Distributed computing for the factory-floor: a real-time approach using WorldFIP networks

Eduardo Tovar, Francisco Vasques

Abstract

Field communication systems (fieldbuses) are widely used as the communication support for distributed computer-controlled systems (DCCS) within all sort of process control and manufacturing applications. 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, within which the most relevant is the decentralisation and distribution of the processing power over the field. A widely used fieldbus is the WorldFIP, which is normalised as European standard EN 50170. Using WorldFIP to support DCCS, an important issue is "how to guarantee the timing requirements of the real-time traffic?" WorldFIP has very interesting mechanisms to schedule data transfers, since it explicitly distinguishes periodic and aperiodic traffic. In this paper, we describe how WorldFIP handles these two types of traffic, and more importantly, we provide a comprehensive analysis on how to guarantee the timing requirements of the real-time traffic.

Keywords: Factory-floor communications; Hard real-time systems; WorldFIP networks; Distributed computer-controlled systems

1. Introduction

In the past decade, manufacturing schemes have changed dramatically. In particular, the Computer Integrated Manufacturing [1] concept has been stressed as a means to achieve greater production competitiveness [2]. The driving forces behind the changes also resulted from the increased development and utilisation of new technologies that make massive use of microprocessor-based equipment. Integration implies that 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 [3]. However, the use of communication networks at the field level is a much more recent trend [4]. Indeed, only more recently network interfaces become cost-effective for the interconnection of devices such as sensors and actuators, 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 and numerically-controlled machines).

The field level includes process-relevant field devices, such as sensors and actuators. The process control level is hierarchically located above the field level, and directly influences it in the form of control signals. If the control is computer-based, the process control level uses data received from sensors to compute new commands, which are then transmitted to the actuators.

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 (sensors/actuators) and the man–machine interfaces. The sensors and actuators are connected to the computer system via point-to-point links. Fig. 1(a) illustrates such kind of architecture.

There are, however, several advantages in using a field level communication network as the replacement for the point-to-point links between the sensors/actuators and the computer system. As it can be depicted from Fig. 1(b), a cost reduction can be obtained by replacing a significant part of the wiring by a field level communication network. 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.

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 relies on the distribution of the control algorithms. In a centralised computer-controlled architecture, all the control algorithms are implemented in a single computer system. In a decentralised computer-controlled architecture, the control algorithms are also centralised in a single computer system (now also a network node), even if some processing tasks (signal conditioning or pre-processing operations) may be executed in the network nodes that interface to the sensors and

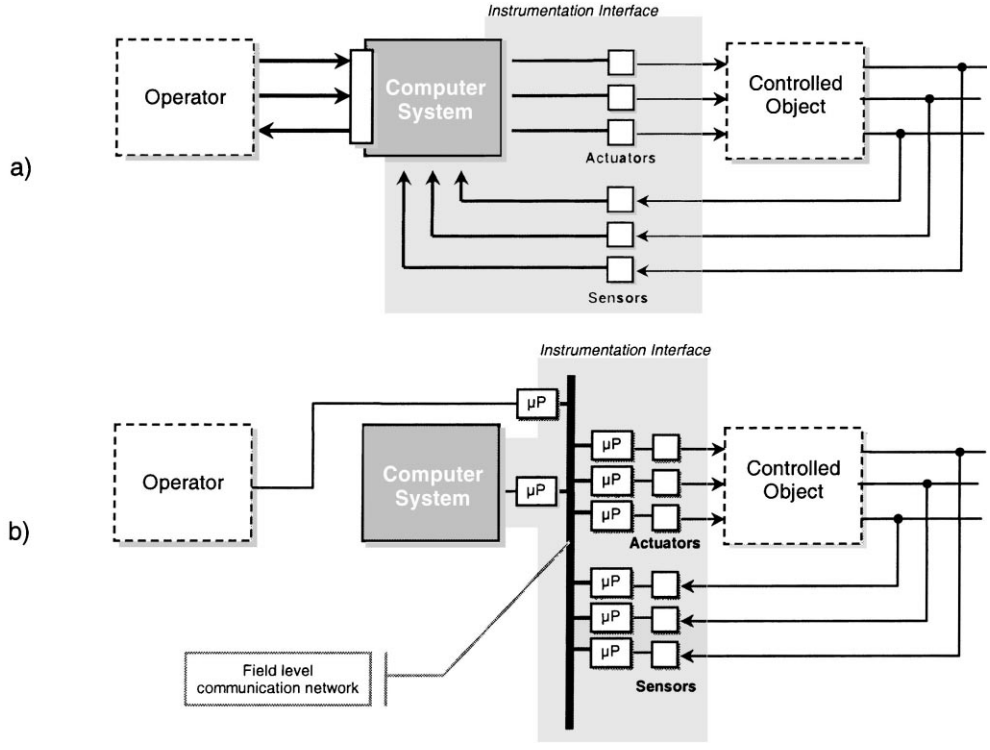

Fig. 1. This figure highlights the main advantage of a decentralised computer-controlled architecture (b) when compared to a centralised computer-controlled architecture (a).

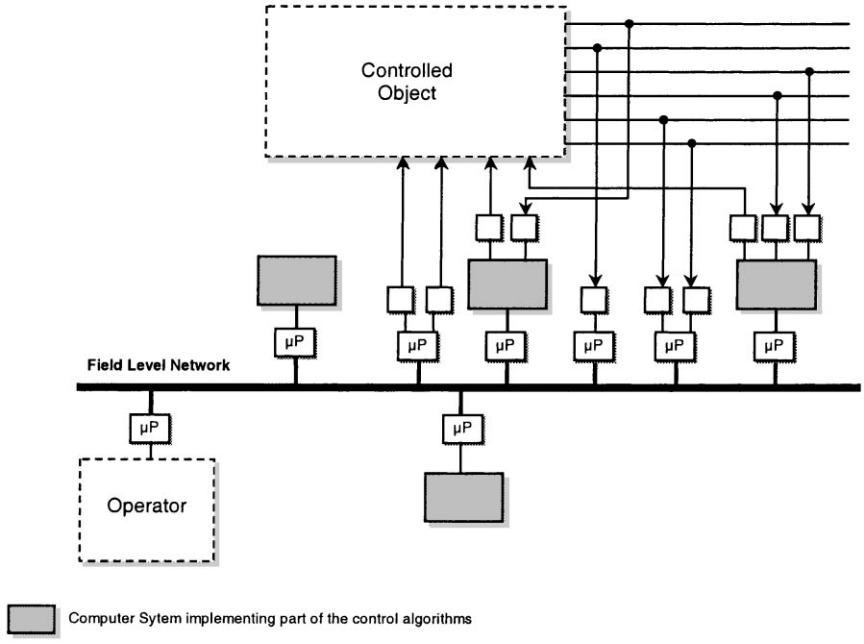

Fig. 2. This figure represents a distributed computer-controlled architecture.

actuators. Contrarily, in a distributed computer-controlled architecture, the tasks of the control algorithms may be distributed throughout several computing nodes. Fig. 2 depicts the organisation of a distributed computer-controlled architecture.

The ability to support distributed control algorithms is another advantage of using a field level network. This eases the design of computer-controlled systems where decentralisation of control and measurement tasks, as well as the number of microprocessor-controlled devices is ceaselessly increasing.

Field level networks aimed at the interconnection of sensors, actuators and controllers are commonly known as fieldbus networks. In the past, the fieldbus scope was dominated by vendor specific solutions, which were mostly restricted to specific application areas. Moreover, concepts behind each proposed network were highly dependent on the manufacturer of the automation system, each one with different technical implementations and also claiming to fulfil different application requirements, or the same requirements with different technical solutions [5]. More recently, standardised fieldbuses supporting the open system concept, thus vendor independent, started to be commonly used. Particular relevance must be given to the European Standard EN 50170 [6], which encompasses three widely used fieldbuses: P-NET [7], PROFIBUS [8] and WorldFIP [9]. All these fieldbus protocols aim at the support of distributed computer-controlled systems (DCCS).

This paper addresses the ability of WorldFIP to cope with the real-time requirements of DCCS. Typically, DCCS include process variables that must be transferred between network devices, either in a periodic or in an aperiodic basis. The WorldFIP protocol is designed to support both types of traffic. A potential large leap towards its use in DCCS relies on the evaluation of its temporal behaviour. Particularly, an important problem associated with the WorldFIP fieldbus is its inability to guarantee the timing requirements associated to the aperiodic traffic. As it will be seen, real-time requirements for the periodic traffic can be easily guaranteed by the WorldFIP protocol. However, for the aperiodic traffic more complex analysis must be made to evaluate the message's response time, in order to guarantee its real-time requirements. The main focus of this paper will be in the analysis of the aperiodic message's response time.

The remainder of this paper is organised as follows. Section 2 describes the most important concepts of the

WorldFIP networks. In Section 3 we evaluate the worst-case response time for the aperiodic traffic as a function of the periodic traffic schedule, thus performing an integrated analysis of both types of traffic. In Section 4 we present a numerical example, which highlights the relevance of the proposed methodology. Finally, in Section 5 some conclusions are drawn.

2. Basic analysis of the WorldFIP protocol

A WorldFIP network interconnects stations with two types of functionality: 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 paper, we address WorldFIP networks supporting only the exchange of identified variables, since they are the basis of WorldFIP real-time services. The exchange of messages, which is used to support manufacturing message services (MMS) [10], is out of the scope of this paper.

2.1. Producer/distributor/consumer concept

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

In order to manage transactions associated to a single variable, a unique identifier is associated to 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 AL services generate activity on the bus.

Produced and consumed buffers can be also 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 sends an indication of transmission to the AL (*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*.

2.2. Buffer transfer timings

A buffer transfer implies the transmission of a pair of frames: *ID_DAT*, followed by *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, added to the time needed to transmit the *RP_DAT* frame, and finally added to twice the turnaround time (t_r). The turnaround time is the time elapsed between any two consecutive frames.

Every transmitted frame is encapsulated with control information from the physical layer (PL). Specifically, an *ID_DAT* frame has always 8 bytes (corresponding to a 2 bytes identifier plus 6 bytes of control information), whereas a *RP_DAT* frame has also 6 bytes of control information plus up to 128 bytes of useful data. The duration of a message transaction is

$$C = \frac{\text{len}(ID\_DAT) + \text{len}(RP\_DAT)}{\text{bps}} + 2 \times t_r \qquad (1)$$

where *bps* stands for the network data rate and $\text{len}(\langle frame \rangle)$ is the length, in bits, of *frame* $\langle frame \rangle$.

For instance, assuming a variable with a 4 bytes data field, if $t_r = 20\ \mu s$[1] and the network data rate is

[1] The turnaround time (t_r) is imposed [9] to be within the interval $10 \times (1/bps) \leq t_r \leq 70 \times (1/bps)$.

Table 1
Example set of periodic buffer transfers

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

2.5 Mbps then, the duration of an elementary transaction will be $(64 + 80)/2.5 + 2 \times 20 = 97.6$ μs (Eq. (1)).

2.3. Bus arbitrator table

In WorldFIP networks, the *bus arbitrator table* (BAT) regulates the scheduling of all buffer transfers. The BAT imposes the timings of the periodic buffer transfers, and also regulates the aperiodic buffer transfers, as it will be explained in Section 2.4.

Assume a distributed system within which six variables are to be periodically scanned, with scan rates as shown in Table 1. The WorldFIP BAT must cope with these real-time requirements.

Two important parameters are associated with a WorldFIP BAT: the *micro-cycle* (elementary cycle) and the *macro-cycle*. The micro-cycle imposes the maximum rate at which the BA performs a set of scans. Usually, the micro-cycle is set equal to *highest common factor* (HCF) of the required scan periodicities. Using this HCF rule, and for the example of Table 1, the value for the micro-cycle is set to 1 ms. A possible schedule for all the periodic scans is illustrated in Fig. 3, where we consider $C = 97.6$ μs (1) for each elementary transaction.

It is easy to depict that, for this example, the sequence of micro-cycles is repeated after each 12 micro-cycles. This sequence of micro-cycles is said to be a macro-cycle, and its length is given by the *lowest common multiple* (LCM) of the scan periodicities. The HCF/LCM approach for building the WorldFIP BAT has the following properties:

1. The scanning periods of the variables are multiples of the micro-cycle.
2. 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 micro-cycle. All other variables suffer from a slight communication jitter. For instance, concerning variable *F*, the interval between micro-cycles 1 and 7 is $(1-5\times0.098)+5+(3\times0.098) = 5.80$ ms, whereas the interval between micro-cycles 7 and 13 is $(1 - 3 \times 0.098) + 5 + (5 \times 0.098) = 6.20$ ms.
3. The length of the macro-cycle can induce a memory size problem, since the table parameters must be stored in the BA. For instance, if scanning

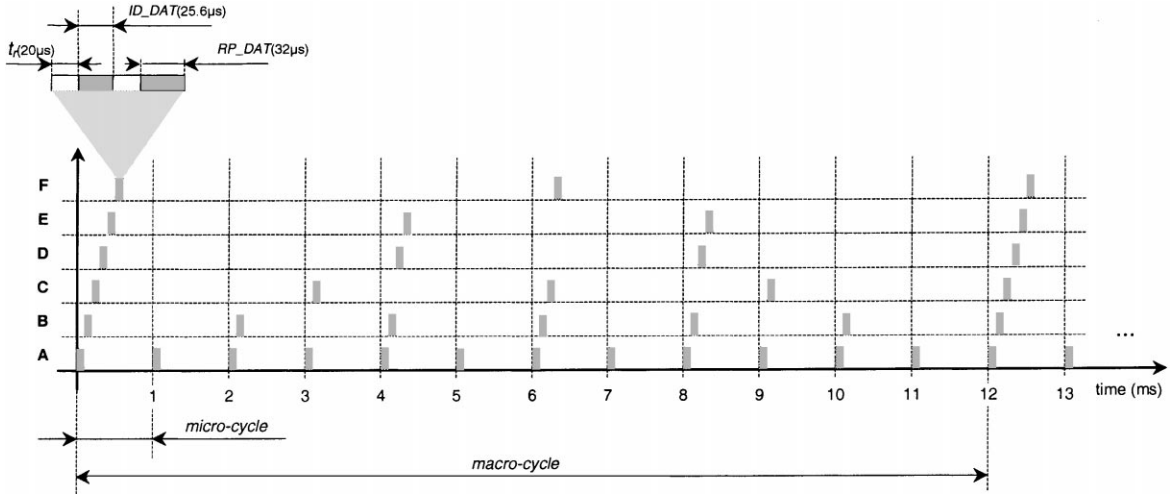

Fig. 3. This figure illustrates a possible schedule for the periodic buffer transfers of Table 1.

periodicities of variables E and F were, respectively, 5 and 7 ms, the length of the macro-cycle would be 420 micro-cycles instead of only 12.

Both the communication jitter and memory size problems have been addressed in the literature. In [11] the authors discuss different methodologies for reducing the BAT size, without penalising the communication jitter. The idea is very simple, and it basically consists on reducing some of the scan periodicities in order to obtain a harmonic pattern. The problem of table size has also been addressed in [12,13], however in a different perspective. In the referred work, the authors discuss an online scheduler (instead of storing the schedule in the BA's memory), which is not directly applicable to the WorldFIP case.

It is also worth mentioning that the schedule shown in Fig. 3 represents a macro-cycle composed of synchronous micro-cycles, that is, for the specific example, each micro-cycle starts exactly 1 ms after the previous one. Within a micro-cycle, the spare time between the end of the last scan for a periodic variable and the end of the micro-cycle can be used by the BA to process aperiodic requests for buffer transfers (see Section 2.4), message transfers and padding identifiers. A WorldFIP BA can also manage asynchronous micro-cycles, not transmitting padding identifiers at the end of the micro-cycle. In such case, a new micro-cycle 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.

2.4. Aperiodic buffer transfers

In a WorldFIP system, not all the variables are to be included in the BAT. Some 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. In the context of this paper, we consider that aperiodic requests use the *urgent* priority level.

The BA handles the aperiodic buffer transfers after processing the periodic traffic in a micro-cycle. The portion of the micro-cycle reserved for the periodic buffer exchanges is denoted as the *periodic window*, whereas the time left after the periodic window is denoted as the *aperiodic window*. The aperiodic buffer transfers take place in three stages (Fig. 4):

1. A station with a pending aperiodic request must wait for its next periodic buffer transfer (say periodic variable X) to notify the BA, setting an aperiodic request bit in the *RP_DAT* frame. The bus arbitrator stores the indication of a yet not identified aperiodic request in a queue of requests for variable transfers. At the end of this interval, the BA is aware of a pending request in the station that produces periodic variable X.
2. In a subsequent aperiodic window, the BA asks the producer of the variable X (*ID_RQ* frame) to transmit the list of its pending aperiodic requests. The producer of X responds with a *RP_RQ* (list of

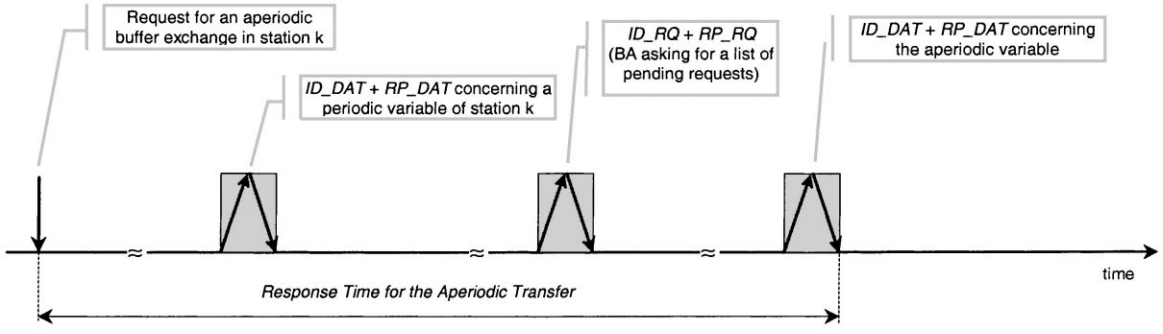

Fig. 4. Timings for transactions associated with the processing of one aperiodic variable.

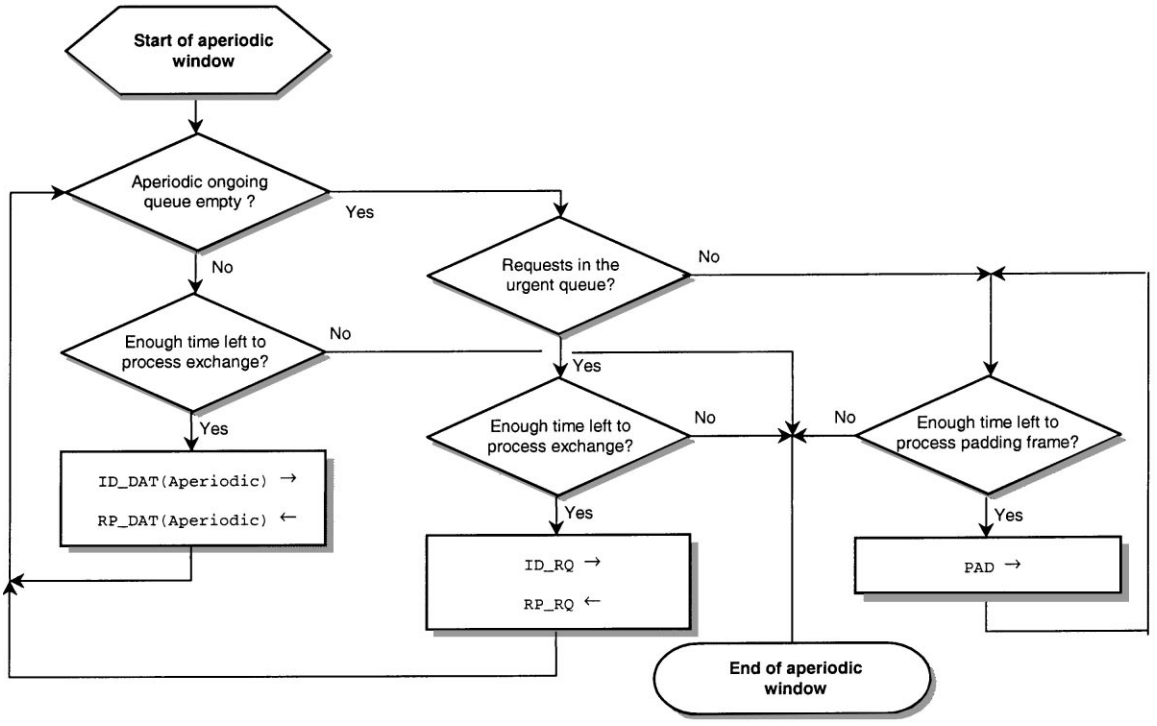

Fig. 5. This figure illustrates the (urgent) aperiodic buffer transfer mechanisms of the WorldFIP protocol.

identifiers) frame. This list is placed in another BA's queue, the *ongoing aperiodic queue*. At the end of this interval the BA knows which are the requested identifiers.

3. Finally, the BA processes requests for aperiodic transfers that are stored in its ongoing aperiodic queue. For each transfer, the BA uses the same mechanism as that used for the periodic buffer transfers (*ID_DAT* followed by *RP_DAT*).

It is 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. Finally, 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 it can be depicted from Fig. 5, traffic concerning the aperiodic buffer transfers (transactions *ID_RQ/RP_RQ* or *ID_DAT/RP_DAT*) can only be carried out if there is time left in a specific micro-cycle to completely process them.

As previously mentioned, in this paper we only consider the transfer of identified variables, for both periodic and aperiodic buffer transfers. Concerning the aperiodic requests, we consider that all these requests use *urgent* priority level.

3. Response time analysis of WorldFIP traffic

3.1. Model for the WorldFIP buffer transfers

Consider the following model for the WorldFIP buffer transfers (periodic and aperiodic). Assume a system with *np* periodic variables (Vp_i, $i = 1, \ldots, \mathrm{np}$) and *na* aperiodic variables (Va_j, $j = 1, \ldots, \mathrm{na}$). Each periodic variable is characterised as

$$\mathrm{Vp}_i = (\mathrm{Tp}_i, \mathrm{Cp}_i) \tag{2}$$

where Tp_i corresponds to the periodicity of Vp_i and Cp_i is the length of the transaction corresponding to the buffer transfer of Vp_i (as given by Eq. (1)). Each

aperiodic variable Va_j is characterised as

$$Va_j = (Ta_j) \tag{3}$$

where Ta_j corresponds to the minimum inter-arrival time between two consecutive requests of Va_j.

3.2. Building the WorldFIP BAT

The real-time requirements of the WorldFIP periodic traffic can be easily guaranteed since the BAT implements a static schedule for the periodic buffer transfers. In this section, we present three different algorithms to build the BAT schedule: two adapted algorithms from the well-known Rate Monotonic (RM) and Earliest Deadline First (EDF) algorithms [14], respectively, and also a third algorithm which we denote as the Deferred Release (DR) algorithm.

Building the BAT schedule with either the RM or the EDF algorithms has several advantages, since it allows for the use of well-known schedulability analysis theory.

Considering the RM scheduling algorithm, variables are scheduled according to their periodicity: the variable with the smallest periodicity will have the highest priority. If several variables have the same periodicity (priority tie), the higher priority is given to the variable with the smaller identifier. In Appendix A, a detailed algorithm for building the BAT using the RM algorithm is presented. The algorithm indicates whether all traffic is schedulable or not (line 22). In the algorithm, the vector load[ ] is used to store the load in each micro-cycle as the traffic is scheduled. It also assumes that the array Vp[,] is ordered from the variable with the shortest period (Vp[1,]) to the variable with the longest period (Vp[np,]). The RM approach for building the WorldFIP BAT has some aspects that are worthwhile to be addressed. The following example highlights some of these aspects.

Table 2
BAT (using RM) for example of Table 1[a]

	Micro-cycle											
	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

[a] $Cp_i = 184$ µs.

Consider the set of periodic buffer transfers presented in Table 1, where the network data rate is 1 Mbps instead of 2.5 Mbps, which results in longer messages: $Cp_i = (64 + 80)/1 + 2 \times 20 = 184$ µs (Table 2). Note that each micro-cycle is able to schedule up to five periodic buffer transfers (Fig. 6). As a consequence, there is the *insertion of idle time* in the resulting schedule, since the remaining time in the first micro-cycle (1 ms $-5 \times 184 = 80$ µs) is not enough to process a sixth transaction.

An alternative to the use of the RM algorithm to build the BAT schedule is the EDF algorithm. When

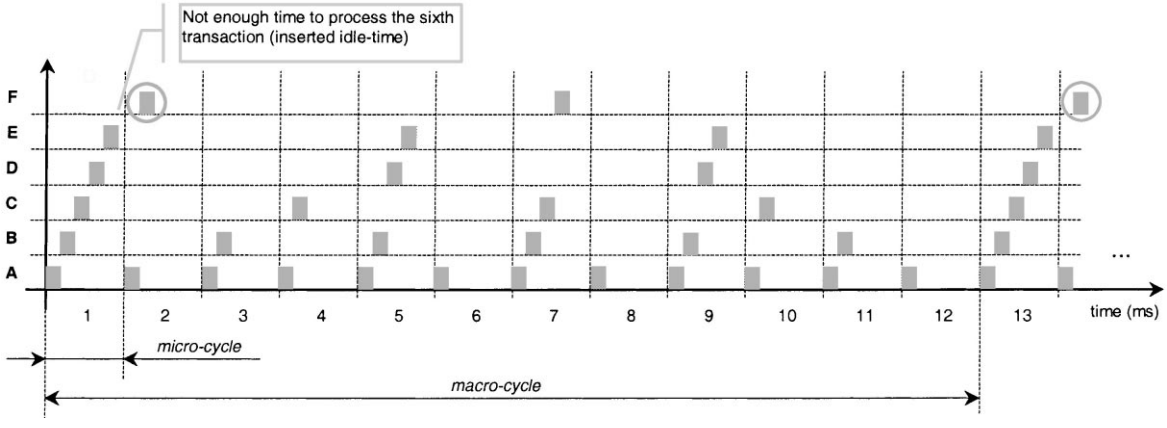

Fig. 6. Schedule (RM approach) for the set example of Table 2.

Table 3
Example set of periodic buffer transfers (Cp = 0.30 ms)

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

compared to the RM scheduling, one of the known advantages of the EDF scheduling is that it allows a better utilisation [14]. Considering the EDF algorithm, variables are scheduled according to the earliest deadline (we assume deadlines equal to periods and thus the variable with the earliest deadline is the one with the earliest end of period). If several variables have the same deadline (priority tie), the higher priority is given to the variable with the earliest request.

In Appendix B, a detailed algorithm for building the BAT with the EDF algorithm is presented. In the algorithm, the array disp[,] is used to store in disp[i,1] whether there is any pending request for variable i, in disp[i,2] the related deadline (multiple of the micro-cycle), and in disp[i,3] the micro-cycle at which the request is made.

The increased utilisation allowed by the EDF scheduling algorithm is emphasised, considering the set of periodic buffer transfers shown in Table 3 (utilisation equal to 90%).

Considering the RM algorithm, the first request for Vp_F would miss its deadline (Table 4). In fact, there is no empty slot for variable Vp_F in the first three micro-cycles, since each micro-cycle is only able to schedule up to three buffer transfers.

Considering the EDF algorithm, the first request for Vp_F does not miss its deadline (Table 5), since the second request of variable Vp_C is scheduled only in the fourth micro-cycle. Thus, by increasing the communication jitter, the set of periodic buffer transfers shown in Table 3 is now schedulable.

Table 4
BAT (using RM) for example of Table 3

	Micro-cycle					
	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*	0	0	0	0	1

Table 5
BAT (using EDF) for example of Table 3

	Micro-cycle					
	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

A particular characteristic of using EDF to build the WorldFIP BAT is that, in spite of having a utilisation smaller than 100%, there is not any available time left on the BAT to schedule another aperiodic buffer transfer. This is due to the inserted idle-time in each micro-cycle, which has the length $1000 - 3 \times 300 = 100$ μs.

With RM and EDF algorithms, besides the inserted idle time aspect, variables that are scheduled in subsequent micro-cycles will suffer from an increased communication jitter. For instance, the reduction of the jitter associated to the transfer of the low priority variables can be easily achieved by manipulation of the release time of such variables.

In Appendix C, a detailed algorithm for building the BAT using a Deferred Release (DR) approach is presented. In this algorithm, each variable is scheduled according to its exact periodicity (in terms of number of micro-cycles). That is, either the variable is scheduled in the correct micro-cycle, or it is not scheduled (line 25). The release time of each periodic variable is chosen in order to minimise the maximum length of the periodic windows in the resulting BAT schedule (line 18).

For instance, the resulting schedule for the example of Table 1 ($Cp_i = 184$ μs) would be as presented in Table 6 and Fig. 7 (compared to Table 2 and Fig. 6 for the RM case, results in a clear reduction of the communication jitter).

3.3. Response time characterisation of an aperiodic buffer transfer

The response time of an aperiodic buffer transfer is a function of the periodic traffic pattern, since the

Table 6
BAT (using DR) for example of Table 1[a]

	Micro-cycle											
	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]	0	1	0	0	0	1	0	0	0	1	0	0
bat[E, cycle]	0	0	0	1	0	0	0	1	0	0	0	1
bat[F, cycle]	0	1	0	0	0	0	0	1	0	0	0	0

[a] $\mathrm{Cp}_i = 184\ \mu\mathrm{s}$.

length of each aperiodic window is a function of the generated BAT schedule. Thus, the exact characterisation of the periodic traffic pattern (as defined in the BAT schedule) is fundamental for the response time analysis of the aperiodic traffic.

We define the response time of an aperiodic buffer transfer as the time interval between placing, at time instant t_0, the request in the local urgent queue and the completion of the buffer transfer concerning the aperiodic variable Va_j in a BA's aperiodic window (Fig. 4). This response time includes the following three components:

1. The time elapsed between t_0 and the time instant when the requesting station (station k) is able to indicate the BA (via an *RP_DAT* frame, 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 (at the end of this interval, the BA is aware of a pending request in station k, without being able to identify it).
2. The time interval during which the request indication stays in the BA's urgent queue till the related *ID_RQ/RP_RQ* pair of frames is processed in an aperiodic window (at the end of this interval, the BA knows which is the pending request in station k).
3. The time interval during which the request for variable Va_j stays in the BA's ongoing aperiodic queue till the related *ID_DAT_Ap/RP_DAT* pair of frames is processed in an aperiodic window.

These two last components (2 and 3) can be grouped as the BA's *processing interval* of an aperiodic buffer transfer. All these components must be upper bounded, in order to have a bounded response time for an aperiodic buffer transfer.

3.4. Upper bound for the dead interval

The upper bound for the dead interval in a station k is related to the smallest scanning period of a periodic variable, produced in that station. It is important to note that a periodic variable (Vp_i) is not polled at regular intervals due to the communication jitter inherent to the BAT schedule. Therefore, the upper bound for the dead interval in a station k is

$$\sigma^k = \mathrm{Tp}_j + J_{\mathrm{Vp}_j} + \mathrm{Cp}_j,$$
$$\text{with } \mathrm{Vp}_j : \mathrm{Tp}_j = \min_{\mathrm{Vp}_i \text{ produced in } k} \{\mathrm{Tp}_i\} \tag{4}$$

where J_{Vp_i} is the maximum communication jitter of that Vp_j. We consider that a local aperiodic request is processed (setting the request bit in the *RP_DAT*

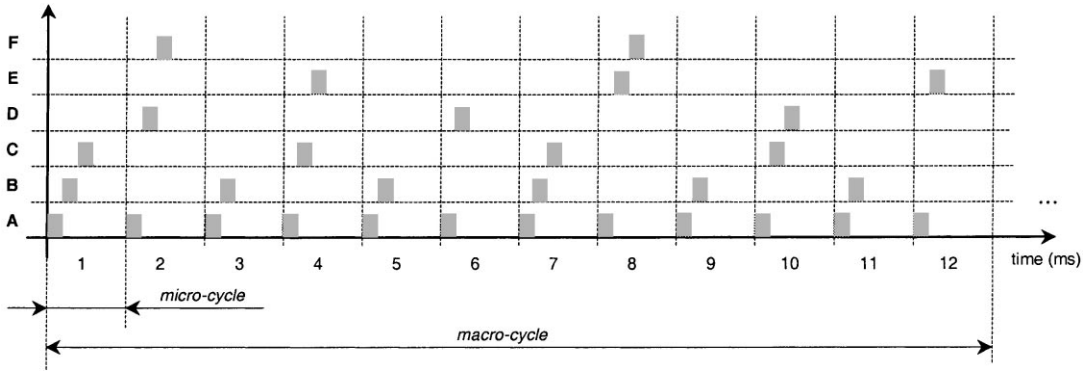

Fig. 7. Schedule (DR approach) for the set example of Table 6.

frame) only if it arrives before the start of the related *ID_DAT*. Hence, the term Cp_j is included in Eq. (4).

For instance, concerning the set of periodic variables of Table 1, if variable *F* is the only produced variable at station *k*, then $\sigma^k = 6 + 0.20 + 0.098 = 6.30$ ms (see Section 2.3 for the evaluation of the jitter).

In Appendix D we describe a detailed algorithm for the evaluation of the communication jitter associated to a periodic variable, which performs the analysis of the generated BAT schedule. This algorithm is the basis for the evaluation of the dead interval in a specific station *k*.

3.5. Upper bound for the processing interval

We assume that all the aperiodic traffic has a minimum inter-arrival time between requests, which is greater than its worst-case response time. Therefore, no 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 variables in the network (Section 3.1).

The upper bound for the BA's processing interval occurs when both the periodic and aperiodic traffic are simultaneously at their maximum load.

We consider that for each aperiodic variable a request for identification must be made, and thus the network load for the aperiodic traffic is maximised. As a consequence, to process the maximum number of aperiodic variables, 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 (see Fig. 5). That is, the number of transactions to be processed is $2 \times na$.

If all the aperiodic variables have a similar length, the length of the longest transaction in an aperiodic window may be defined as

$$\mathrm{Ca}^* = \max_{i=1,..,\mathrm{na}} \left\{ \frac{\mathrm{len}(ID\_dat) + \mathrm{len}(RP\_DAT)}{\mathrm{bps}} + 2 \times t_\mathrm{r}, \times \frac{\mathrm{len}(ID\_RQ) + \mathrm{len}(RP\_RQ)}{\mathrm{bps}} + 2 \times t_\mathrm{r} \right\} \quad (5)$$

We define the BA's processing interval of *na* consecutive aperiodic requests, as an *aperiodic busy interval* (ABI), since all aperiodic windows within this interval are used to process aperiodic traffic. We denote $N_{\mathrm{ABI}}(j)$ as the number of micro-cycles during an ABI starting at micro-cycle *j* and len_abi(*j*) as the length of such ABI.

The upper bound for the length of the ABI occurs when the ABI is starting at the critical micro-cycle, which is defined as the micro-cycle that leads to the *longest* ABI.

It is obvious that using either the RM or the EDF scheduling policies to build the BAT, the critical micro-cycle is the first one of the macro-cycle. However, for the case of different approaches to build the BAT schedule (e.g. the presented DR algorithm), the critical micro-cycle may not be the first one. For these other scheduling algorithms, it is necessary to determine which is the critical micro-cycle, in order to evaluate the length of the longest aperiodic busy interval.

The length of the longest ABI is evaluated as follows. Firstly, we determine recursively the value for each $N_{\mathrm{ABI}}(j)$. After, we evaluate the length of each ABI (len_abi(*j*)), considering that during one ABI there are only $(N_{\mathrm{ABI}}(j)-1)$ micro-cycles which are completely utilised.

The length of the aperiodic window in the *l*th cycle ($l = 1, \ldots, N, N+1, \ldots$) is

$$\mathrm{aw}(l) = \mu\mathrm{Cy} - \sum_{i=1}^{\mathrm{np}} (\mathrm{bat}[i, l^*] \times \mathrm{Cp}_i) \quad (6)$$

where $\mathrm{bat}[i, l^*]$ is a matrix representing the schedule of the periodic traffic and $l^* = [(l-1)\mathrm{mod}N] + 1$. The use of variable l^* stresses that cycle $l = 1$ is equivalent to cycle $l = N + 1$, as a macro-cycle has only *N* micro-cycles. Therefore, the number of aperiodic transactions that fit in the *l*th aperiodic window is

$$\mathrm{nap}(l) = \left\lfloor \frac{\mathrm{aw}(l^*)}{\mathrm{Ca}^*} \right\rfloor \quad (7)$$

The number of micro-cycles during an ABI starting at micro-cycle *j* ($N_{\mathrm{ABI}}(j)$) is then

$$N_{\mathrm{ABI}}(j) = \min\{\Psi\}, \quad \text{with } \Psi = \sum_{i=j}^{(j-1)+N_{\mathrm{ABI}}} \mathrm{nap}(l^*) \wedge \Psi \geq 2 \times na \quad (8)$$

This value must be recursively evaluated for each micro-cycle *j*, since the number of aperiodic transactions that fit in an aperiodic window depends

on the micro-cycle that is being considered. The recursion stops when the number of available ''slots'' (each ''slot'' with the length of Ca^*) is at least $2 \times na$.

Knowing the number of micro-cycles during an ABI, the length of each aperiodic busy interval (len_abi(j)) may be evaluated as follows:

$$\begin{aligned}\text{len\_abi}(j) = {} & (N_{\text{ABI}}(j)-1)\times \mu\text{Cy} \\ & + \sum_{i=1}^{\text{np}} (\text{bat}[i, ((j-1)+N_{\text{ABI}}(j))^*] \times \text{Cp}_i) \\ & + \ 2 \times \text{na} - \sum_{l=j}^{(j-1)+N_{\text{ABI}}(j)-1} \text{nap}(l^*) \Bigg) \times \text{Ca}^* \end{aligned} \tag{9}$$

where $(N_{\text{ABI}}(j)-1) \times \mu\text{Cy}$ indicates the length of the completely utilised micro-cycles, $\sum_{i=1,..,\text{np}} (\text{bat}[i, (j-1+N_{\text{ABI}}(j))^*] \times \text{Cp}_i)$ indicates the length of the periodic window in the last micro-cycle of the ABI, with $((j-1)+N_{\text{ABI}}(j))^* = [(j-1)+N_{\text{ABI}}(j) \ -1) \text{mod} N] + 1$ and $(2 \times \text{na} - \sum_{l=1,..,j+N(j)-2} \text{nap}(l^*)) \times \text{Ca}^*$ indicates how many aperiodic transactions are still pending to be processed on the last micro-cycle of the ABI.

The length of the longest ABI is then

$$\text{len\_abi} = \max_{j \in \{1,..,N\}} \{\text{len\_abi}(j)\} \tag{10}$$

In Appendix E a detailed algorithm for evaluating the length of the longest ABI is provided.

3.6. Worst-case response time for the aperiodic requests

The worst-case response time of an aperiodic buffer transfer at station k is equal to the length of the dead interval of that station plus the length of the longest ABI

$$\text{Ra}^k = \sigma^k + \text{len\_abi} \tag{11}$$

Thus, the minimum inter-arrival time between any two consecutive aperiodic requests of the same aperiodic variable in a station k is

$$\text{Ta}_j \geq \text{Ra}^k = \sigma^k + \text{len\_abi} \tag{12}$$

Inequality (Eq. (12)) presents a pre-run-time schedulability test, which can be used to validate the real-time requirements of WorldFIP applications.

Table 7
Example set of periodic buffer transfers

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

4. Numerical example

Consider that a WorldFIP network with six periodic variables (as defined in Table 7) must also support seven aperiodic buffer exchanges. Assume that the length of the each $\text{Vp}_i, \forall_i$ is $\text{Cp}_i = \text{Cp} = 200$ μs, and that the length of an aperiodic buffer transfer is $\text{Ca}^* = 100$ μs. The value of the micro-cycle is 1 ms.

If the BAT is implemented as shown in Table 8 (enforcing micro-cycles 4 and 6 to be the most loaded ones), the longest ABI will not be at the beginning of the macro-cycle. Thus, the worst-case response time for an aperiodic transfer will occur when the request arrives to the BA at the beginning of the longest ABI and not at the beginning of the macro-cycle.

To evaluate the worst-case response time of an aperiodic transfer, we will proceed as defined in Section 3.5. The length of each aperiodic window during the macro-cycle (6) and the number of aperiodic transactions that fit in each aperiodic window (7) are, respectively given in Table 9.

Table 8
BAT for the numerical example

	Micro-cycle					
	1	2	3	4	5	6
bat[A, cycle]	1	1	1	1	1	1
bat[B, cycle]	0	1	0	1	0	1
bat[C, cycle]	0	1	0	1	0	1
bat[D, cycle]	1	0	0	1	0	0
bat[E, cycle]	0	0	1	0	0	1
bat[F, cycle]	0	0	1	0	0	0

Table 9
Aperiodic windows for example of Table 8

Micro-cycles	1	2	3	4	5	6
aw(l) (μs)	600	400	400	200	800	200
nap(l)	6	4	4	2	8	2

Table 10
Characterisation of the aperiodic busy interval for each micro-cycle j

Micro-cycle j	1	2	3	4	5	6
$N_{\mathrm{ABI}}(j)$	3	4	3	4	3	4
len_abi (j) (ms)	3.0	3.6	3.0	3.6	2.8	3.8

Using the recurrence relationship given by Eq. (8), we may now evaluate the number of micro-cycles during an ABI, for each micro-cycle j. For the case of an ABI starting at the second micro-cycle ($j = 2$), $N_{\mathrm{ABI}}(2)$ is evaluated as follows:

$$N_{\mathrm{ABI}}(2) = 1, \qquad \Psi = \sum_{l=2}^{2} \mathrm{nap}(l^*) = 4 \leq 14$$

$$N_{\mathrm{ABI}}(2) = 2, \qquad \Psi = \sum_{l=2}^{3} \mathrm{nap}(l^*) = 4 + 4 < 14$$

$$N_{\mathrm{ABI}}(2) = 3, \qquad \Psi = \sum_{l=2}^{4} \mathrm{nap}(l^*) = 4 + 4 + 2 < 14$$

$$N_{\mathrm{ABI}}(2) = 4,$$

$$\Psi = \sum_{l=2}^{5} \mathrm{nap}(l^*) = 4 + 4 + 2 + 8 \geq 14,$$

and then iteration stops. Thus, $N_{\mathrm{ABI}}(2) = 4$.

For the case of ABIs starting in the other micro-cycles, the values for $N_{\mathrm{ABI}}(j)$ are presented in Table 10. We may now evaluate the length of each ABI. For the case of an ABI starting at the second micro-cycle ($j = 2$), len_abi(2), the three components of Eq. (9) are evaluated as follows:

$$(N_{\mathrm{ABI}}(2) - 1) \times \mu\mathrm{Cy} = 3 \times 1000 = 3000\,\mu\mathrm{s}$$

$$\sum_{i=1}^{6} (\mathrm{bat}[i, (1 + N_{\mathrm{ABI}}(2))^*] \times \mathrm{Cp}_i)$$

$$= \sum_{i=1}^{6} (\mathrm{bat}[i, 5] \times 200) = 200\,\mu\mathrm{s}$$

$$2 \times \mathrm{na} - \sum_{l=2}^{4} \mathrm{nap}(l^*) \Bigg) \times 100$$

$$= (14 - 4 - 4 - 2) \times 100 = 400\,\mu\mathrm{s}$$

Thus, for the ABI starting at the second micro-cycle len_abi(2) = 3000 + 200 + 400 = 3600 µs. For the case of the ABIs starting at the other micro-cycles, the values for len_abi(j) are also presented in Table 10.

Thus, the length of the longest ABI is 3.8 ms and the critical micro-cycle is the sixth one. In Fig. 8, we illustrate the timing behaviour of an aperiodic buffer transfer for both the longest and the smallest ABIs.

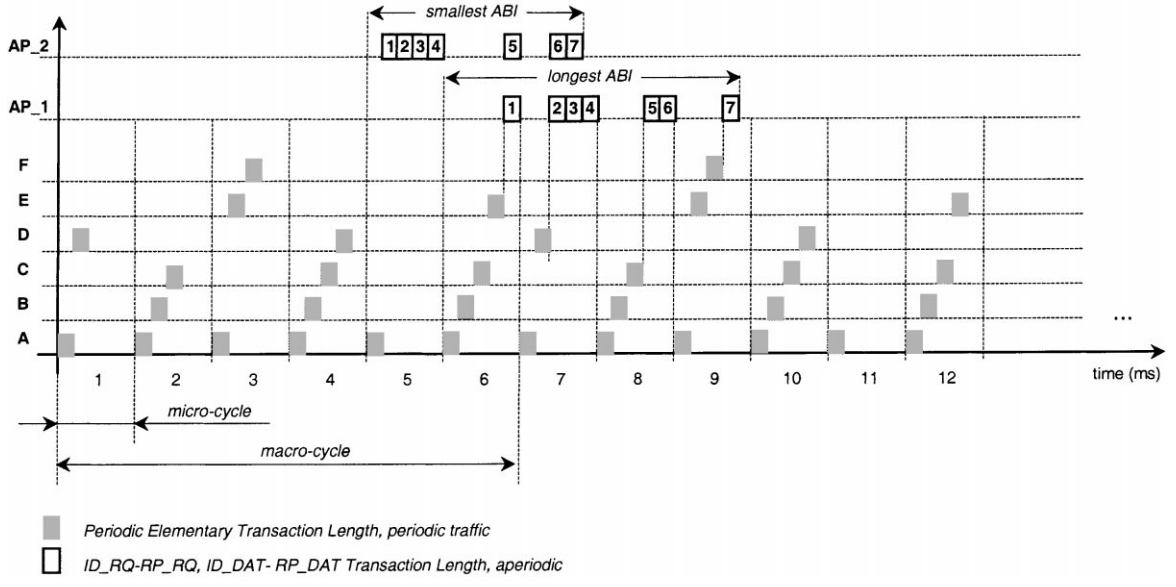

Fig. 8. This figure illustrates the timing behaviour of an aperiodic buffer transfer for the example of Table 10.

Finally, to evaluate the worst-case response time of an aperiodic buffer transfer (11), we must consider the value of the dead interval. As an example, for a request made at a station k, which produces the periodic variable F (assume that F is the only periodic variable produced in station k), the dead interval (4) is 6.30 ms. Therefore, the worst-case response time of an aperiodic variable requested in that station is $\mathrm{Ra}^k = 6.30 + 3.80 = 10.10$ ms.

5. Conclusions

The use of WorldFIP networks to support distributed computer-controlled systems (DCCS), requires a thorough analysis of the communication timing behaviour.

In this paper, we addressed the ability of the WorldFIP to cope with the real-time requirements of DCCS. The WorldFIP protocol is designed to support both periodic and aperiodic traffic. The real-time requirements for the periodic traffic can be easily guaranteed by an adequate setting of the WorldFIP BAT. However, for the aperiodic traffic more complex analysis must be made to evaluate the message's response time, which has been the main focus of this paper.

To our best knowledge, little work has been done on the analysis of the message's response time in WorldFIP networks. Both [15] and [16] address the issue of finding worst-case response times for aperiodic requests in WorldFIP networks. However, those works are based on some restrictive assumptions:

1. In [15] the authors consider that the aperiodic traffic is transferred only after all the periodic traffic in a macro-cycle, which leads to a very pessimistic result.
2. In [16] the authors consider equal lengths for the periodic windows, since they do not discuss the BAT construction and thus they cannot evaluate the length of the periodic window in each micro-cycle. Thus, the overall results are also very pessimistic.

More importantly, none of the analyses considered that at the end of each micro-cycle some transactions would simply not fit. They did also not take into account the communication jitter in the evaluation of the dead interval.

We significantly improved those results by approaching the analysis for the aperiodic traffic in an integrated manner with the methodologies used to build the BAT. Thus, we reduced the pessimism considering the actual length of the periodic window in each micro-cycle. We also considered the effect of both the communication jitter and the inserted idle time.

Acknowledgements

This work was partially supported by ISEP, FLAD, DEMEGI/FEUP and FCT.

Appendix A.

A.1. Rate monotonic (RM) algorithm for building the BAT

```
------------------------------------------------------------------------
- Rate Monotonic for Building the BAT
------------------------------------------------------------------------
function rm_bat;
input:   np       /* number of periodic variables */
         Vp[i,j]  /* array containing the periodicity and length of the */
                  /* variables ORDERED by periodicities, with */
                  /* Vp[i,1]=Tpi, Vp[i,2]=Cpi, */
                  /* i ranging from 1 to np and j ranging from 1 to 2 */
         µCy      /* value of the micro-cycle */
         N        /* number of micro-cycles in the macro-cycle */
output:
         bat[i,cycle]   /* i ranging from 1 to np */
                        /* cycle ranging from 1 to N */
begin
   1:    for i = 1 to np do
   2:       cycle = 1;
   3:       repeat
   4:          if load[cycle] + Vp[i,2] <= µCy then
   5:             bat[i,cycle] = 1;
   6:             load[cycle] = load[cycle] + 1;
   7:             cycle = cycle + (Vp[i,1] div µCy)
   8:          else;
   9:             cycle1 = cycle1 + 1;
   10:            ctrl = FALSE;
   11:            repeat
   12:               if load[cycle1] + Vp[i,2] <= µCy then
   13:                  ctrl = TRUE
   14:               end if;
   15:            until (ctrl = TRUE) or (cycle1 >= (cycle + (Vp[i,1]
   16:                                 div µCy)));
   17:            if cycle1 >= (cycle + (Vp[i,1] div µCy)) then
   18:               bat[i,cycle1] = 1;
   19:               load[cycle1] = load[cycle1] + 1;
   20:               cycle = cycle + (Vp[i,1] div µCy)
   21:            else
   22:               /* MARK Vpi NOT SCHEDULABLE with RM Algorithm */
   23:               cycle = cycle + (Vp[i,1] div µCy)
   24:            end if
   25:         end if
   26:      until cycle > N
   27:   end for
return bat;
------------------------------------------------------------------------
```

A.2. Earliest deadline (EDF) algorithm for building the BAT

```
-------------------------------------------------------------------------
- Earliest Deadline First for Building the BAT
-------------------------------------------------------------------------
function edf_bat;
input:      np        /* number of periodic variables */
            Vp[i,j]   /* array containing the periodicity and length of the */
                      /* variables, with Vp[i,1]=Tpi, Vp[i,2]=Cpi,*/
                      /* i ranging from 1 to np and j ranging from 1 to 2 */
            μCy          /* value of the micro-cycle */
            N         /* number of micro-cycles in the macro-cycle */
output:
            bat[i,cycle]   /* i ranging from 1 to np */
                           /* cycle ranging from 1 to N */
begin
      1:    cycle = 0;
      2:    repeat
      3:       /* determine request generated in each micro-cycle */
      4:       for i = 1 to np do
      5:          if cycle mod Vp[i,1] = 0 then
      6:             disp[i,1] = 1;
      7:             disp[i,2] = Vp[i,1] + cycle;
      8:             disp[i,3] = cycle
      9:          else
      10:                /* MARK variable Vpi NOT SCHEDULABLE */
      11:         end if
      12:      end for;
      13:
      14:         /* Schedule Variables in Current Micro-cycle */
      15:         load_full = FALSE;
      16:         repeat
      17:            no_rq = TRUE;
      18:            earliest = MAXINT;
      19:            var_chosen = 0;           /* no variable chosen */
      20:
      21:            /* Chose Earliest Deadline from Pending Requests */
      22:            for i = 1 to np do
      23:               if disp[i,1] = 1 then   /* if there is request */
      24:                  no_rq = FALSE;
      25:                  if disp[i,2] <= earliest then
      26:                     if var_chosen = 0 then
      27:                        earliest = disp[i,2];
      28:                        var_chosen = i
      29:                     else
      30:                        /* decide earliest request from two */
      31:                        /* with equal deadlines */
      32:                        if disp[i,3] < disp[var_chosen,3] then
      33:                           earliest = disp[i,2];
      34:                           var_chosen = i
      35:                        end if
      36:                     end if
      37:                  end if
      38:               end if
      39:            end for;
      40:
      41:            /* Verify the Load in the Current Micro-cycle */
      42:            if load[cycle + 1] + Vp[i,2] <= μCy then
      43:               bat[var_chosen, cycle + 1] = 1
      44:               load[cycle + 1] = load[cycle + 1] + Vp[var_chosen,2];
```

```
45:                disp[var_chosen,1] = 0;
46:                disp[var_chosen,2] = 0;
47:                disp[var_chosen,3] = 0;
48:             else
49:                load_full = TRUE
50:             end if;
51:          until (load_full = TRUE) or (no_rq = TRUE)
52:
53:          cycle = cycle + 1;
54:       until cycle = N;
return bat;
-------------------------------------------------------------------------
```

A.3. Deferred release (DR) algorithm for building the BAT

```
-------------------------------------------------------------------------
- Deferred Release (DR) for Building the BAT
-------------------------------------------------------------------------
function dr_bat;
input:      np        /* number of periodic variables */
            Vp[i,j]   /* array containing the periodicity and length of the */
                      /* variables ORDERED by periodicities, with */
                      /* Vp[i,1]=Tpi, Vp[i,2]=Cpi, */
                      /* i ranging from 1 to np and j ranging from 1 to 2 */
            μCy       /* value of the micro-cycle */
            N         /* number of micro-cycles in the macro-cycle */
output:
            bat[i,cycle]   /* i ranging from 1 to np */
                           /* cycle ranging from 1 to N */
begin
      1:    for i = 1 to np do
      2:       min_load = np × max(Vp[i,2]);
      3:       for cycle = 1 to (Vp[i,1] div μCy) do
      4:          cycle1 = cycle;
      5:          max_load = 0;
      6:          repeat
      7:             if load[cycle1] > max_load then
      8:                max_load = load[cycle1]
      9:             end if;
      10:            cycle1 = cycle1 + (Vp[i,1] div μCy)
      11:         until cycle1 > N;
      12:         if max_load < min_load then
      13:            cycle_min = cycle;
      14:            min_load = max_load;
      15:         end if;
      16:      end for;
      17:
      18:      cycle = cycle_min;
      19:      repeat
      20:         if (load[cycle] + Vp[i,2]) < μCy then
      21:            load[cycle] = load[cycle] + Vp[i,2];
      22:            bat[i,cycle] = 1;
      23:            cycle = cycle + (Vp[i,1] div μCy)
      24:         else
      25:            /* MARK Vpi NOT SCHEDULABLE with DR Algorithm */
      26:            exit
      27:         end if;
      28:      until cycle > N;
return bat;
-------------------------------------------------------------------------
```

A.4. Algorithm for the evaluation of the communication jitter

```
------------------------------------------------------------------------
- evaluation of the maximum communication jitter of a periodic variable
------------------------------------------------------------------------
function NR;
input:   np       /* number of periodic variables */
         Vp[i,j]  /* array containing the periodicity and length of the */
                  /* variables, with Vp[i,1]=Tpi, Vp[i,2]=Cpi,*/
                  /* i ranging from 1 to np and j ranging from 1 to 2 */
         µCy      /* value of the micro-cycle */
         N        /* number of micro-cycles in the macro-cycle */
         bat[i,cycle]   /* i ranging from 1 to np */
                        /* cycle ranging from 1 to N */
output:  J[i]     /* maximum polling jitter of variable Vpi */

begin
   1:    for i = 1 to np do
   2:
   3:       /* Evaluate number of hits of variable Vpi */
   4:       hits = 0;
   5:       for cycle = 1 to N do
   6:          if bat[i,cycle] = 1 then
   7:             hits = hits + 1;
   8:          end if
   9:       end for;
  11:       /* Find first hit in a macro-cycle */
  12:       cycle = 0;
  13:       repeat
  14:          cycle = cycle + 1
  15:       until bat[i,cycle] = 1;
  16:
  17:       /* Find last hit in a macro-cycle */
  18:       cycle1 = N + 1;
  19:       repeat
  20:          cycle1 = cycle1 - 1
  21:       until bat[i,cycle1] = 1;
  22:
  23:       /* Evaluate the time span between the last hit in a */
  24:       /* macro-cycle and the first hit in a subsequent macro-cycle */
  25:       span = (N - cycle1 + cycle - 1) × µCy;
  26:       load_par = 0;
  27:       for j = 1 to i - 1 do
  28:          if bat[j,cycle] = 1 then
  29:             load_par = load_par + Vp[j,2]
  30:          end if
  31:       end for;
  32:       span = span + load_par;
  33:       load_par = 0;
  34:       for j = 1 to i - 1 do
  35:          if bat[j,cycle1] = 1 then
  36:             load_par = load_par + Vp[j,2]
  37:          end if
  38:       end for;
  39:       span = span + (µCy - load_par);
  40:
  41:       /* Evaluate the time span between each of the hits */
  42:       /* within a macro-cycle */
  43:       for k = 1 to hits - 1 do
  44:          cycle1 = cycle;
  45:          repeat
```

```
   46:            cycle1 = cycle1 + 1
   47:         until bat[i,cycle1] = 1;
   48:         span1 = (cycle1 - cycle - 1) × μCy;
   49:         load_par = 0;
   50:         for j = 1 to i - 1 do
   51:            if bat[j,cycle] = 1 then
   52:               load_par = load_par + Vp[j,2]
   53:            end if
   54:         end for;
   55:         span1 = span1 + (μCy - load_par);
   56:         load_par = 0;
   57:         for j = 1 to i - 1 do
   58:            if bat[j,cycle1] = 1 then
   59:               load_par = load_par + Vp[j,2]
   60:            end if
   61:         end for;
   62:         span1 = span1 + load_par;
   63:
   64:         if span1 > span then
   65:            span = span1
   66:         end if;
   67:         cycle = cycle1;
   68:      end for;
   69:      J[i] = span - Vp[i,1];
   70:   end for
return J;
-------------------------------------------------------------------------------
```

A.5. Algorithm for the evaluation of the length of the ABI

```
   ---------------------------------------------------------------
   - 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
      1:    len_abi_max = 0;
      2:
      3:    for j = 1 to N do    /* test all micro-cycles as critical */
      4:       ncy_abi = function_ncy_abi (j);
      5:
      6:       /* determine number of aperiodic transfers in */
      7:       /* the ncy_abi - 1 microcycles */
      8:       for cycle = 1 to ncy_abi - 1 do
      9:          count_p = 0;
      10:         for i = 1 to np do
      11:            if bat[i, ((cycle - 1) mod N) + 1] = 1 then
      12:               count_p = count_p + 1
      13:            end if
      14:         end for;
```

```
   15:          aw = μCy - count_p × cp;
   16:          na_aux = na_aux + aw div ca
   17:       end for;
   18:
   19:       /* determine the number of periodic scans */
   20:       /* in microcycle */
   21:       /* number ncy_abi */
   22:       count_p = 0;
   23:       for i = 1 to np do
   24:          if bat[i, ((ncy_abi - 1) mod N) + 1] = 1 then
   25:             count_p = count_p + 1
   26:          end if
   27:       end for;
   28:       len_p = count_p × cp;
   29:
   30:       /* determine length of aperiodic busy window */
   31:       len_abi = (ncy_abi - 1) × μCy + len_p + (2 × na -
   32:            na_aux) × ca
   33:       if len_abi > len_abi_max then
   34:          len_abi_max = len_abi
   35:       end if;
   36:    end for;
return len_abi;
-----------------------------------------------------------------
- Number of Cycles of the Aperiodic Busy Interval
-----------------------------------------------------------------
function ncy_abi;
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
      1:    cycle = j;
      2:    na_aux = 0;
      3:    repeat
      4:       cycle = cycle + 1;
      5:       count_p = 0;
      6:       for i = 1 to np do
      7:          if bat[i,((cycle - 1) mod N) + 1] = 1 then
      8:             count_p = count_p + 1;
      9:          end if;
      10:      end for;
      11:      aw = μCy - count_p × cp;
      12:      na_aux = na_aux + aw div ca;
      13:   until na_aux >= 2 × na;
      14:   ncy_abi = cycle;
   return ncy_abi;
   -----------------------------------------------------------------
```

References

[1] M. Groover, Automation Production Systems and Computer Integrated Manufacturing, Prentice-Hall, Englewood Cliffs, NJ, 1986.

[2] W. Gutshke, K. Mertins, CIM: competitive edge in manufacturing, Robotics and Computer Integrated Manufacturing 2 (1) (1987) 77–87.

[3] J. Pimentel, Communication Networks for Manufacturing. Prentice-Hall, Englewood Cliffs, NJ, 1990.

[4] J.-D. Decotignie, P. Pleinevaux, A survey on industrial communication networks, Annales des Télécomunications 48 (9) (1993) 55–63.

[5] A. Cardoso, E. Tovar, Industrial communication networks: issues on heterogeneity and internetworking, in: Proceedings of the 6th International Conference on Flexible Automation and Intelligent Manufacturing, Begell House Publishers, 1996, pp. 139–148.

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

[7] P-Net, The P-NET Standard, International P-NET User Organisation ApS, 1994.

[8] Profibus, PROFIBUS Standard DIN 19245 Part I and II, Profibus Nutzerorganisation e.V., 1992 (Translated from German).

[9] Afnor, Normes FIP NF C46-601–NF C46-607, UTE-Union Technique de l'Electricité, 1990.

[10] ISO 9506, Industrial Automation Systems — Manufacturing Message Specification, ISO, 1990.

[11] Y. Kim, S. Jeong, W. Kown, A pre-run-time scheduling method for distributed real-time systems in a FIP environment, in: Control Engineering Practice, Vol. 6, Pergamon, Oxford, pp. 103–109.

[12] S. Kumaran, J.-D. Decotignie, Multicycle operations in a field bus: application layer implications, in: Proceedings of IEEE Annual Conference of Industrial Electronics Society, IEEE, 1989, pp. 531–536.

[13] P. Raja, G. Noubir, Static and dynamic polling mechanisms for fieldbus networks, Operating Systems Review ACM 27 (3) (1993) 34–45.

[14] L. Liu, J. Layland, Scheduling algorithms for multiprogramming in a hard real-time environment, Journal of the ACM 20 (1) (1973) 46–61.

[15] F. Vasques, G. Juanole, Pre-run-time schedulability analysis in fieldbus networks, in: Proceedings of the 20th Annual Conference of the IEEE Industrial Electronics Society, IEEE, 1994, pp. 1200–1204.

[16] P. Pedro, A. Burns, 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, IEEE, 1997, pp. 3–10.