed methodology to both token-passing networks, we need to homogenise the assumptions and the notations used in the previous analysis of P-NET (Chapter 4) and PROFIBUS (Chapter 5) networks.

The proposed methodology is based on the following assumption: in P-NET and PROFIBUS networks, at each token visit, at most only one real-time message cycle is performed. While in the case of P-NET networks this is its standard behaviour, in the case of PROFIBUS networks this results from considering the unconstrained low-priority traffic profile (Section 3.4.2).

Concerning the previously defined notations, the upper bound for the token inter-arrival time at a master station k has been denoted as V (equation (4.4)) or as T^k_{cycle} (equation (5.6)), for the cases of P-NET and PROFIBUS networks, respectively. In order to have a similar notation for both cases, we will use V to denote the upper bound for the token cycle time. We will also use the notation ns^k to refer to the number of real-time message streams in a master k (P-NET or PROFIBUS master). Therefore, ns^k stands for

nh^k, which was used in Chapter 5 to denote the number of high-priority message streams belonging to a PROFIBUS master. Similarly, the PROFIBUS high-priority message stream set is now denoted as S_i^k, instead of Sh_i^k.

7.2.2. Analogies to the Blocking and Task's Computation Times

In the schedulability analysis of tasks in non pre-emptive single processor environments, the concept of processor's busy period denotes the time interval within which the processor is not idle (see Section 2.4.4). Assume the following task set:

Table 7.1: A Task Set Example ($D = T$)

Task	*Computation Time (C)*	*Period (T)*
A	10	60
B	10	80
C	10	100
D	10	100

Assuming a RM priority assignment policy in a non pre-emptive context, Fig. 7.2 illustrates a time-line considering that the first instance of task *D* (lower-priority task) is released marginally after time instant 0, and before all other instances of higher-priority tasks.

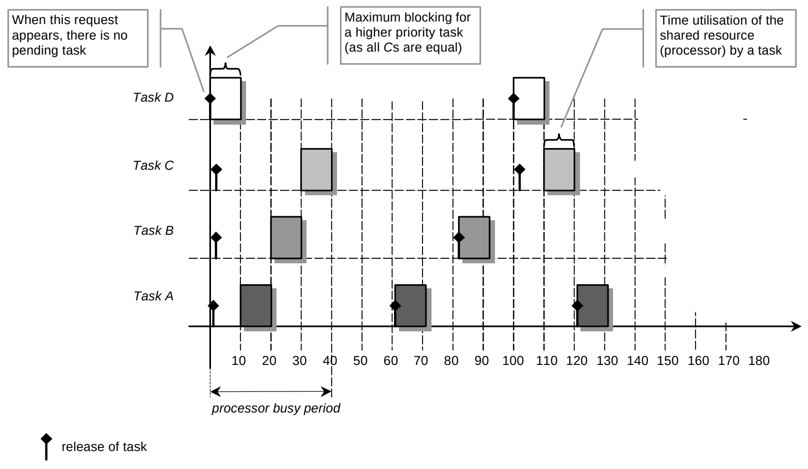


Fig. 7.2 Scheduling example (using RM) for the task set shown in Table 7.1

Note that the blocking of a task in a non pre-emptive context is equal to the maximum execution length of a lower-priority task (see equation (2.4), in Chapter 2).

Assume now the following message stream set for a token-passing example:

Table 7.2: Message Stream Set Example ($D = T$)

Message	*Message Cycle Length (C)*	*Period (T)*
A	2	60
B	2	80
C	2	100
D	2	100

This case will be shown to be loosely equivalent to the previous task scheduling example, if the token cycle time is equal to the tasks' execution time ($V = C_{task}$).

Consider Fig. 7.3, which illustrates the time-line for a message scheduling with a messages' release pattern (arrival of requests to the queue) similar to the previous tasks' release pattern.

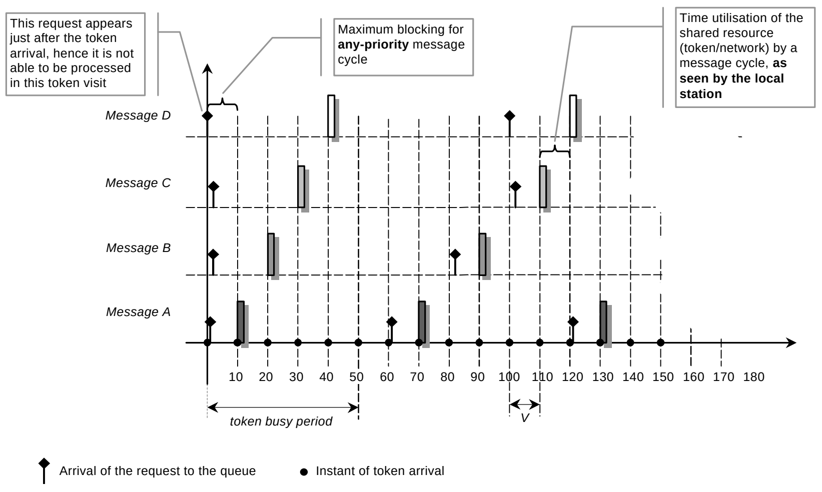

This request appears just after the token arrival, hence it is not able to be processed in this token visit
Maximum blocking for **any-priority** message cycle
Time utilisation of the shared resource (token/network) by a message cycle, **as seen by the local station**
Message D
Message C
Message B
Message A
10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180
token busy period
V
Arrival of the request to the queue
Instant of token arrival

Fig. 7.3 Scheduling example (using RM) for the message cycle set shown in Table 7.2

It is clear that the message blocking time is equal to the token cycle time. However, this blocking term is independent of the priority ordering of message transfers. Therefore, the blocking problem in the task scheduling theory can only be considered to be loosely equivalent to the blocking problem in token-passing networks, since the priority ordering property is not preserved.

It is also clear that the tests available for the schedulability analysis of non pre-emptable tasks in single processor systems can be adapted to the message scheduling in token-passing networks, considering that the blocking term is equal to the token cycle time, independently of the message priority.

Therefore, the computation time of a task can be considered equivalent to the token cycle time, since in a token-passing network, where, at most, only one message is processed per token visit, the shared resource (network access/token) is only available once in every V interval. This means that the contribution of each higher-priority message cycle to the overall queuing delay of a lower-priority message cycle is always equal to V.

Finally, in Table 7.3 we summarise the analogies between task scheduling in non pre-emptive single processor environments and message scheduling in specific token-passing networks (considering, at most, one processed message per token visit).

Table 7.3: Analogies between Task Scheduling and Message Scheduling in Token Passing Networks

	Task Scheduling	*Message Scheduling*
Maximum blocking (except for the lowest priority task/message)	$B_i = \max_{\forall j \in lp(i)} \{C_j\}$	V
Maximum blocking (lowest priority task/message)	0	V
Resource usage time for the higher-priority tasks/messages	C_j	V
Resource usage time for the task/message itself	C_i	C_i

7.2.3. Basic Message Response Time Evaluation

Considering priority-based dispatching mechanisms, the worst-case response time for a message request occurs when the request is placed in the master's queue just after the token arrival, hence not being able to be processed in that token visit. If there were any other message request pending before the token arrival, then the token would have been used to transmit that message; otherwise, the master would not use the token at all.

Therefore, the generic worst-case response time of a message stream S_i^k will be as follows:

$$R_i^k = Q_i^k + C_i^k = V + \Phi \times V + C_i^k \qquad \textbf{(7.1)}$$

where the first term V denotes the message blocking, and the symbol Φ denotes the number of higher-priority messages (interference) that can be scheduled ahead of a message from S_i^k.

Equation (7.1) is similar to equation (5.4) for the PROFIBUS case, considering $\Phi = nh^k - 1$, which means that a message from stream Sh_i^k will suffer the interference from, at most, all other (nh^k - 1) messages.

For the P-NET case (equation (4.5)) there is however a slightly difference, which is a consequence of the better characterisation of its token holding time. In fact, we considered that the critical instant (FCFS case) occurs when all ns^k requests are placed in the outgoing queue just after a message cycle has been completed (see Fig. 4.1). In this chapter, we must re-define the critical instant as being the instant when a message request is placed in the outgoing queue just after the token arrival, hence, not being able to be processed in that token visit. This reflects the worst-case situation, since in the case of priority-based outgoing queues, the message, even if it is the highest-priority one, will be blocked during V. Therefore, Fig. 7.4 updates Fig. 4.4 to reflect this new definition of the critical instant.

This new definition of the critical instant enables the formulation of the response time of a P-NET message similarly to equation (7.1). In fact, adding C_i^k to the previous response time formulation (equation (4.5)), it follows that: $R_i^k = ns^k \times V + C_i^k$, as illustrated by Fig. 7.4.

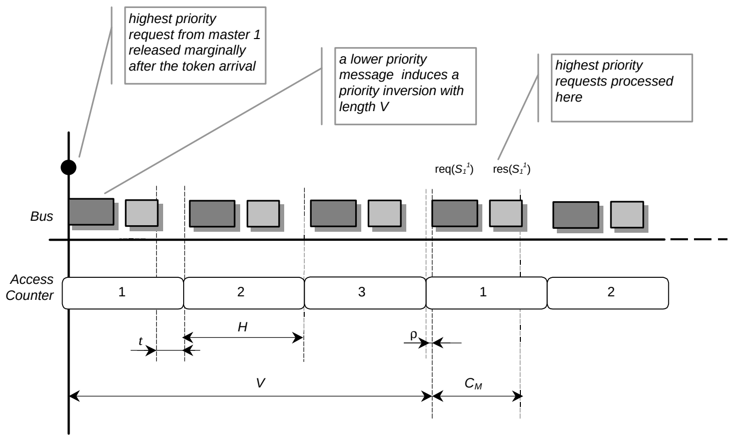

highest priority request from master 1 released marginally after the token arrival
a lower priority message induces a priority inversion with length V
highest priority requests processed here
req(S_1^1)
res(S_1^1)
Bus
Access Counter
1
2
3
1
2
H
t
ρ
V
C_M

Fig. 7.4 New definition of the critical instant

7.3. (Token) Utilisation-Based Tests

In this section, we derive (token) utilisation-based feasibility tests for both fixed and dynamic priority assignment schemes. Such feasibility tests, which can be quite pessimistic, provide an easy tool to evaluate the schedulability of the overall message set with a reduced complexity.

7.3.1. Case of Rate Monotonic Priority Assignment

Considering the analogies to the blocking and tasks' computation time drawn in the previous section, the schedulability test (2.5) for the RM dispatched tasks can be adapted to encompass the characteristics of the P-NET/PROFIBUS token-passing protocols, as follows:

$$\left(\sum_{i=1}^{ns^k} \frac{V}{T_i^k}\right) + \max_{1 \le i \le n}\left\{\frac{V}{T_i^k}\right\} \le ns^k \times \left(2^{\frac{1}{ns^k}} - 1\right), \ \forall_k \qquad \textbf{(7.2)}$$

As the worst-case token cycle time (V) is constant, equation (7.2) can be re-written as:

$$V \times \left[\left(\sum_{i=1}^{ns^k} \frac{1}{T_i^k}\right) + \frac{1}{\min\{T_i^k\}}\right] \le ns^k \times \left(2^{\frac{1}{ns^k}} - 1\right), \ \forall_k \qquad \textbf{(7.3)}$$

Note that, as we are considering token-passing networks, where, at most, one message is transmitted per token arrival, the interference from other masters is only reflected on the evaluation of the V parameter.

Consider the following message stream set:

Table 7.4: Message Stream Set Example

Stream	*Period*
S_1^k	5
S_2^k	7
S_3^k	8
S_4^k	12

Considering that the worst-case token rotation time is V = 1, it follows that the schedulability test (7.3) is:

$$1\times\left[\left(\sum_{i=1}^{4}\frac{1}{T_i^k}\right)+\frac{1}{5}\right]\leq 4\times\left(2^{\frac{1}{4}}-1\right)\Leftrightarrow\frac{1}{5}+\frac{1}{7}+\frac{1}{8}+\frac{1}{12}+\frac{1}{5}=0.75\leq 0.76$$

Therefore the message stream set presented in Table 7.4 is schedulable by the RM algorithm in a token-passing network. In Fig. 7.5, we present a possible time-line for the message scheduling, assuming that all messages are requested just before the first token arrival, but with none of them being dispatched in that first token visit. In this way, we represent a blocking term right at the beginning of the time-line.

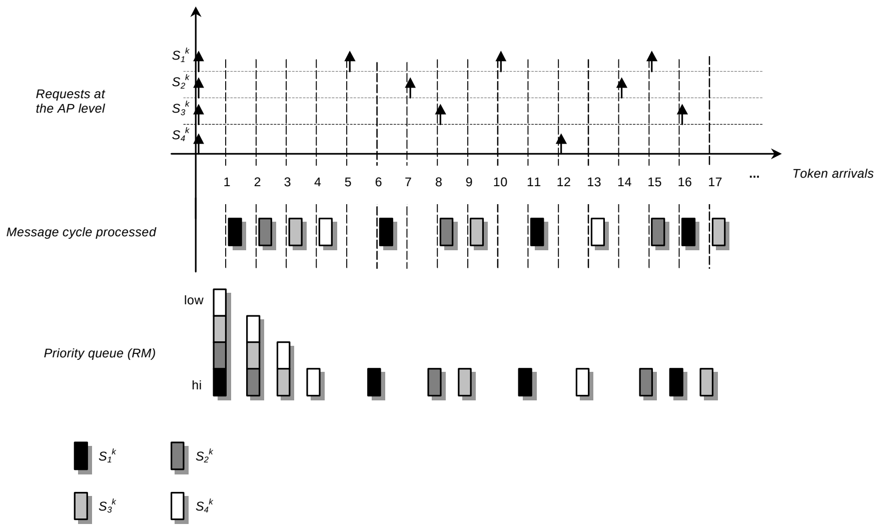


Fig. 7.5 Schedule example for the message stream set of Table 7.4

This example highlights some of the pessimism associated to the utilisation-based tests, since, although the schedulability test is just marginally true, none of the message cycles is scheduled close to its deadline.

7.3.2. Case of Earliest Deadline First Priority Assignment

Considering again the analogies to the blocking and tasks' computation time drawn in Section 7.2, the schedulability test (inequality (2.14), in Section 2.5.1) for the EDF dispatched tasks can also be adapted to encompass the characteristics of the P-NET/PROFIBUS token-passing protocols, as follows:

$$V \times \left[\left(\sum_{i=1}^{ns^k} \frac{1}{T_i^k} \right) + \frac{1}{\min\limits_{1 \le i \le n} \left\{ T_i^k \right\}} \right] \le 1, \ \forall_k \qquad \textbf{(7.4)}$$

Consider the following message stream set, where we consider values for periods that are marginally smaller than multiples of the worst-case token cycle time ($V = 1$):

Table 7.5: Message Stream Set Example

Stream	*Period*
S_1^k	4^-
S_2^k	5^-
S_3^k	6^-
S_4^k	8^-

The application of the schedulability test (7.4) to this message stream set is:

$$1 \times \left[\left(\sum_{i=1}^{4} \frac{1}{T_i^k} \right) + \frac{1}{4} \right] \le 1 \Leftrightarrow \left(\frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{8} + \frac{1}{4} \right) \le 1 \Leftrightarrow 0.99 \le 1 \ \text{TRUE}$$

Hence, this message stream set is schedulable considering the EDF priority assignment scheme, while with the RM assignment scheme would not pass the feasibility test (7.3): $0.99 \le 0.76$ is FALSE.

7.4. Response Time Tests

In this section we derive response time feasibility tests for both fixed and dynamic priority assignment schemes. Such feasibility tests, compared to the (token) utilisation-based tests are more complex, but also much less pessimistic, as it will be shown in this section. This is an expected result, as response time tests for task scheduling are sufficient and necessary conditions, while the utilisation-based tests are generally only sufficient conditions (refer to Chapter 2). It is also important to note that for the case of $D_i^k < T_i^k$ there are no simple utilisation-based tests for the case of fixed priorities.

7.4.1. Response Time Tests: Fixed Priority Assignment

Based on the analogies made in Section 7.2.2, which led to equation (7.1), the worst-case response time analysis for the non pre-emptive context (refer to Section 2.4.4) can be adapted to encompass the characteristics of the P-NET/PROFIBUS token-passing protocols. The worst-case message response time is:

$$R_i^k = Q_i^k + C_i^k \tag{7.5}$$

where Q_i^k is defined as:

$$Q_i^k = V + \sum_{\forall j \in hp(i)} \left\lceil \frac{Q_i^k}{T_j^k} \right\rceil \times V = V \times \left(1 + \sum_{\forall j \in hp(i)} \left\lceil \frac{Q_i^k}{T_j^k} \right\rceil \right) \tag{7.6}$$

Note that this queuing delay is the equivalent to the task's interference in a non pre-emptive context (equation 2.11). Considering again the message stream set example presented in Table 7.5, the worst-case response time for each message stream will be as shown in Table 7.6 (assuming $C_i^k = 0.2, \forall_i$).

Table 7.6: Worst-Case Response Times (RM Case) for the Stream Set of Table 7.5

Stream	***Response***
S_1^k	1.2
S_2^k	2.2
S_3^k	3.2
S_4^k	7.2

For S_4^k, the iterations for evaluating the queuing delay are as follows:

$$Q_4^{k(0)} = 1 + \left\lceil \frac{0}{4^-} \right\rceil + \left\lceil \frac{0}{5^-} \right\rceil + \left\lceil \frac{0}{6^-} \right\rceil = 1; \quad Q_4^{k(1)} = 1 + \left\lceil \frac{1}{4^-} \right\rceil + \left\lceil \frac{1}{5^-} \right\rceil + \left\lceil \frac{1}{6^-} \right\rceil = 4$$

$$Q_4^{k(2)} = 1 + \left\lceil \frac{4}{4^-} \right\rceil + \left\lceil \frac{4}{5^-} \right\rceil + \left\lceil \frac{4}{6^-} \right\rceil = 5; \quad Q_4^{k(3)} = 1 + \left\lceil \frac{5}{4^-} \right\rceil + \left\lceil \frac{5}{5^-} \right\rceil + \left\lceil \frac{5}{6^-} \right\rceil = 6$$

$$Q_4^{k(4)} = 1 + \left\lceil \frac{6}{4^-} \right\rceil + \left\lceil \frac{6}{5^-} \right\rceil + \left\lceil \frac{6}{6^-} \right\rceil = 7; \quad Q_4^{k(5)} = 1 + \left\lceil \frac{7}{4^-} \right\rceil + \left\lceil \frac{7}{5^-} \right\rceil + \left\lceil \frac{7}{6^-} \right\rceil = 7$$

and iterations stop at this point, since as $Q_4^{k(5)} = Q_4^{k(4)} = 7$. Therefore, $R_4^k = 7 + 0.2 = 7.2$, which is smaller than its relative deadline (its period), and thus, the message stream set is RM schedulable.

Considering the same message stream set, the (token) utilisation-based test (7.3) gives $0.99 \leq 0.76$, which is equivalent to state that this message stream set may or not be schedulable. Therefore, it turns out that the response time test is much less pessimistic than the (token) utilisation-based test.

The time-line presented in Fig. 7.6 illustrates the above results.

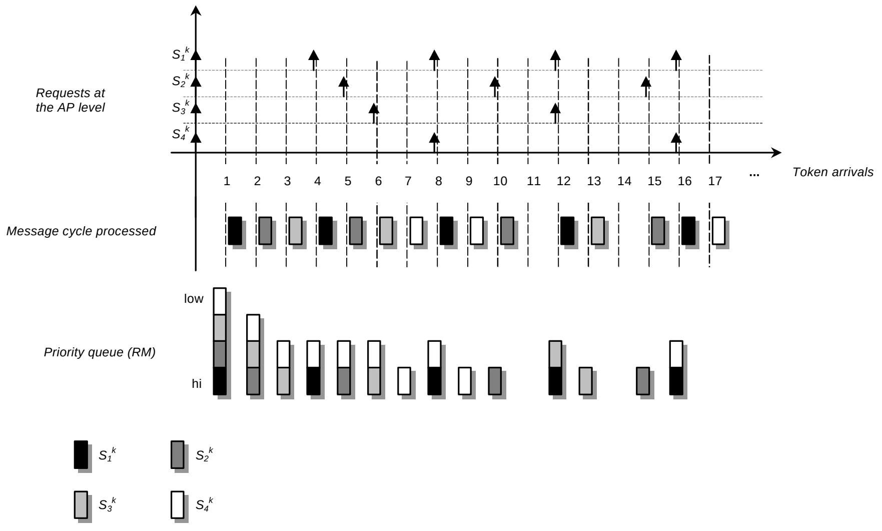


Fig. 7.6 RM Schedule for the message stream set of Table 7.5. This figure also illustrates the worst-case response time for the message stream set (Table 7.6)

7.4.2. Response Time Tests: Dynamic Priority Assignment

Based on the analogies made in Section 7.2.2, which led to equation (7.1), the worst-case response time analysis for the non pre-emptive context (refer to Section 2.5.5) can also be adapted to encompass the characteristics of the P-NET/PROFIBUS token-passing protocols.

The worst-case message response time is, obviously, given by equation (7.5). However, a major difference exists for the definition of the queuing delay, which for the EDF case must be defined as:

$$Q_i^k = V \times \left(1 + \sum_{\substack{j \neq i \\ D_j^k \leq D_i^k}} \min \left\{ 1 + \left\lfloor \frac{Q_i^k}{T_j^k} \right\rfloor , 1 + \left\lfloor \frac{D_i^k - D_j^k}{T_j^k} \right\rfloor \right\} \right) \quad \textbf{(7.7)}$$

that is, a message request concerning stream S_i^k will be delayed by all message requests of other streams having earlier or equal absolute deadlines than the absolute deadline for S_i^k (absolute deadlines are the difference between the relative deadline, D_i^k, and the beginning of the evaluation interval - assumed at time instant 0). Note that while $\sum(1 + \lfloor Q_i^k / T_j^k \rfloor)$ requests having relative deadlines smaller or equal to D_i^k can be placed in the AP queue, from those requests, only a maximum of $1 + \lfloor (D_i^k - D_j^k) / T_j^k \rfloor$ will have absolute deadlines earlier than D_i^k. We illustrate this in the following example.

Assume the message stream set example of Table 7.7 ($D = T$).

Table 7.7: Message Stream Set Example

Stream	***Period***
S_1^k	4^-
S_2^k	5^-
S_3^k	6^-
S_4^k	7^-

If we consider the synchronous release pattern for message streams, a time-line for the EDF schedule will be as illustrated in Fig. 7.7.

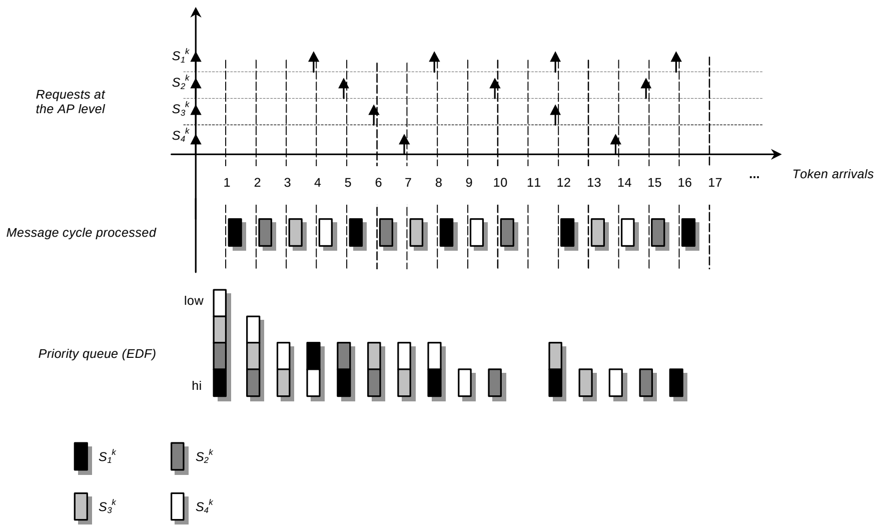


Fig. 7.7 Schedule for the message stream set of Table 7.7, with messages dispatched according the EDF priority assignment, and considering a synchronous release pattern

As it can be seen from Fig. 7.7, there is a request for S_1^k arriving to the queue before the processing of the first request for S_4^k. However, as that request for S_1^k has an absolute deadline which is later than the absolute deadline for S_4^k, it will be processed only after the request for S_4^k. This behaviour of the EDF scheduler is effectively translated by equation (7.7), as can be seen by the following successive iterations ($V = 1$):

$$Q_4^{k(0)} = 1 + \min\left\{1+\left\lfloor \frac{0}{4^-} \right\rfloor, 1+\left[\frac{7^- - 4^-}{5^-} \right]\right\} + \left\{1+\left\lfloor \frac{0}{5^-} \right\rfloor, 1+\left[\frac{7^- - 5^-}{5^-} \right]\right\} +$$

$$+\left\{1+\left\lfloor \frac{0}{6^-} \right\rfloor, 1+\left[\frac{7^- - 4^-}{6^-} \right]\right\} = 1+\min\{1,1\}+\min\{1,1\}+\min\{1,1\} = 4$$

$$Q_4^{k^{(1)}} = 1 + \min\left\{1 + \left\lfloor \frac{4}{4^-} \right\rfloor, 1 + \left\lfloor \frac{7^- - 4^-}{5^-} \right\rfloor\right\} + \left\{1 + \left\lfloor \frac{4}{5^-} \right\rfloor, 1 + \left\lfloor \frac{7^- - 5^-}{5^-} \right\rfloor\right\} +$$
$$+ \left\{1 + \left\lfloor \frac{4}{6^-} \right\rfloor, 1 + \left\lfloor \frac{7^- - 4^-}{6^-} \right\rfloor\right\} = 1 + \min\{2,1\} + \min\{1,1\} + \min\{1,1\} = 4$$

and iterations stop at this point, as $Q_4^{k(1)} = Q_4^{k(0)} = 4$. The maximum queuing delay for a request of stream S_4^k, considering that the streams have a synchronous release pattern, is thus as shown in the time-line illustrated in Fig. 7.7.

Note however that the worst-case response time for EDF dispatched messages is not necessarily found with this synchronous release pattern (refer to Sections 2.5.4 and 2.5.5). Therefore, equation (7.7) must be updated to:

$$Q_i^k(a) = B_i^k(a) + V \times \left(\sum_{\substack{j \neq i \\ D_j^k \leq a + D_i^k}} \left(\min\left\{1 + \left\lfloor \frac{Q_i^k(a)}{T_j^k} \right\rfloor,\ 1 + \left\lfloor \frac{a + D_i^k - D_j^k}{T_j^k} \right\rfloor \right\} \right) + \left\lfloor \frac{a}{T_i^k} \right\rfloor \right) \quad \textbf{(7.8)}$$

where B_i^k is defined as follows:

$$B_i^k(a) = \begin{cases} V, & a = 0 \\ V, & a \neq 0 \ \wedge \ \exists_j : D_j^k > a + D_i^k \end{cases} \quad \textbf{(7.9)}$$

Note that while with the RM/DM approach (Section 7.4.1) the blocking term is *V* and effective for all the message streams, with the EDF approach, we must only consider (if $a \neq 0$) a blocking if it exists a message stream S_j^k ($j \neq i$) with an absolute deadline later than the relative deadline of the instance of S_i^k released at time instant *a*.

A main difference exists in comparison to the analogous formulation for task scheduling (equation 2.28), since in the case of our token-passing model, for $a = 0$ there is always a blocking with the value *V*.

Similarly to the case of task scheduling, *a* belongs to the following set of values:

$$a \in \left\{0, \left\{\bigcup_{l=1}^{ns^k} \left\{\Psi \times T_l^k + D_l^k - D_i^k,\ \Psi \in \aleph_0\right\} \cap [0, L[\right\}\right\} \quad \textbf{(7.10)}$$

where the (token) synchronous busy period is given by:

$$L = \sum_{i=1}^{ns^k} \left\lceil \frac{L}{T_i^k} \right\rceil \times V \quad \textbf{(7.11)}$$

In Appendix D.1, we give the pseudo code details of the algorithm used to evaluate the value for *L*.

The queuing delay is thus:

$$Q_i^k = \max_a \left\{0, Q_i^k(a) - a\right\} \quad \textbf{(7.12)}$$

since the computation $Q_i^k(a)$ may occasionally give a value smaller than a (for instance, when the value of a corresponds to more than one request of S_i^k during the interval under analysis, the interval $[0,Q_i^k(a)]$.

Finally, substituting equation (7.12) back in equation (7.5), we define the worst-case response time of a message stream dispatched according to the EDF scheme as follows:

$$R_i^k = \max_a \{0, Q_i^k(a) - a\} + C_i^k \tag{7.13}$$

In Appendix D.1, D.2 and D.3, we give the pseudo code details for the evaluation of L (equation (7.11)), for the determination of the a values for each stream S_i^k (equation (7.10)), and for the evaluation of the Q_i^k (equation (7.8)), respectively.

The analysis outlined will be now illustrated for the stream set example of table 7.7. The results presented were obtained using the following exact characterisation of the message stream set of table 7.7:

Table 7.8: Exact Characterisation of the Message Stream Set of Table 7.7

Stream	C_i^k	T_i^k	D_i^k
S_1^k	0.2	3.99	3.99
S_2^k	0.2	4.99	4.99
S_3^k	0.2	5.99	5.99
S_4^k	0.2	6.99	6.99

For this message stream set, the value for L (upper bound for a) is (using (7.11) - Algorithm D.1): $L = 9$. Therefore, the values of a that must be tested for each message stream (equation (7.10) - Algorithm D.2) is:

Table 7.9: *a* Values Concerning Stream Set of Table 7.8

Stream	***a* = 0**	***a1***	***a2***	***a3***	***a4***	***a5***	***a6***	***a7***
S_1^k	0.00	1.00	2.00	3.00	3.99	5.99	7.98	7.99
S_2^k	0.00	1.00	2.00	2.99	4.99	6.98	6.99	8.99
S_3^k	0.00	1.00	1.99	3.99	5.98	5.99	7.99	8.98
S_4^k	0.00	0.99	2.99	4.98	4.99	6.99	7.98	8.97

In order to evaluate the queuing delay for each release pattern, equation (7.8) must be evaluated for each a value (Algorithm D.3). The results for $(Q_i^k(a) - a)$ are:

Table 7.10: Results for all $(Q_i^k(a) - a)$ Concerning the Message Stream Set of Table 7.8

Stream	***a* = 0**	***a1***	***a2***	***a3***	***a4***	***a5***	***a6***	***a7***
S_1^k	1.00	1.00	1.00	0.00	-0.99	1.01	-0.98	0.01
S_2^k	2.00	2.00	1.00	0.01	-1.99	0.02	2.01	1.01
S_3^k	3.00	2.00	1.01	-0.99	-2.98	-2.99	2.01	2.02
S_4^k	4.00	2.01	0.01	-1.98	-1.99	-3.99	3.02	2.03

In this table, for each message stream the value of $\max\{0, Q_i^k(a) - a\}$ is highlighted. The worst-case response times for the message streams are presented in Table 7.11.

Table 7.11: Worst-Case Response Times for the Message Stream Set of Table 7.8

Stream	*Response*	*a*
S_1^k	2.01+0.2=1.21	5.99
S_2^k	2.01+0.2=2.21	6.99
S_3^k	3.00+0.2=3.20	0.00
S_4^k	4.00+0.2=4.20	0.00

Therefore, the message stream set is EDF schedulable, since $R_i^k \le T_i^k$ (D_i^k), $\forall_i$, while it would not be schedulable with the RM approach. In fact, stream S_4^k, and using equation (7.6), will have the following worst-case queuing delay:

$$Q_4^{k(0)} = 1 + \left\lceil \frac{0}{4^-} \right\rceil + \left\lceil \frac{0}{5^-} \right\rceil + \left\lceil \frac{0}{6^-} \right\rceil = 1; \quad Q_4^{k(1)} = 1 + \left\lceil \frac{1}{4^-} \right\rceil + \left\lceil \frac{1}{5^-} \right\rceil + \left\lceil \frac{1}{6^-} \right\rceil = 4$$

$$Q_4^{k(2)} = 1 + \left\lceil \frac{4}{4^-} \right\rceil + \left\lceil \frac{4}{5^-} \right\rceil + \left\lceil \frac{4}{6^-} \right\rceil = 5; \quad Q_4^{k(3)} = 1 + \left\lceil \frac{5}{4^-} \right\rceil + \left\lceil \frac{5}{5^-} \right\rceil + \left\lceil \frac{5}{6^-} \right\rceil = 6$$

$$Q_4^{k(4)} = 1 + \left\lceil \frac{6}{4^-} \right\rceil + \left\lceil \frac{6}{5^-} \right\rceil + \left\lceil \frac{6}{6^-} \right\rceil = 7; \quad Q_4^{k(5)} = 1 + \left\lceil \frac{7}{4^-} \right\rceil + \left\lceil \frac{7}{5^-} \right\rceil + \left\lceil \frac{7}{6^-} \right\rceil = 7$$

and thus R_4^k = 7 + 0.2 = 7.2, which is larger than T_4^k (D_4^k)= 6.99. Fig. 7.8 puts this to evidence.

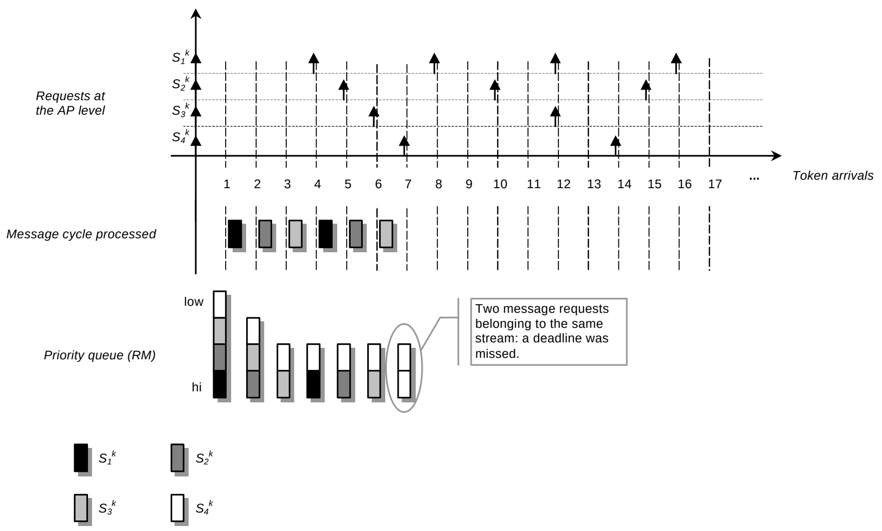


Fig. 7.8 RM schedule for the stream set of Table 7.7

As a final remark, it is important to note that the stream set of Table 7.8 does not emphasise the importance of parameter *a* in equation (7.10). This is only due to the specific characteristics of the particular stream set. In fact, the results in Tables 7.10 and 7.11 show that considering *a* = 0 corresponds virtually to the actual worst-case response time. The following example will better illustrate the importance of parameter *a* in the evaluation of the queuing delay. The only difference to the previous example is the value of D_2^k.

Table 7.12: Message Stream Set Example

Stream	C_i^k	T_i^k	D_i^k
S_1^k	0.2	3.99	3.99
S_2^k	0.2	4.99	3.90
S_3^k	0.2	5.99	5.99
S_4^k	0.2	6.99	6.99

For this stream set example, the set of values for *a* would be as follows ($L = 9$):

Table 7.13: *a* Values Concerning Stream Set of Table 7.12

Stream	*a* = 0	*a1*	*a2*	*a3*	*a4*	*a5*	*a6*	*a7*
S_1^k	0.00	2.00	3.00	3.99	4.90	7.98	7.99	---
S_2^k	0.00	0.09	2.09	3.09	4.08	4.99	8.07	8.08
S_3^k	0.00	1.00	1.99	2.90	5.98	5.99	7.89	7.99
S_4^k	0.00	0.99	1.90	4.98	4.99	6.89	6.99	8.97

Using the resulting values for each $Q_i^k(a)$, the difference ($Q_i^k(a) - a$) results as follows:

Table 7.14: Results for all ($Q_i^k(a) - a$) Concerning the Message Stream Set of Table 7.12

Stream	*a* = 0	*a1*	*a2*	*a3*	*a4*	*a5*	*a6*	*a7*
S_1^k	2.00	1.00	0.00	-0.99	1.10	-0.98	0.01	---
S_2^k	1.00	1.91	0.91	-0.09	-1.08	-1.99	-1.07	0.92
S_3^k	3.00	2.00	1.01	0.10	-2.98	-2.99	1.11	3.01
S_4^k	4.00	2.01	1.10	-1.98	-1.99	-3.89	-3.99	2.03

As it can be seen, for stream S_2^k, with $a = 0.09$, the queuing delay (as compared to the case of $a = 0$) increases from 1.00 to 1.91. This is an understandable result, as its "absolute deadline" will then be 0.09 + 3.90 = 3.99, and therefore, S_i^k will be scheduled earlier.

7.5. Considering the effect of Unused Tokens (P-NET Networks)

The worst-case response time tests developed in Section 7.3 and 7.4 embody a non-negligible level of pessimism, which results from considering that the real token rotation time is constant and equal to *V*.

In this section we extend that analysis to consider the approach outlined in Section 4.4, where we developed worst-case response time analysis of P-NET messages considering the effect of the unused tokens, hence considering the actual token rotation time.

7.5.1. Extending the Response Time Tests for the Fixed Priority Assignment

The worst-case queuing delay (7.6) can be updated to include the effect of unused tokens as follows:

$$Q_i^k = V \times \left(1 + \sum_{\forall j \in hp(i)} \left\lceil \frac{Q_i^k}{T_j^k} \right\rceil \right) - \left(\sum_{y=1}^{n} Ut^y \right) \times (H - s) \tag{7.14}$$

with H as defined in equation (4.3), and with Ut^y (4.16) re-defined as follows:

$$Ut^y = NC_i^k - \min\left\{ NC_i^k,\ ns^y + \sum_{l=1}^{ns^y} \left\lceil \frac{Q_i^k + Ja*^y}{T_i^y} \right\rceil \right\} \tag{7.15}$$

In equation (7.15), NC_i^k is the worst-case number of consecutive token visits needed to process a message cycle of stream S_i^k. Note that this number is no longer ns^k and it depends on the priority of the particular message stream. NC_i^k is given by:

$$NC_i^k = 1 + \sum_{\forall j \in hp(i)} \left\lceil \frac{Q_i^k}{T_j^k} \right\rceil \tag{7.16}$$

In equation (5.15), $Ja*^y$ denotes the master's logical ring aggregate jitter (refer to Definition 4.7, in Chapter 4). Its evaluation must be subject to some modifications, which will now be explained.

The critical instant in master k is now assumed to be marginally after the token arrival. Compared to the critical instant (t_c) defined in Section 4.4, the new definition for the critical instant (refer to Section 7.2.3) is: $t_c + t + r$.

Therefore, the master's logical ring request jitter (see Definition 4.4) is now given by:

$$Jr^{*y} = Jr^y + t + r \tag{7.17}$$

where Jr^y is the previously defined logical ring request jitter (4.11).

Some adaptations must also be made for the definition of the logical ring visit jitter (Jv^y - see Definition (4.6)), since equation (7.14) is only for the queuing delay (refer to Section 2.4.4 for an explanation). Therefore, $Jv*^y$ must be defined as:

$$Jv^{*y} = Jv^y - C_M \tag{7.18}$$

with Jv^y as defined by equation (4.15), and C_M the length of any message cycle in the network.

Consequently, $Ja*^y$ is defined as follows:

$$Ja^{*y} = Jr^{*y} - Jv^{*y} = Jr^y + t + r - Jv^y + C_M = Jr^y - Jv^y + H = Ja^y + H \tag{7.19}$$

Finally, substituting equations (7.15) and (7.16) in (7.14) we have the formulation for the worst-case queuing delay of a stream S_i^k, scheduled according to RM/DM priority scheme, and considering the actual token rotation times.

$$
\begin{aligned}
Q_i^k &= V \times \left(1 + \sum_{\forall j \in hp(i)} \left\lceil \frac{Q_i^k}{T_j^k} \right\rceil \right) - \\
&- \left(\sum_{y=1}^{n} \left(1 + \sum_{\forall j \in hp(i)} \left\lceil \frac{Q_i^k}{T_j^k} \right\rceil - \min\left\{ 1 + \sum_{\forall j \in hp(i)} \left\lceil \frac{Q_i^k}{T_j^k} \right\rceil ,\ ns^y + \sum_{l=1}^{ns^y} \left\lfloor \frac{Q_i^k + Ja*^y}{T_i^y} \right\rfloor \right\} \right) \right) \times (H - s)
\end{aligned}
\qquad \textbf{(7.20)}
$$

with $Ja*^y$ as defined in equation (7.19).

7.5.2. Extending the Response Time Tests for the Dynamic Priority Assignment

As the worst-case response time may occur for a message request of stream S_i^k produced at time instant a we need first to determine the value for a that leads to the worst-case response time (using the methodology outlined in Section 7.4). We denote this value of a as a_i^k.

Equation (7.8) can be updated to:

$$
\begin{aligned}
Q_i^k &= B_i^k\left(a_i^k\right) + V \times \left(\sum_{\substack{j \neq i \\ D_j^k \le a_i^k + D_i^k}} \left(\min\left\{ 1 + \left\lfloor \frac{Q_i^k}{T_j^k} \right\rfloor ,\ 1 + \left\lfloor \frac{a_i^k + D_i^k - D_j^k}{T_j^k} \right\rfloor \right\} \right) + \left\lfloor \frac{a_i^k}{T_i^k} \right\rfloor \right) - \\
&- \left(\sum_{y=1}^{n} Ut^y \right) \times (H - s)
\end{aligned}
\qquad \textbf{(7.21)}
$$

with a_i^k being the value of a which verifies equation (7.12) with respect to equation (7.8).

Ut^y (4.16) must be re-defined as follows:

$$
Ut^y = NCedf_i^{\ k} - \min\left\{ NCedf_i^{\ k},\ ns^y + \sum_{l=1}^{ns^y} \left\lfloor \frac{Q_i^k + Ja*^y + \Omega - a_i^k}{T_i^y} \right\rfloor \right\} \qquad \textbf{(7.22)}
$$

In this equation we are considering the maximum interference from masters y ($y \neq k$), as we are considering that ns^y requests are placed in each master y at time instant $J*r^y + \Omega$ before a_i^k. Ω represents the offset for the logical ring release jitter (difference between the value of a_i^k and the previous token arrival at master k). Finnaly, the number of consecutive token visits needed to process a message cycle of stream S_i^k is now defined as follows:

$$
NCedf_i^{\ k} = \max\left\{ 0,\ 1 + \sum_{\substack{j \neq i \\ D_j^k \le a + D_i^k}} \left(\min\left\{ 1 + \left\lfloor \frac{Q_i^k}{T_j^k} \right\rfloor ,\ 1 + \left\lfloor \frac{a_i^k + D_i^k - D_j^k}{T_j^k} \right\rfloor \right\} \right) + \left\lfloor \frac{a_i^k}{T_i^k} \right\rfloor - \left\lceil \frac{a_i^k}{V} \right\rceil \right\} \qquad \textbf{(7.23)}
$$

Note that equation (7.23) gives only the number of token rotations starting from time instant a_i^k, since we are subtracting the value $\lceil a_i^k / V \rceil$, as $\lceil a_i^k / V \rceil$ corresponds to the number of token rotations from the beginning of the interval under evaluation (time instant 0) up to time instant a_i^k.

7.6. Summary

The worst-case response time of messages in P-NET and PROFIBUS networks is highly dependent on the number of message streams in each master. The impact of the FCFS behaviour in the message response time, motivated us to consider priority-based queuing mechanisms implemented at the masters' application process (AP) level.

We proposed an architecture where requests generated at the AP level are stored in an AP queue. Priority-based mechanisms are then provided to guarantee that at each token visit, the highest-priority message request will be the one to be transmitted.

Our main contribution was the adaptation, by providing the convenient analogies, of the feasibility tests available for task sets such as they could also be used as feasibility tests of messages in P-NET and PROFIBUS.

We reasoned on how the blocking effect (resulting from non pre-emption) in the schedulability analysis of tasks could be mapped to each case of priority scheme used to schedule messages. We showed how the worst-case execution time of tasks could be translated to the upper bound of the token rotation time in P-NET and PROFIBUS token passing networks. More important, we demonstrated how the simple utilisation-based feasibility tests for non pre-emptive independent tasks could be easily adapted to be used as (token) utilisation-based tests. However, as these tests can be quite pessimistic, we developed response-time tests which were also adapted from the well know response time tests used for RM/DM scheduled non pre-emptable independent tasks and we also adapted the more recently developed response time tests for EDF scheduled non pre-emptable independent tasks. Finally, we illustrated, for the P-NET case, how the analysis may turn even more effective by embodying previous analysis (Section 4.4) considering the actual token rotation time.

7.7. References

Tovar, E., Vasques, F. and Burns, A. (1998). Adding Local Priority-Based Dispatching Mechanisms to P-NET Networks: a Fixed Priority Approach. Polytechnic Institute of Porto, Technical Report HURRAY-TR-9822, December 1998, to appear in the *Proceedings of the 11th Euromicro Conference on Real-Time Systems*.

Tovar, E. and Vasques, F. (1999). From Task Scheduling in Single Processor Environments to Message Scheduling in a Profibus Fieldbus Network. In *Lecture Notes in Computer Science,* No. 1586, pp. 339-352.

Chapter 8

Conclusions and Future Work

In this chapter we review both the research objectives and results of this thesis, giving emphasis on how the main contributions targeted the original research objectives. Finally, we provide some perspectives on future research work

8.1. Review of the Research Objectives

Fieldbus networks are widely used as the communication support for distributed computer-controlled systems (DCCS), in applications ranging from process control to discrete manufacturing. Usually, DCCS impose real-time requirements; that is, traffic must be sent and received within a bounded interval, otherwise a timing fault is said to occur.

Fieldbus networks are usually broadcast networks, where several network nodes share a common communication medium. Messages are transmitted from a source node to a destination node via the shared communication medium. As a consequence, messages to be transmitted by a specific network node may experience some queuing delays, resulting not only from contention between message requests from the same network node, but also with message requests from the other network nodes.

Therefore, a potential leap towards the use of fieldbuses in time-critical applications lies on the accurate characterisation of the temporal behaviour of its Medium Access Control (MAC) protocol.

The main research objective of this thesis was to develop analysis and methodologies to guarantee, prior to run-time, that real-time distributed computer-controlled systems (DCCS) could be successfully implemented with standard fieldbus communication networks, such as those defined by the European Standard EN 50170: P-NRT, PROFIBUS and WorldFIP.

A DCCS is implemented by a set of computational devices. Each computational device runs a number of tasks. These tasks communicate their results by passing messages between computational devices across a communication network.

In this thesis we addressed one of the components that contribute to the overall latencies of the communicating tasks: messages' response time. In order to guarantee that the timing requirements of DCCS are met, the communication delay between a sending task queuing a message, and the related receiving task (it can be in the same network node, or even the same task) being able to access that message, must be upper bounded.

A basic requirement for a network used as the communication support for DCCS, is that a deterministic access to the shared communication medium must be provided to the message requests, or in other words, the queuing delay of a message request must be upper bounded.

Considering this broad definition for deterministic access, all the three analysed protocols fulfil this basic requirement. In fact, not many local area networks miss this requirement, at least in the way we have defined it. A notable example that confirms the exception is the CSMA/CD MAC protocol.

In WorldFIP, and concerning the periodic traffic, the deterministic access is guaranteed by the bus arbitrator, which controls data transfers according a static-scanning table. For the aperiodic traffic, the deterministic access is also guaranteed, provided that there is at least one microcycle with an aperiodic window length suitable for processing aperiodic requests. Note that in the analysis presented in Chapter 6, Section 6.6, only urgent requests for aperiodic buffer transfers are considered. If message requests and/or normal requests for aperiodic buffer transfers were also supported, these would have unbounded queuing delays, since the BA gives priority to process the urgent queue for aperiodic buffer transfers. However, as these two other services are not intended to support real-time data transfers, the aperiodic transfers can also be considered to have deterministic access.

PROFIBUS adopts a simplified version of the timed-token protocol where each master has a bounded interval to access the network. As in the original time-token protocol, PROFIBUS considers two types of quality of service: synchronous (high-priority or real-time) and asynchronous (low-priority). The main difference of the PROFIBUS protocol, as compared to the original timed-token protocol, is the absence of synchronous bandwidth allocation. In the original timed-token protocol, this parameter could be configured in each station, providing a guaranteed token holding time for processing high-priority messages. In the PROFIBUS protocol, each master is guaranteed to be able to process, at least, one high-priority message request. This configures also a deterministic access type. Contrarily, the other services (low-priority requests) that are provided by the PROFIBUS data link layer offer no guarantees of deterministic access. It depends on the network load of high-priority traffic and on how the token target rotation timer (T_{TR}) is set. Significantly, if T_{TR} is set very close to 0, low-priority traffic will have no access to the network at all. As a conclusion, the low-priority traffic may affect the throughput of the high-priority traffic. However, and considering the worst-case situation, the high-priority traffic still has a guaranteed deterministic access. Hence, we considered that only the high-priority PROFIBUS quality of service is intended to support real-time traffic.

Finally, P-NET also offers a deterministic access, since it is based on a virtual token passing (VTP) mechanism, where each master is guaranteed a bounded access delay. This determinism is not achieved by means of controlling the token rotation time, as for the timed-token protocol. Instead, the bounded access delay is implicitly guaranteed by the fact that, at each token visit, only one message request may be performed. For the P-NET case, it is however important to mention that there is only one type of quality of service for message transfers, which means that, as non real-time traffic cannot be distinguished from real-time traffic, it interferes even more significantly with the real-time traffic.

Therefore, the basic real-time requirement for each one of these three fieldbus protocols is fulfilled: deterministic access.

The subsequent step to achieve our research objectives was to provide analysis to test the messages' timing requirements (namely their deadlines) against the worst-case peak load conditions.

In this thesis we developed, for each one of the three network protocols, a set of analytical methods to analyse, prior to run-time, if no message deadline will be missed. These methods constitute a set of powerful tools to guarantee the timing requirements of distributed time-critical applications where distribution is supported by the communication services of one of these three networks. These methods were presented in Chapters 4, 5 and 6, for P-NET, PROFIBUS and WorldFIP networks, respectively.

In distributed computer-controlled systems, the real-time performance depends also on the ability of the communication networks to schedule messages according to their priorities. While in WorldFIP networks the periodic messages (concerning identified variables) can be scheduled by the bus arbitrator on a priority basis, in P-NET and PROFIBUS networks, the first-come-first-served (FCFS) behaviour of their communication queues precludes this ability. In this thesis we proposed a methodology to guarantee the schedulability of messages based on the priority ordered application process queue, in both P-NET and PROFIBUS masters. The implementation of this priority based scheduling is even more important in P-NET networks, since, as it was mentioned, its communication queue treats all the requests in the same manner. It is however important to stress that RM/DM scheduling is preferable to EDF scheduling in the case of P-NET networks, since, a fixed priority scheme enables the definition of higher-priority and lower-priority message streams and thus the lower-priority traffic does not interfere with the scheduling of the higher-priority message streams.

8.2. Main Research Contributions of this Thesis

We have made some important contributions to the schedulability analysis of messages, for all the three EN 50170 profiles. In fact, and contrarily to the case of CAN networks, there were not much previous work addressing real-time aspects devoted to these fieldbus networks. This is not a surprise, since as CAN is based on a priority bus, all the methodologies used for the task scheduling can almost be directly applied. For the EN 50170 networks, real-time analysis is more challenging, since they must combine specific models for the MAC behaviour with the traditional real-time analysis. We hope that we have made relevant contributions to fill this gap. Below, we summarise our main contributions.

1. We developed a simple methodology to evaluate the worst-case response-time of P-NET messages, based on the concept of full token utilisation. This approach to evaluate the response time of P-NET messages is only effective for P-Net networks with high bus utilisation. Therefore, we extended the analysis to a more sophisticated P-NET model, which considers the actual token utilisation by the different network masters. The major contribution of this model is to provide a less pessimistic, and thus more accurate, analysis for the evaluation of the worst-case communication response time of P-NET messages.

2. Also concerning P-NET networks, we showed how, by using P-NET hopping devices, a significant reduction in the messages' response time could be achieved. However, this increase in the network responsiveness can only be obtained for message transactions within the same network segment. Such multi-hop messages may also have real-time requirements, and thus, we also developed feasibility analysis for this type of messages.
3. We developed a simple worst-case response time analysis for PROFIBUS messages. This analysis improves previous related work, where the FCFS behaviour of the communication queues was not considered. It also improves previous works in the sense that our analysis is based on response time tests, hence less pessimistic.
4. Due to the absence of synchronous bandwidth allocation, guaranteed real-time approaches for PROFIBUS networks very much depend on the upper bound for the inter-arrival time of the token to a station. Therefore, the PROFIBUS token cycle time is an important parameter for the real-time analysis. In this thesis, we provide an accurate evaluation of the PROFIBUS token cycle time, which allows for an accurate evaluation of the worst-case response time of PROFIBUS messages.
5. For the WorldFIP periodic traffic, message deadlines can be easily guaranteed, since the bus arbitrator implements a static schedule for the periodic variables. Therefore, real-time guarantees for periodic traffic very much rely on methodologies for building the WorldFIP BAT. In this thesis, we show how the typical priority assignment schemes can be used to build a WorldFIP BAT, guaranteeing the real-time requirements of the periodic traffic.
6. Concerning the aperiodic traffic, we showed how some previous works revealed to be quite pessimistic. We significantly improved previous results by approaching the analysis for the aperiodic traffic in an integrated way with the methodologies used to set the bus arbitrator table. Thus, we reduce the pessimism by taking into consideration the actual length of the periodic window in each microcycle. Also important, we introduced both the effect of the communication jitter and of the padding window.
7. We demonstrated how the methodologies used to guarantee the timing requirements of tasks in single processor environments, can be successfully adapted to encompass the characteristics of P-NET and PROFIBUS networks. The major importance of these methodologies is that they enable the real-time analysis of P-NET and PROFIBUS messages that are scheduled, at the application process level, according to priority-based schemes. For the proposed architecture we developed feasibility tests, adapted from both the utilisation-based and response time tests used for the schedulability analysis of tasks in single processor environments.

8.3. Future Work

Although the provided analysis and methodologies constitute a set of powerful tools to guarantee the messages' timing requirements of the EN 50170 profiles, some improvements can still be made. More precisely, some reductions in the pessimism level of some results may be achieved, thus improving the analysis in the sense that systems with tighter communication deadlines may be guaranteed.

It is also important to stress that almost all the provided pre-run-time schedulability analysis restrict, in some way, the average utilisation of the system (either the shared processor - for the case of tasks scheduling, or the shared communication medium - in the case of message transfers). This is a consequence of the real-time guaranteed approaches, which rely on testing the systems' schedulability against worst-case peak situations. This is why priority-based dispatching has major advantages over the first-come-first-served dispatching. This is also why among priority-based schemes, and from the schedulability point of view, dynamic priority-based (EDF) have important advantages over fixed priority-based (RM/DM) schemes. Finally, this is also why the determination of the worst-case peak-load conditions must be as close to reality as possible.

In this context we survey some of the provided results which can be improved or extended. This section also briefly describes some interesting research topics, which are worthwhile investigating further.

1. For the case of PROFIBUS networks, in the performed analysis we considered the worst-case token rotation time. This assumption has an important level of pessimism, particularly in the case of a low network load. It would be worthwhile to develop models giving the worst-case time elapsed between any number of token visits, by considering the rate characteristics of the different message streams. In this way the analysis would be improved in the same sense as the analysis provided for P-NET networks in Section 4.4.
2. One important parameter of the original timed-token protocol is the station's synchronous bandwidth allocation. As the P-NET standard allows the on-line change of the master's address, it would be worthwhile to investigate the advantages of allocating a set of addresses to each master instead of a single one. With this method it would be possible to emulate the timed-token protocol, and thus the well-known bandwidth allocation schemes could be applied to P-NET networks.
3. In P-NET, hopping devices relay messages at the data link / network layer. This precludes the use of AP priority-based mechanisms to the P-NET multi-hop messages. It would be worthwhile to investigate methodologies to overcome this aspect. A hypothesis would be to give higher priorities to the multi-hop message streams.
4. For the PROFIBUS, in Section 5.5 we addressed the so-called Constrained Low-Priority Traffic Profile. The approach used to guarantee the real-time behaviour of the high-priority traffic was to set the T_{TR} parameter as large as possible such that at each token visit all high-priority traffic would be processed. Typically this results in a large value for T_{TR}, able to comply with the worst-case peak load token rotation, and therefore, in average, part of the token visits will not be fully utilised to transmit high-priority-traffic. This "time not used" to transmit high-priority traffic, could be used to transmit low-priority traffic, thus enhancing the throughput of the low-priority traffic.

5. In the evaluation of the worst-case response time for the WorldFIP aperiodic requests, the critical instant was defined at the beginning of the macrocycle. This is valid for the methodologies used to set the WorldFIP BAT, as they produce a static schedule where the microcycles most loaded with periodic scans are at the beginning of the macrocycle. There are however some other possibilities to set the WorldFIP BAT. In such case, the critical instant should be re-defined as the worst-case response time for an aperiodic request would occur at a different point of the macrocycle. A solution can be to test all possible microcycles.
6. In the performed analysis, the feasibility tests (both response time and utilisation-based tests) use information on the periodicity of the message streams (minimum inter-request time). However, message requests are queued by application tasks, and therefore, in some way, message requests inherit from tasks their period, and in the case of message requests dispatched in a priority-basis, also their priority. Therefore, one could approach differently the provided feasibility tests, not considering directly the periodicities of the message stream requests but the periodicities of the tasks that generate such requests. In this case, one should include the effect of message release jitter; that is, the minimum inter-arrival time between message requests from the same message stream is likely to be smaller than the minimum interval between two consecutive releases of the sending task. In the worst-case, the message release jitter can be equal to the worst-case response time of the sending task. This approach has been addressed in the literature, and offers a number of advantages, as it is easier to engineer the overall real-time DCCS from the perspective of timing characteristics of the applications tasks.
7. As a final remark, it would be interesting to combine the proposed real-time analysis with some dependability analysis. That is, how mechanisms provided to support some fault-tolerance level can affect the real-time behaviour of the DCCS, realising, on the other hand, that system's dependability is a requirement for a hard real-time system.

Appendix

Pseudo Code Algorithms

A. Pseudo Code Algorithms Referenced in Chapter 4

A.1. Message Worst-Case Response Time Considering Actual Token Utilisation

```
Function pnet_sched_analysis;
input:   n        /* number of masters */
         pass     /* time to pass the token after message cycle */
         idle     /* time to pass the token if no message cycle */
                  /* transmitted */
         ns[w]    /* array containing number of streams in each master; */
                  /* w ranges from 1 to n */
         M[x, y, z]  /* message streams information; */
                     /* x ranges from 1 to n; */
                     /* y ranges from 1 to max (ns[])*/
                     /* z ranges 1 to 3; */
                     /* z = 1(len. of mes. cycle); z = 2 (period); */
                     /* z = 3 (relative deadline) */
output:  O[x, y]  /* similar to M [x, y, z] except for z */
                  /* if O[x, y] = 1 stream marked as not schedulable; */
                  /* if O[x, y] = 0 stream schedul. */
         R[x, y]  /* worst-case response time; x ranges from 1 to n; */
                  /* y ranges from 1 to max (ns[])*/
begin
         /* computation of CM */
         CM = 0;
         for i = 1 to n do
            for j = 1 to ns[i] do
               if M[i, j, 1] > CM then
                  CM = M[i, j, 1]
               end if
            end for
         end for;
         H = react + CM + pass;
         for i = 1 to n do
            R_tdma = ns[i] * n * H;
            R = 0;
            repeat
               R_Before = R; unt = 0;
               for j = 1 to n do
                  if j <> i then
                     /* computation of visit jitter */
                     jv = calc_visit (i, j)
                     /* computation of aggregate jitter */
                     jitter = ((n + i – j) mod n) * H – jv;
```

```
                 add_req = 0;

                 for l = 1 to ns[j] do
                    add_req = add_req + int ((R + jitter)/M[j, l, 2])
                 end for;
                 if (add_req + ns[j]) < ns[i] then
                    unt = unt + (ns[i] – add_req – ns[j])
                 end if
              end if;
              R = R_tdma - unt * (H – idle)
           end for
        until R = R_Before;
        for j = 1 to ns[i] do
           R[i, j] = R;
           if M[i, j, 3] < R then
              /* mark message stream j of master i not schedulable */
              O[i, j] = 1
           end if
        end for
     end for
return O, M


Function visit_jitter (i, j);
input:   i                 /* equivalent master k */
         j                 /* master y */
         H, ns[w], idle    /* global vars */
output:  vj                /* visit jitter */

Begin
     jv = ((n + i – j) mod n) * idle + CM;
     if j > i then
        for k = j + 1 to n do
           if ns[k] >= ns[i] then
              jv = jv + H – idle
           end if
        end for
        for k = 1 to i –1 do
           if ns[k] >= ns[i] then
              jv = jv + H – idle
           end if
        end for
     else
        for k = j + 1 to i –1 do
           if ns[k] >= ns[i] then
              jv = jv + H – idle
           end if
        end for
     end if
return jv
```

B. Pseudo Code Algorithms Referenced in Chapter 5

B.1. Evaluation of the Token Lateness

```
-----------------------------------------------------------------------
- Evaluation of the maximum token lateness
-----------------------------------------------------------------------
function t_del;
input:   n         /* number of master */
         nh[k]     /* number of high-priority message streams */
                   /* in each master k */
         nh[k]     /* number of low-priority message streams */
                   /* in each master k */
         Ch[i,j]   /* array containing the length of each message */
                   /* high-priority message stream */
                   /* i ranging from 1 to n (number of masters) */
                   /* each j ranging from 1 to $nh^k$ */
         Cl[i,j]   /* array containing the length of each message */
                   /* low-priority message stream */
                   /* i ranging from 1 to n */
                   /* each j ranging from 1 to $nl^k$ */

output:
         t_del[k]     /* k ranging from 1 to n */

begin
         /* evaluate the max Ch for each master */
         /* evaluate the max Cl for each master */
         for k = 1 to n do
            maxh = 0;
            maxl = 0;
            for i = 1 to nh[k] do
               if Ch[i,k] > maxh then
                  maxh = Ch[i,k];
               end if;
               if Cl[i,k] > maxl then
                  maxl = Ch[i,k];
               end if;
            end for;
            max_h[k] = maxh;
            max_l[k] = maxl;
         end for;

```

```
   /* evaluate tdel for each master */
   for k = 1 to n do
      maxtdel = 0;
      for j = k to n do
         if max_h[j] > max_l[j] then
            term1 = max_h[j];
         else
            term1 = max_l[j];
         end if;
         term2 = 0;
         for i = j + 1 to n do
            term2 = term2 + max_h[i];
         end for;
         for i = 1 to k - 1 do
            term2 = term2 + max_h[i];
         end for;
         if (term1 + term2) > maxtdel then
            maxtdel = term1 + term2;
         end if;
      end for;

      for j = 1 to k - 1 do
         if max_h[j] > max_l[j] then
            term1 = max_h[j];
         else
            term1 = max_l[j];
         end if;
         term2 = 0;
         if k <> n then
            for i = j + 1 to n do
               term2 = term2 + max_h[i];
            end for;
            for i = 1 to k - 1 do
               term2 = term2 + max_h[i];
            end for;
         else
            for i = j + 1 to n do
               term2 = term2 + max_h[i];
            end for;
         end if;
         if (term1 + term2) > maxtdel then
            maxtdel = term1 + term2;
         end if;
      end for;

      t_del[k] = maxtdel;
   end for;
return t_del;
------------------------------------------------------------------------
```

C. Pseudo Code Algorithms Referenced in Chapter 6

C.1. Evaluation of the Microcycle

```
-------------------------------------------------------------------------
- Evaluation of the Microcycle Value
-------------------------------------------------------------------------
function microcycle;
input:   np       /* number of periodic variables */
         tp[i]    /* vector containing the periodicity of the variables */
output:  μCy      /* value of the microcycle */

begin
         min = MAXINT;
         for i = 1 to np do
            if tp[i] < min then
               min = tp[i]
            end if
         end for;
         μCy = min + 1;
         repeat
            μCy = μCy - 1;
            ctrl = TRUE;
            for i = 1 to np do
               if tp[i] mod μCy <> 0 then
                  ctrl = FALSE
               end if
            end for
         until control = TRUE;
return μCy;
-------------------------------------------------------------------------
```

C.2. Evaluation of the Macrocycle

```
-------------------------------------------------------------------------
- Evaluation of the Macrocycle Value
-------------------------------------------------------------------------
function macrocycle;
input:   np       /* number of periodic variables */
         tp[i]    /* vector containing the periodicity of the variables */
         μCy         /* value of the microcycle */
output:  Mcy         /* value of the macrocycle */

begin
         max = 0;
         for i = 1 to np do
            if tp[i] > max then
               max = tp[i]
            end if
         end for;
         N = max - 1;
         ctrl = FALSE;
```

```
      while ctrl = FALSE do
         N = N + 1;
         ctrl = TRUE;
         for i = 1 to np do
            if N mod (tp[i]/μCy) <> 0 then
               ctrl = FALSE
            end if
         end for
      end while;
      MCy = N × μCy;
return MCy;
-------------------------------------------------------------------------
```

C.3. Building the BAT (RM Approach)

```
-------------------------------------------------------------------------
- Rate Monotonic for Building the BAT
-------------------------------------------------------------------------
function rm_bat;
input:   np       /* number of periodic variables */
         Vp[i,j]  /* array containing the periodicity of the variables */
                  /* ORDERED by periodicities; i ranging from 1 to np */
                  /* and the length of Cpi, j ranging from 1 to 2 */
         μCy      /* value of the microcycle */
         N        /* number of microcycles in the macrocycle */
output:
         bat[i,cycle]   /* i ranging from 1 to np */
                        /* cycle ranging from 1 to N */
begin
      for i = 1 to np do
         cycle = 1;
         repeat
            if load[cycle] + Vp[i,2] <= μCy then
               bat[i,cycle] = 1;
               load[cycle] = load[cycle] + 1;
               cycle = cycle + (Vp[i,1] div μCy)
            else;
               cycle1 = cycle1 + 1;
               ctrl = FALSE;
               repeat
                  if load[cycle1] + Vp[i,2] <= μCy then
                     ctrl = TRUE
                  end if;
               until (ctrl = TRUE) or (cycle1 >= (cycle + (Vp[i,1]
                                  div μCy)));
               if cycle1 >= (cycle + (Vp[i,1] div μCy)) then
                  bat[i,cycle1] = 1;
                  load[cycle1] = load[cycle1] + 1;
                  cycle = cycle + (Vp[i,1] div μCy)
               else
                  /* MARK Vpi NOT SCHEDULABLE with RM Algorithm */
                  cycle = cycle + (Vp[i,1] div μCy)
               end if
            end if
         until cycle > N
      end for
return bat;
-------------------------------------------------------------------------
```

C.4. Building the BAT (EDF Approach)

```
---------------------------------------------------------------------
- Earliest Deadline First for Building the BAT
---------------------------------------------------------------------
function edf_bat;
input:   np       /* number of periodic variables */
         Vp[i,j]  /* array containing the periodicity of the variables */
                  /* i ranging from 1 to np */
                  /* and the length of Cpi, j ranging from 1 to 2 */
         μCy      /* value of the microcycle */
         N        /* number of microcycles in the macrocycle */
output:
         bat[i,cycle]   /* i ranging from 1 to np */
                        /* cycle ranging from 1 to N */
begin
         cycle = 0;
         repeat
            /* determine request generated in each microcycle */
            for i = 1 to np do
               if cycle mod Vp[i,1] = 0 then
                  disp[i,1] = 1;
                  disp[i,2] = Vp[i,1] + cycle;
                  disp[i,3] = cycle
               else
                  /* MARK variable Vpi NOT SCHEDULABLE */
               end if
            end for;

            /* Schedule Variables in Current Microcycle */
            load_full = FALSE;
            repeat
               no_rq = TRUE;
               earliest = MAXINT;
               var_chosen = 0;            /* no variable chosen */

               /* Chose Earliest Deadline from Pending Requests */
               for i = 1 to np do
                  if disp[i,1] = 1 then   /* if there is request */
                     no_rq = FALSE;
                     if disp[i,2] <= earliest then
                        if var_chosen = 0 then
                           earliest = disp[i,2];
                           var_chosen = i
                        else
                           /* decide earliest request from two */
                           /* with equal deadlines */
                           if disp[i,3] < disp[var_chosen,3] then
                              earliest = disp[i,2];
                              var_chosen = i
                           end if
                        end if
                     end if
                  end if
               end for;

               /* Verify the Load in the Current Microcycle */
               if load[cycle + 1] + Vp[i,2] <= μCy then
                  bat[var_chosen, cycle + 1] = 1
                  load[cycle + 1] = load[cycle + 1] + Vp[var_chosen,2];
                  disp[var_chosen,1] = 0;
                  disp[var_chosen,2] = 0;
                  disp[var_chosen,3] = 0;
               else
                  load_full = TRUE
               end if;
            until (load_full = TRUE) or (no_rq = TRUE)

            cycle = cycle + 1;
         until cycle = N;
      return bat;
      ------------------------------------------------------------------------
```

C.5. Evaluation of the Communication Jitter

```
--------------------------------------------------------------
- evaluation of the maximum comm. jitter of a periodic var
--------------------------------------------------------------
function Jitter;
input:   np       /* number of periodic variables */
         Vp[i,j]  /* array containing the periodicity of */
                  /*the variables */
                  /* i ranging from 1 to np */
                  /* and the length of Cpi, */
                  /* j ranging from 1 to 2 */
         µCy      /* value of the microcycle */
         N        /* number of microcycles in the */
                  /* macrocycle */
         bat[i,cycle]   /* i ranging from 1 to np */
                        /* cycle ranging from 1 to N */
output:  J[i]  /* maximum polling jitter of variable Vp_i */

begin
   for i = 1 to np do

      /* Evaluate number of hits of variable Vpi */
      hits = 0;
      for cycle = 1 to N do
         if bat[i,cycle] = 1 then
            hits = hits + 1;
         end if
      end for;

      /* Find first hit in a macrocycle */
      cycle = 0;
      repeat
         cycle = cycle + 1
      until bat[i,cycle] = 1;

      /* Find last hit in a macrocycle */
      cycle1 = N + 1;
      repeat
         cycle1 = cycle1 - 1
      until bat[i,cycle1] = 1;

      /* Evaluate time span between the last hit in a */
      /* macrocycle and the 1st in a subsequent */
```

```
         span = (N - cycle1 + cycle - 1) $\times$ $\mu Cy$;
         load_par = 0;
         for j = 1 to i - 1 do
            if bat[j,cycle] = 1 then
               load_par = load_par + Vp[j,2]
            end if
         end for;
         span = span + load_par;
         load_par = 0;
         for j = 1 to i - 1 do
            if bat[j,cycle1] = 1 then
               load_par = load_par + Vp[j,2]
            end if
         end for;
         span = span + ($\mu Cy$ - load_par);

         /* Evaluate time span between each of the hits */
         /* within a macrocycle */
         for k = 1 to hits - 1 do
            cycle1 = cycle;
            repeat
               cycle1 = cycle1 + 1
            until bat[i,cycle1] = 1;
            span1 = (cycle1 - cycle - 1) $\times$ $\mu Cy$;
            load_par = 0;
            for j = 1 to i - 1 do
               if bat[j,cycle] = 1 then
                  load_par = load_par + Vp[j,2]
               end if
            end for;
            span1 = span1 + ($\mu Cy$ - load_par);
            load_par = 0;
            for j = 1 to i - 1 do
               if bat[j,cycle1] = 1 then
                  load_par = load_par + Vp[j,2]
               end if
            end for;
            span1 = span1 + load_par;

            if span1 > span then
               span = span1
            end if;
            cycle = cycle1;
         end for;
         J[i] = span - Vp[i,1];
      end for
return J;
--------------------------------------------------------------
```

C.6. Number of Microcycles in an Aperiodic Busy Interval

```
-------------------------------------------------------------
- Number of Cycles of the Aperiodic Busy Interval
-------------------------------------------------------------
function ncy_apbi;
input:  np       /* number of periodic variables */
        na       /* number of aperiodic variables */
        μCy      /* value of the microcycle */
        bat[i,l] /* i ranging from 1 to np */
        ca       /* length of any aperiodic transaction */
        cp       /* length of all periodic transaction */
        N        /* number of microcycles in the */
                 /* macrocycle */
output: ncy_abi  /* number of cycles of the abi */

begin
        cycle = 0;
        na_aux = 0;
        repeat
           cycle = cycle + 1;
           count_p = 0;
           for i = 1 to np do
              if bat[i,((cycle - 1) mod N) + 1] = 1 then
                 count_p = count_p + 1;
              end if;
           end for;
           aw = μCy - count_p $\times$ cp;
           na_aux = na_aux + aw div ca;
        until na_aux >= 2 $\times$ na;
        ncy_abi = cycle;
return ncy_abi;
-------------------------------------------------------------
```

C.7. Length of an Aperiodic Busy Interval

```
-------------------------------------------------------------
- Length of Aperiodic Busy Interval
-------------------------------------------------------------
function len_apbi;
input:  np       /* number of periodic variables */
        na       /* number of aperiodic variables */
        μCy      /* value of the microcycle */
        bat[i,l] /* i ranging from 1 to np */
        ca       /* length of any aperiodic transaction */
        cp       /* length of all periodic transactions */
        N        /* number of microcycles in the */
                 /* macrocycle */
        ncy_abi  /* number of microcycles of the abi */
output: len_abi  /* length of the aperiodic busy interval */

begin
        /* determine number of aperiodic transfers in */
        /* the ncy_abi - 1 microcycles */
        for cycle = 1 to ncy_abi - 1 do
```

```
      count_p = 0;
      for i = 1 to np do
         if bat[i, ((cycle - 1) mod N) + 1] = 1 then
            count_p = count_p + 1
         end if
      end for;
      aw = μCy - count_p × cp;
      na_aux = na_aux + aw div ca
   end for;

   /* determine the number of periodic scans */
   /* in microcycle */
   /* number ncy_abi */
   count_p = 0;
   for i = 1 to np do
      if bat[i, ((ncy_abi - 1) mod N) + 1] = 1 then
         count_p = count_p + 1
      end if
   end for;
   len_p = count_p × cp;

   /* determine length of aperiodic busy window */
   len_abi = (ncy_abi - 1) × μCy + len_p + (2 × na -
            na_aux) × ca
return len_abi;
--------------------------------------------------------------
```

D. Pseudo Code Algorithms Referenced in Chapter 7

D.1. Evaluation of the Synchronous (Token) Busy Interval - *L* (EDF case)

```
-------------------------------------------------------------
- Evaluation of the (Token) Synchronous Busy Period
-------------------------------------------------------------
function TSBP;
input:   ns       /* number of streams of master k */
         V        /* token cycle time */
         s[i,j]   /* i ranging from 1 to ns */
                  /* j = 1 -> length of a message cycle of the stream */
                  /* j = 2 -> periodicity of the stream */
                  /* j = 3 -> relative deadline of the stream */
output:  L        /* (Token) Synchronous Busy Period */

begin
         $L = ns \times V$;
         repeat
            L_before = L;
            for i = 1 to ns do
               aux = L / s[i,2];
               if (frac(aux) <> 0) then
                  $L = L + (\mathbf{trunc}(aux) + 1) \times V$
               else
                  $L = L + \mathbf{trunc}(aux) \times V$
               end if;
         until L_before = L;
return L;
-------------------------------------------------------------
```

D.2. Finding the Set of Values for *a* (EDF case)

```
-------------------------------------------------------------
- Finding the Set of Values for the a parameter
-------------------------------------------------------------
function a_values;
input:   ns             /* number of streams of master k */
         L              /* value of for the synchronous busy period */
         s[i,j]         /* i ranging from 1 to ns */
                        /* j = 1 -> length of a message cycle of stream */
                        /* j = 2 -> periodicity of the stream */
                        /* j = 3 -> relative deadline of the stream */
         str            /* particular stream to evaluate
output:  List_a[str,i]  /* list of the values for parameter a, */
                        /* concerning stream str */
                        /* i ranges from 1 to na[str] */
         na[str]        /* number of values for parameter a, */
                        /* concerning stream str */
```

```
begin
    kapa = 0;
    na[str] = 1;
    repeat
       store = FALSE;
       for j = 1 to ns do
          aux = kapa $\times$ s[j,2] + s[j,3] - s[str,3];
          if (aux < L) and (aux > 0) then
             exist = FALSE;
             i = 1;
             while List_a[str,i] <> aux do
                i = i + 1;
             end while;
             if List_a[str,i] <> 0 then
                for i1 = 1 to na[str] + 1 downto i do
                   List_a[str, i1] = List_a[str, i1 - 1]
                end for;
                List_a[str, i1] = aux;
                na[str] = na[str] + 1
             end if;
             store = TRUE;
          end if;
          if (aux < 0) then
             store = TRUE;
          end if;
       end for;
       kapa = kapa + 1
    until store = FALSE
return List_a, na;
--------------------------------------------------------------
```

D.3. Evaluation of the Queuing Delay (EDF case)

```
--------------------------------------------------------------
- Evaluation of the Queuing Delay of a Stream
--------------------------------------------------------------
function Q_Delay;
input:   ns              /* number of streams of master k */
         V               /* token cycle time */
         s[i,j]          /* i ranging from 1 to ns */
                         /* j = 1 -> length of a message cycle */
                         /* of the stream */
                         /* j = 2 -> periodicity of the stream */
                         /* j = 3 -> relative deadline of the stream */
         L               /* (Token) Synchronous Busy Period */
         List_a[str,i]   /* list of the values for parameter a, */
                         /* concerning stream str */
                         /* i ranges from 1 to na[str] */
         na[str]         /* number of values for parameter a, */
                         /* concerning stream str */

output:
         qd[i]           /* maximum queuing delay for a stream i */
                         /* i ranging from 1 to ns */
```

```
begin
    for i = 1 to ns do
       for iter = 1 to na[i]         /* for each value of a */
             a = List_a[i,iter];
             q = 0;
             repeat
                q_before = q;
                requests = 0;
                for j = 1 to ns do
                   if j <> i then
                      if s[j,3] <= (a + a[i,3]) then
                         term1 = 1 + trunc (q / s[j,2]);
                         term1 = 1 + trunc ((a + s[i,2] - s[j,3]) /
                                           s[j,2];
                         if term1 < term2 then
                            minimum = term1
                         else
                            minimum = term2
                         end if;
                         minimum = minimum + trunc(a/s[i,3]);
                      end if;
                   end if;
                   requests = requests + minimum;
                end for;

                block = 0;
                if a = 0 then
                   block = V
                end if;
                for z = 1 to ns do
                   if z <> i then
                      if s[z,3] > (a + s[i,3]) then
                         block = 1;
                      end if;
                   end if;
                end for;

                q = (block + req) × V
             until q = q_before;
          if (q - a) > qd[i] then
             qd[i] = (q - a)
          end if;
       end for;               /* cycle concerning values for a */
    end for;
return qd;
--------------------------------------------------------------
```

List of Publications Related to this Thesis

A. To Appear in Scientific Journals

1. Tovar, E., Vasques, F. and Burns, A. Supporting Real-Time Distributed Computer-Controlled Systems with Multi-hop P-NET Networks. Polytechnic Institute of Porto, Technical Report HURRAY-TR-9813, September 1998, to appear in *Control Engineering Practice*, Pergamon Publishers.

2. Tovar, E. and Vasques, F. Real-Time Fieldbus Communications Using PROFIBUS Networks. Polytechnic Institute of Porto, Technical Report HURRAY-TR-9803, April 1998, to appear in *IEEE Transactions on Industrial Electronics*.

3. Tovar, E. and Vasques, F. Cycle Time Properties of the PROFIBUS Timed Token Protocol. Polytechnic Institute of Porto, Technical Report HURRAY-TR-9811, August 1998, to appear in *Computer Communications*, Elsevier Science.

B. Submitted for Publications in Scientific Journals

4. Tovar, E., Vasques, F. and Burns, A. Communication Response Time in P-NET Networks: Worst-Case Analysis Considering the Actual Token Utilisation. Department of Computer Science, University of York, Technical Report YCS 312, January 1999, submitted for publication to the *Journal of Real-Time Systems*, Kluwer.

5. Tovar, E. and Vasques, F. Distributed Computing for the Factory-Floor: a Real-Time Approach using WorldFIP Networks. Polytechnic Institute of Porto, Technical Report HURRAY-TR-9908, March 1999, submitted to *Computers in Industry*, Elsevier Science.

C. Published in Conference Proceedings

6. Cardoso, A. and Tovar, E. Industrial Communication Networks: Issues on Heterogeneity and Internetworking. In *Proceedings of the 6th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM'96)*, Atlanta, USA, pp. 139-148, May 1996.

7. Tovar, E. and Vasques, F. A Communication Support for Real-Time Distributed Computer Controlled Systems. In *Proceedings of the IEE International Workshop on Discrete Event Systems*, pp.178-183. Published by IEE, April 1998.

8. Tovar, E. and Vasques, F. Enhancing P-NET Real-Time Properties Using Priority Queuing Mechanisms. *In Proceedings of the WIP Session of the 4th IEEE Real-Time Technologies and Applications Symposium*, Denver, Colorado, USA, 3-5 June 1998, pp.27-30. Also Available as Technical Report BUCS-TR-98-013 from Boston University, USA.

9. Tovar, E. and Vasques, F. Guaranteeing Real-Time Message Deadlines in PROFIBUS Networks. In *Proceedings of the 10th Euromicro Workshop on Real-Time Systems*, pp. 79-86, Published by IEEE Computer Society Press, June 1998.

10. Tovar, E. and Vasques, F. Pre-Run-Time Schedulability Analysis of P-NET FieldBus Networks. In *Proceedings of the 24th IEEE Annual Conference of the IEEE Industrial Electronics Society (IECON'98)*, Aachen, Germany, August 1998, pp.236-241.

11. Tovar, E. and Vasques, F. Setting Target Rotation Time in PROFIBUS Based Real-Time Distributed Applications. In *Proceedings of the 15th IFAC Workshop on Distributed*

Computer Control Systems, pp. 1-6, Published by Pergamon, an Imprint of Elsevier Science, September 1998.

12. Tovar, E. and Vasques, F. A Communication Support for Real-Time Distributed Computer Controlled Systems. In *Proceedings of the IEE International Workshop on Discrete Event Systems (WODES'98)*, Cagliari, Italy, 26-28 August 1998, pp.178-183. Published by IEE, August 1998.

13. Tovar, E., Vasques, F. and Burns, A. Evaluating P-NET Message's Response Time with Fixed Priority Queuing at the Application Process Level. In Proceedings of the WIP Session of the 19th IEEE Real-Time Systems Symposium (RTSS'98), Madrid, Spain, 2-4 December 1998, pp.19-22. Also Available as Technical Report UNL-CSE-98-002 from University of Nebraska-Lincoln, USA

14. Tovar, E. and Vasques, F. From Task Scheduling in Single Processor Environments to Message Scheduling in a Profibus Fieldbus Network. In *Lecture Notes in Computer Science*, No. 1586, pp. 339-352, WPDRTS'99, April 1999.

D. To Appear in Conference Proceedings

15. Tovar, E., Vasques, F. and Burns, A. Adding Local Priority-Based Dispatching Mechanisms to P-NET Networks: a Fixed Priority Approach. Polytechnic Institute of Porto, Technical Report HURRAY-TR-9822, December 1998, to appear in the *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, June 1999.

16. Tovar, E. and Vasques, F. Contributions for the Worst-Case Response Time Analysis fo Real-Time Sporadic Traffic in WorldFIP Networks. Polytechnic Institute of Porto, Technical Report HURRAY-TR-9910, March 1999, to appear in the *WIP session proceedings of the Euromicro RTS'99*, June 1999.

17. Tovar, E. and Vasques, F. Engineering Real-Time Applications with P-NET, Technical Report HURRAY-TR-9916, April 1999, to appear in the *Proceedings of the 6th P-NET Conference*, May 1999.

E. Submitted for Publication in Conference Proceedings

18. Tovar, E. and Vasques, F. Factory Communications: on the Configuration of the WorldFIP Bus Arbitrator Table. Polytechnic Institute of Porto, Technical Report HURRAY-TR-9909, March 1999. Submitted to *ETFA'99*.

19. Tovar, E. and Vasques, F. Using WorldFIP Networks to Support Periodic and Aperiodic Traffic. Polytechnic Institute of Porto, Technical Report HURRAY-TR-9917, April 1999, submitted to *IECON'99*.

20. Tovar, E., Vasques, F. and Cardoso, A. Guaranteeing Timing Requirements Using P-NET Fieldbus Networks. Submitted to *ETFA'99*.