Industrial Multimedia Over Factory-Floor Networks

Eduardo Tovar [1], Francisco Vasques[2], Filipe Pacheco[1], Luís Ferreira[1]

[1] IPP-HURRAY Research Group, Polytechnic Institute of Porto (ISEP-IPP), Portugal

{emt, ffp, llf}@dei.isep.ipp.pt

[2] Department of Mechanical Engineering, University of Porto (FEUP), Portugal

vasques@fe.up.pt

Abstract

In this paper we describe a real-time industrial communication network able to support both control-related and multimedia traffic. The industrial communication network is based on the PROFIBUS standard, with multimedia capabilities being provided by an adequate integration of TCP/IP protocols into the PROFIBUS stack. From the operational point of view the integration of TCP/IP into PROFIBUS is by itself a challenge, since the master-slave nature of the PROFIBUS MAC makes complex the implementation of the symmetry inherent to IP communications. From the timeliness point of view the challenge is two folded. On one hand the multimedia traffic should not interfere with the timing requirements of the "native" control-related PROFIBUS traffic (typically hard real-time). On the other hand multimedia traffic requires certain levels of quality-of-service to be attained. In this paper we provide a methodology that enables fulfilling the timing requirements for both types of traffic in these real-time industrial LAN. Moreover, we describe suitable algorithms for the scheduling support of concurrent multimedia streams.

1. Introduction

Communication networks aimed at the interconnection of sensors, actuators and controllers are commonly known as fieldbus networks. In last decade, more and more standardised fieldbuses have been accepted supporting the open system communication concept and thus having a vendor independent communication. PROcess FIeld BUS (PROFIBUS) [1] is one of the most popular fieldbuses, which have significantly evolved in recent years. In fact, PROFIBUS and its communication stack were originally created as a communication support for the cell and field levels within automated factory environments. Its great flexibility however was achieved at the cost of a reduced network responsiveness, which means that PROFIBUS is not suited for high-speed real-time data exchanges. To overcome this problem, a new upper layer protocol have been developed for PROFIBUS, called Decentralised Periphery (DP) [2]. Such developments make PROFIBUS particularly suited for use as a control network in those applications

where a number of small-sized data variables have to be exchanged at a high rate. More recently, a new Physical Layer has been defined to improve the network responsiveness, which is able to support transmissions speeds up to 12 Mbit/sec.

Such recent developments are pulling PROFIBUS to support a new wide class of applications, such as industrial multimedia applications. Examples of such applications for the industrial environment include monitoring applications interfacing to microphones and cameras, remote access to maintenance data including graphics and videos, etc. These kinds of applications are usually supported by the widely used TCP/IP stack. Thus, the most effective way to integrate such applications within the PROFIBUS communication stack is to integrate the TCP/IP stack into the PROFIBUS stack.

Such integration must be correctly specified, in order to provide the adequate Quality of Service to the supported TCP/IP applications, while guaranteeing that the timing requirements of the control-related traffic are always satisfied. A transparent solution for such integration is proposed in [6], based in an adequate interface structured in three sub-layers: *IP-Mapper*; *Admission Control and Scheduler (ACS)* and *Dispatcher* (Figure 1).

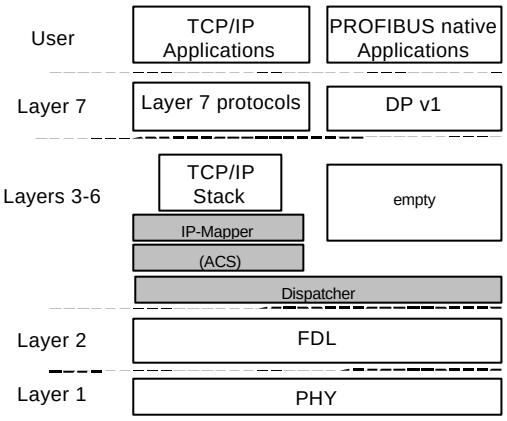

Figure 1 - Integration of the TCP/IP stack in a PROFIBUS network

The IP-Mapper sub-layer resides directly below the TCP/IP Protocol Stack. This layer is responsible for the conversion of IP packets into/from PROFIBUS FDL frames. Therefore, it maps the TCP/IP services into the PROFIBUS FDL services and performs the identification, fragmentation and re-assembly of the IP packets to/from PROFIBUS FDL frames. The IP-Mapper layer is also responsible for the transparent support of the peer-to-peer relationship inherent to the IP protocol, mapping it to the PROFIBUS FDL master/slave paradigm [6].

The Admission Control and Scheduling (ACS) sub-layer resides directly under the IP-Mapper sub-layer. The ACS sub-layer is responsible for the control/limitation of the network resources usage by the TCP/IP applications. Moreover, this sub-layer must implement appropriate scheduling policies able to provide the desired Quality of Service for the multimedia applications.

The Dispatcher sub-layer resides above the PROFIBUS FDL. Both PROFIBUS native traffic and IP traffic (fragmented in PROFIBUS frames) pass through this sub-layer. The dispatcher is responsible for maintaining proper timing constraints for the different types of traffic that are conveyed through the

network, thus providing the desired Quality of Service for multimedia applications, while guaranteeing that the timing requirements of the control-related traffic are always satisfied.

The remainder of the paper is organised as follows: in Section 2 we briefly describe the PROFIBUS MAC mechanisms, its timing behaviour and the proposed approach to jointly schedule IP-related traffic with native PROFIBUS traffic. In Section 3 we describe how by proper pre-run-time schedulability analysis the dispatcher sub-layer can be parameterised and implemented in order to provide the timing guarantees for both the control-related and multimedia traffic. Finally, in Section 4, we propose and discuss scheduling strategies for providing the QoS guarantees to concurrent IP traffic. Both on-line as well off-line implementations of the algorithms are described. In Section 5 we draw some conclusions.

2. Timing Behaviour of the PROFIBUS Protocol

A brief explanation of the PROFIBUS MAC timing behaviour is required, before discussing the rationale for the three sub-layers.

2.1. Timing Characteristics of the PROFIBUS MAC Protocol

The PROFIBUS medium access control (MAC) protocol is based on a token passing procedure (simplified version of the timed token protocol [3]) used by masters to grant the bus access to each one of them, and a master-slave procedure used by masters to communicate with slaves. One of the PROFIBUS MAC main functions is the control of the token cycle time, which is now briefly explained.

PROFIBUS defines two categories of messages: high priority and low priority. These two categories of messages use two independent outgoing queues. 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 is used as follows. 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}. If at the arrival, the token is late, that is, the real token rotation time (T_{RR}) was greater than the target rotation time (T_{TR}), the master station may execute, at most, one high priority message cycle (a message cycle is composed by a request frame and the associate response frame). 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. 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 T_{TH} overrun).

2.2. Consequences of the Token Lateness

In PROFIBUS, if a station receives a late token (T_{RR} greater than T_{TR}), then only one high priority message will be transmitted. As a consequence, low priority traffic may drastically affect the high priority traffic capabilities. In fact, if the low priority traffic is unconstrained, when a station receives an early token (T_{RR} smaller than T_{TR}) it may use all the available time ($T_{TH} = T_{TR} - T_{RR}$) to process low priority

traffic, delaying the token rotation. In this case, the subsequent stations may be limited to only one high priority message transmission when holding the token.

As the number of high priority messages that can be transferred at the token arrival is highly dependent on the amount of traffic transferred by the previous stations, a station receiving the token may become unpredictably confined to a single high priority message transfer.

Two reasons justify a late token arriving to a master [4]:

1. As once a message cycle is started, it is always completed, even if T_{TH} has expired during its execution, a late token may be transmitted to the following stations.
2. If a master receives a late token, it will still be able to send one high priority message. This may further increase the token lateness in the following masters.

Therefore, the token timing behaviour must be carefully controlled. Otherwise, the low-priority traffic of precedent stations may jeopardise the timing requirements associated to the high-priority traffic requested at any station in the network.

In [5], two different approaches were proposed to guarantee the real-time behaviour of the high priority traffic in the PROFIBUS protocol.

1. An *unconstrained low priority traffic* profile, where real-time traffic requirements are satisfied, even when only one high priority message is transmitted per token visit, independently of the low priority traffic load;
2. A *constrained low priority traffic* profile where, by controlling the number of high priority and low priority message transfers, all pending real-time traffic is transmitted at each token visit.

The analysis presented in [5] demonstrates that the first profile is a suitable approach for more responsive systems (tighter deadlines), whilst the second one allows for an increased non-real-time traffic throughput. Within the context of this paper, the second approach is the preferred one, since it allows an increased and predictable low priority traffic throughput. Therefore, in the next subsection we analyse the main characteristics of this profile, indicating how it guarantees the correct timing behaviour of the real-time traffic in the PROFIBUS protocol.

2.3. The Constrained Low Priority Traffic Profile

The correct behaviour of the *constrained low priority traffic* profile requires the fulfilment of the following assumption: *at each token arrival, the master is able to execute all pending real-time message requests*. Therefore, by the thorough analysis of the high priority and low priority traffic requirements, one must define the minimum value for the T_{TR} parameter, which ensures that an arriving token will always have time to process all the pending real-time message requests.

Complementarily, mechanisms must be implemented at each station to control the number of high priority and low priority message transfers at each token visit, preventing the overuse of the token.

2.4. Message Model

Prior to the analysis of how to set the T_{TR} parameter, lets consider the following message model. Consider a PROFIBUS network with n masters, with addresses ranging from 1 to n. Slaves will have network addresses higher than n. Consider a message stream to be a temporal sequence of message cycles concerning, for instance, the remote reading of a specific process variable.

Assume that Sh_i^k and Sl_i^k defines a real-time message stream i in master k ($k = 1, \ldots n$) of, respectively, high or low priority. These message streams are characterised by their timing requirements, that is:

- Ch_i^k and Cl_i^k are the longest message cycle duration associated to requests produced by, respectively, streams Sh_i^k or streams Sl_i^k.
- Th_i^k and Tl_i^k are the periodicity of, respectively, streams Sh_i^k and streams Sl_i^k requests, that is, the minimum inter-arrival time between two consecutive Sh_i^k (Sl_i^k) requests at the outgoing queue.
- nh^k and nl^k are, respectively, the number of messages streams Sh_i^k and Sl_i^k associated with master k.
- We assume that the deadline of a message stream is equal to its period, that is, if in the outgoing queue there are two high priority message requests from the same message stream, this means that the deadline of the first request was missed.

Concerning the non real-time traffic, we consider that it is characterised just by the length of its messages: C^{LP}, since it is not possible to define a minimum inter-arrival time between two consecutive requests. From the deadline assumption, it results that the maximum number of real-time pending requests in the outgoing queue will be ($nh^k + nl^k$).

2.5. Setting TTR for the Constrained Low Priority Traffic Profile

2.5.1. Deadline Constraint

In order to respect the message timing requirements, it must be guaranteed that a requested real-time message cycle may be processed before its deadline. We consider that this requirement must be respected for all the real-time traffic but not for the case of non real-time traffic. As a message cycle cannot be processed while the master does not receive the token, a *deadline constraint* can be defined as follows:

$$\min_{\substack{k=1..n \\ i=1..nh^k \\ j=1..nl^k}} \left\{ Th_i^k, Tl_j^k \right\} \geq Tcycle_{MAX} \tag{1}$$

where $Tcycle_{MAX}$ stands for the maximum elapsed time between two consecutive token arrivals to station k, that is, the maximum T_{RR}^k. Furthermore, as each station must be able to transmit all the pending high priority traffic plus an agreed amount of low priority traffic, an upper bound for $Tcycle$ can be defined as follows:

$$Tcycle = \sum_{k=1}^{n} Hh^k + \sum_{k=1}^{n} Hl^k + \boldsymbol{t} \tag{2}$$

where $Hh^k = \sum_{i=1}^{nh^k} Ch_i^k$ corresponds to the time to process the maximum number of high-priority requests pending at station k and Hl^k corresponds to the maximum amount of time, at each token arrival, to schedule low-priority traffic. The logical ring latency (token walk time, including node latency delay, media propagation delay, etc.) is represented by $\boldsymbol{t}$.

The deadline constraint can then be rewritten as:

$$\min_{i,j,k}\left\{Th_i^k, Tl_j^k\right\} \geq \sum_{k=1}^{n} Hh^k + \sum_{k=1}^{n} Hl^k + \boldsymbol{t} \qquad (3)$$

which ensures that even the most urgent request will be timely transferred.

2.5.2. Protocol Constraint

On the other hand, T_{TR} must be set in order to guarantee that, at the token arrival, there will be always enough time to execute all the pending high-priority requests and the agreed amount of low-priority requests. This means that, at the token arrival, $T_{TH} = T_{TR}$-T_{RR} must be enough to transfer all these requests. As a consequence, T_{TR} must be set as follows:

$$T_{TR} \geq Tcycle_{MAX} + \max_{k=1..n}\left\{T_{TH}^k\right\} \qquad (4)$$

where $T_{TH}^k = Hh^k + Hl^k$ represents the maximum amount of real-time traffic that may need to be transferred at the token arrival.

Therefore, a lower bound for T_{TR} is given by:

$$T_{TR} \geq \sum_{k=1}^{n} Hh^k + \sum_{k=1}^{n} Hl^k + \boldsymbol{t} + \max_{k=1..n}\left\{T_{TH}^k\right\} \qquad (5)$$

Equations (3) and (5) are then the basis for setting the T_{TR} parameter for the constrained low priority profile. Finally, a scenario for this traffic profile is illustrated in Figure 2.

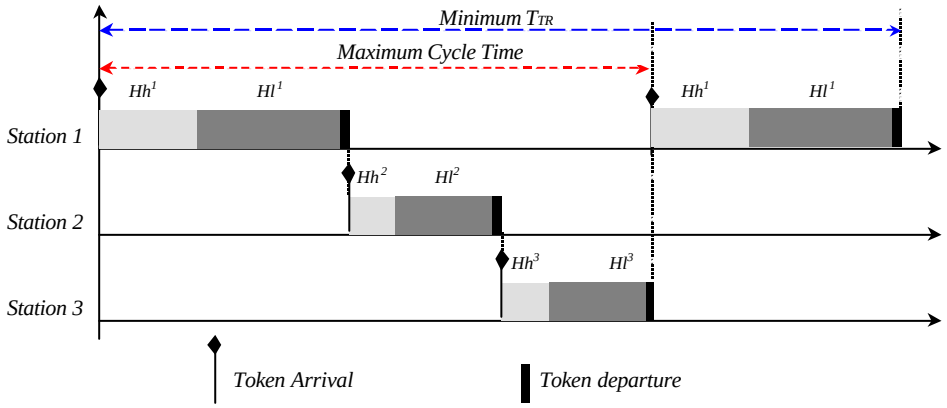

Figure 2 - Constrained Low Priority Traffic Profile

3. Dispatcher Sub-Layer

The jointly schedule of IP-related traffic with DP traffic, guaranteeing the imposed timing requirements (real-time requirements, Quality of Service) can be implemented imposing a *constrained low priority traffic* profile to the PROFIBUS FDL. One of the main responsibilities of the Dispatcher layer is to impose such constrained low priority traffic profile, considering that the processed traffic will be scheduled respecting both the deadline and the protocol constraints.

3.1. Mapping IP / DP Traffic into PROFIBUS FDL Traffic

The Dispatcher sub-layer will consider three traffic classes, which will be supported by five different FIFO queues according to the traffic source (Figure 3):

1. The *Guaranteed High-Priority* traffic, which is PROFIBUS high priority traffic that will be always scheduled on time (provided that both the protocol and deadline constraints are satisfied). This class of traffic is intended to support the DP high priority traffic with timing requirements (DPH).
2. The *Guaranteed Low-Priority* traffic, which is PROFIBUS low priority traffic that will be scheduled on time after the guaranteed high priority traffic (provided that both the protocol and deadline constraints are satisfied). This class of traffic is intended to support the following two sub-classes:
   a) DP low priority traffic with timing requirements (DPL);
   b) IP traffic with QoS requirements (IPH).
3. The *Best-Effort Low-Priority* traffic, which is PROFIBUS low priority traffic that will be scheduled after the guaranteed traffic, without any guarantees of timely delivery. This class of traffic is intended to support the following two sub-classes:
   a) IP traffic without QoS requirements (IP-BE);
   b) non real-time DP low priority traffic (DP-BE).

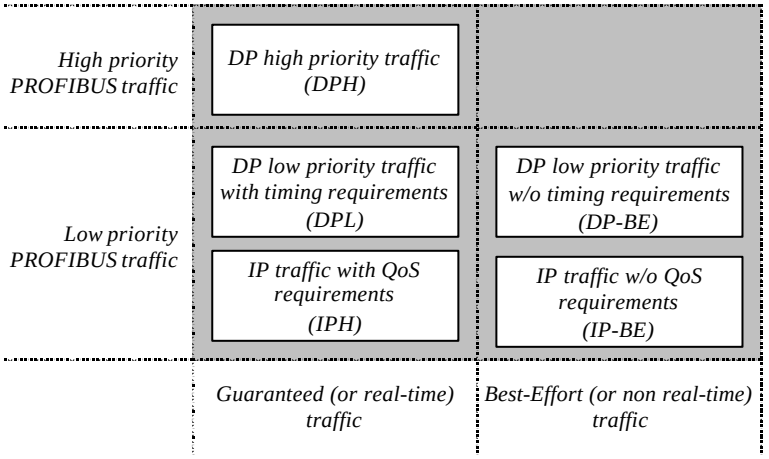

Figure 3 - Mapping the Generated Traffic on the PROFIBUS FDL Traffic

As a consequence, guaranteed traffic (real-time) is to be supported by both the high and low priority PROFIBUS traffic classes, while best-effort traffic (non real-time) is to be supported just by the low priority traffic.

3.2. Dispatcher Parameters and Master Allocation Time

In a typical PROFIBUS system implementation, a specific number of high priority DP message transactions (DPH) have to be periodically supported. Essentially these transactions will be control data exchange with slave devices. Depending on the type of message transactions, the bound for message transaction duration will be different depending on the location, required inserted idle times, etc. The sum of these durations added to a time span that considers a probabilistically obtained number of retries leads to a master parameter: the allocation to DPH traffic in a master *k* (T^k_{DPH}).

It is also important to note that DPH traffic requires to be performed with a minimum time span. As the masters share a timed token to access the transmission medium, the maximum time span between consecutive token arrivals must be bounded. This can be achieved in a PROFIBUS implementation by limiting the actual token holding time in each master in a way that the token will normally not arrive late to the masters. The system wide T_{TR} parameter must be set to a minimum value and the allocations for the other types of traffic (apart from DPH) must be made accordingly to achieve that. These will be also obtained during system planning.

T_{DCY} (token cycle time) must be choose during the system planning according the timing requirements of the most stringent (minimum interval between consecutive transactions) DPH traffic. Assume for the example of Figure 4 that T_{DCY} is 20ms. This is imposed by the application timing requirements and means that all DPH traffic considered in the allocation for each master can be served at a rate which will be in the minimum each 20ms. Of course, when DPH transactions are not performed in one T_{DCY}, the overall master allocation can be re-used for other types of traffic, namely best-effort traffic.

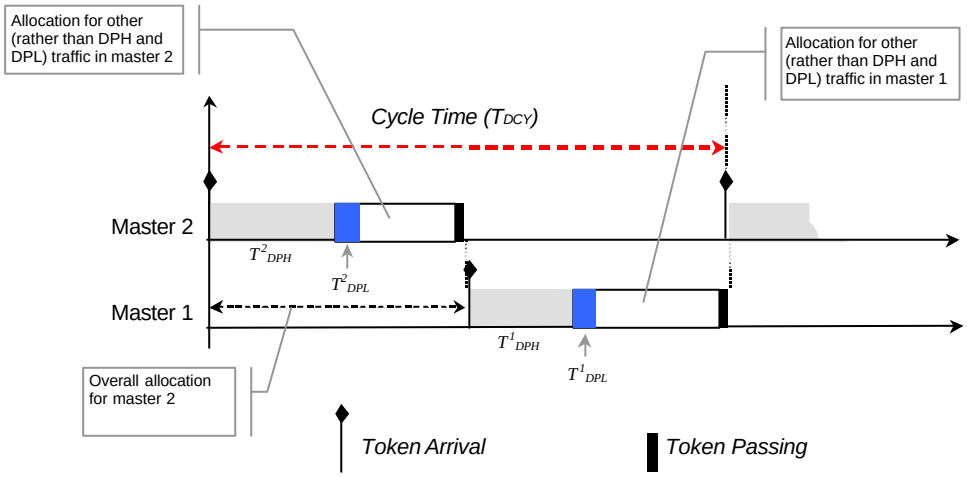

Figure 4 - DP Traffic Allocation Example

In Figure 4, the aggregate allocations concerning the other type of traffic (token passing included) could be up to 12ms (considering 4ms for DPH traffic in each master). The way they are distributed among masters is also a system planning issue. For the second class of control-related DP traffic: DPL traffic, it must be given a fair throughput. But this traffic is not as time stringent as the DPH traffic. It might be reasonable (depending on the actual application) to consider an allocation of one message transaction to each master. This would mean that at least each T_{DCY} a DPL message could be served. Again, if in some

there is no DPL traffic for the allocated time, this time will be reused, if needed to process best-effort traffic.

In the proposed PROFIBUS system implementation, multimedia traffic will also be supported. This traffic will convey IP fragments. While some multimedia traffic will not require stringent timing characteristics (IP best effort traffic - IPBE) some other multimedia traffic have, namely bandwidth and Jitter requirements (IPH traffic). The allocation reasoning for this type of traffic will be discussed in the next section. But assume that in both masters this allocation is 2ms, meaning that both T^1_{IPH} and T^2_{IPH} will have the value of 1ms. This means that for the value chosen for T_{DCY} (imposed essentially by the required rate for DPH transactions), an amount of 15ms (out of 20ms) is allocated for DPH, DPL and IPH traffic related to both masters.

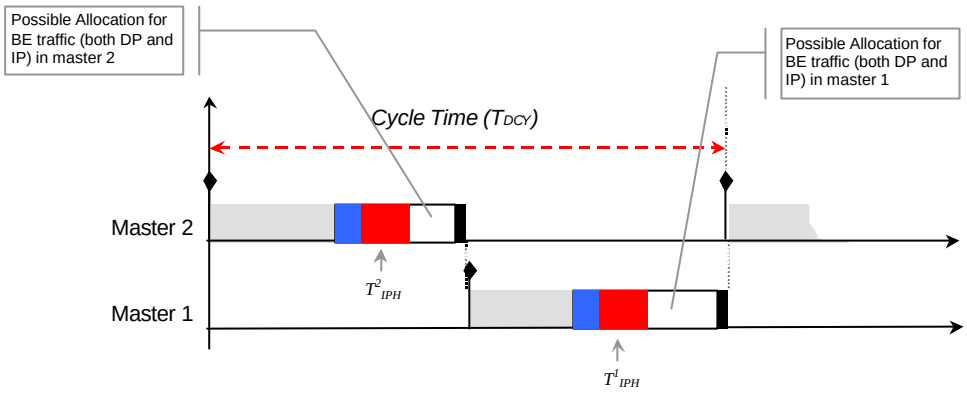

Figure 5 - IP Traffic Allocation Example

T^k_{MA} denotes the master allocation time for processing DPH, DPL, IPH and BE (both IP-BE and DP-BE) traffic in a master k. However, the sums of the T^k_{MA} for all masters can not be equal to T_{DCY}. The reason is that there is some management traffic and the token passing itself, that is generated at the FDL level. This traffic must be taken into account during the system planning.

T^k_{MA}, T^k_{DPH}, T^k_{DPL} and T^k_{IPH} are then relevant parameters for the master k Dispatcher. T_{DCY} is also a relevant parameter and will have the same value in all masters. Best effort traffic will be served at the Dispatcher according to the actual amount of the other traffic that is served in each time the Dispatcher runs (which will be at least with a rate equal to T_{DCY}).

3.3. Dispatcher Algorithm

According to the message model, a pre-defined set of dispatching rules imposes that, at each master station, the Dispatcher will cyclically transfer to the FDL layer:

a) a maximum requested DPH messages (T_{DPH})
b) a maximum configured requested time of DPL messages (T_{DPL})
c) a maximum configured requested time of IPH messages (T_{IPH})
d) a variable number of DP best effort an IP-BE messages

Such dispatching strategy will induce predictable traffic scenarios, where the token holding time (T_{TH}) is never overran (provided that T_{TR} is set according to the rules of the *constrained low priority traffic* profile).

The Dispatcher has 5 FIFO queues, one for each of the traffic classes. Each queue must hold the traffic needed for one dispatcher cycle. The Dispatcher is to be implemented on a cyclic base, and then the dispatching algorithm is trigged every T_{DCY}. At each dispatcher cycle the Dispatcher will serve its queues and transfer traffic to the FDL queues. When the dispatcher algorithm is triggered, it will start processing DPH. After processing DPH traffic, the dispatcher will serve DPL traffic until: the T_{DPL} is consumed *or* there is no more available time for the current dispatcher cycle. After processing DPL traffic, the dispatcher will serve IPH traffic until: the T_{IPH} is consumed *or* there is no more available time for the current dispatcher cycle. Finally, after processing the IPH traffic, the dispatcher will serve Best Effort traffic: one message from DP-BE queue if it fits the remaining time of the dispatcher cycle; one message from IP-BE queue if it fits the remaining time of the dispatcher cycle; this will be repeated until the queues are empty or no traffic from the queues will fit the remaining time of the dispatcher cycle.

3.4. Synchronisation Issues

Ideally, the dispatching activities would be synchronised with the token arrivals at the FDL layer, maximising the available throughput, since at each token arrival there would be, at most, the agreed number messages to be transferred. However, such synchronisation is not easy, since it would imply modifications of the PROFIBUS FDL. Then, in order to guarantee that the assumptions of the constrained low priority traffic profile are always satisfied, it is considered that the token arrives at the station at the same rate that the Dispatcher is executed, that is every T_{DCY}.

As a consequence, the traffic throughput cannot be maximised, since there will be some token arrivals where there is no traffic to be transferred at the FDL layer. For example, if $T_{DCY}=20$ms and $Tcycle_{average}=15$ms, the traffic at station *k* would be processed as presented in Figure 6.

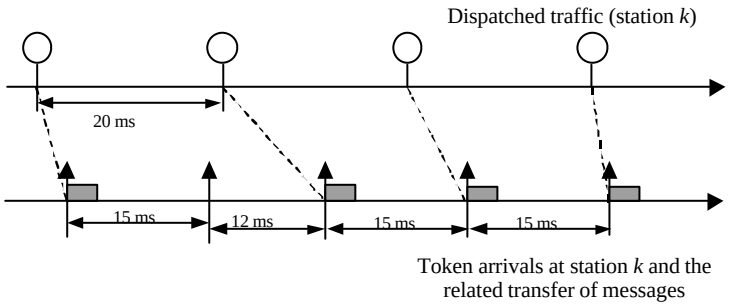

Figure 6 - Absence of Synchronisation between the Dispatcher and the FDL

4. Scheduling the IP Traffic

In each master, the available IPH slots must be used to convey the IP traffic with QoS requirements. The ACS sub-layer is composed of several relationship entities (REs) and a scheduler. Essentially each RE relates to a particular TCP/IP stream flow. Essentially, and apart from a number of functionalities related

to fragment discarding, etc. (see [6] for more details), each RE will include a queue, used to store the IP fragments coming from the IP-mapper. Fragments pending in these queues are passed to the dispatcher sub-layer by the scheduler.

As previously mentioned, the multimedia traffic can typically be of two types: traffic that does not require stringent timing characteristics (denoted as IP Best Effort traffic – IPBE), and multimedia traffic characterised by specific QoS characteristics, namely bandwidth and jitter (referred to as IPH). At the ACS sub-layer, there will exist as many REs (for both IPH and IPBE traffic) as the TCP/IP stream flows. Each RE will also have associated timing parameters that will be used by the scheduler.

4.1. Basics on the Scheduler

Assume that a particular master will have to support a maximum of 5 simultaneous IPH traffic flows, as characterised by the following table:

	Periodicity (T_{DCY})	Message Cycle Duration (µs)
IPH1	1	100
IPH2	3	200
IPH3	3	200
IPH4	4	400
IPH5	4	1000

Table 1

Transactions will typically carry the same amount of multimedia data (IP packets are fragmented in several PROFIBUS frames). The different durations for the message cycles can be justified because transactions may take place between masters and slaves in interconnected by a variable number of network repeaters, which of course will introduce different values of latencies, as it happens in hybrid wired/wireless broadcast networks [7].

Typically, these IPH traffic flows will correspond to streamed data (Audio, Video). Streamed data requires stable bandwidth and is also quite sensitive to jitter and delay. Take for example IPH1. In each 50 T_{DCY} ($50 \times 20 = 1000$ms), 50 IPH1 transactions should be served. If, for example, each transaction corresponds to the transfer of 100 bytes (800 bits) of multimedia data, this provides for bandwidth. In this case it will be $50 \times 800 = 40000$bps. However, it is different to serve these 40000 bits in the last few ms of the second or distributing this load throughout the whole second. Minimising jitter demands following this second approach.

The scheduler will run each T_{DCY}. If at each time the scheduler runs, there are pending fragments in all RE IPH queues, and for the table above, the scheduler could perform in the following way:

Cycle	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
IPH1	1	1	1	1	1	1	1	1	1	1	1	1
IPH2	1			1			1			1		
IPH3	1			1			1			1		
IPH4	1				1				1			
IPH5	1				1				1			
Load (µs)	**1900**	**100**	**100**	**500**	**1500**	**100**	**500**	**100**	**1500**	**500**	**100**	**100**

Table 2

This is a typical example of a multi-cycle schedule [8], where the macro-cycle corresponds to 12 (the pattern repeats after 12 T_{DCY}). The macro-cycle is obtained by lowest common multiple (LCM) of the periodicities. The scheduler will schedule 1 fragment (if for example the periodicity of IPH1 was 0.5 T_{DCY}, then a 2 would appear in each entry of the table for IPH1) for each of the IPH REs in the first micro-cycle. Then in the second micro-cycle it will schedule 1 fragment concerning IPH1 and so forth.

For this particular schedule, the minimum value for T_{IPH} (the allocation for the QoS IP traffic) would be 1900µs. This load may take place each 12 cycles if the 5 IP flows are active simultaneously.

As the scheduler will run each T_{DCY}, the required bandwidth for the IP traffic is guaranteed, and the flow control scheme (window size control) of the TCP is encompassed.

Because it is not relevant (for instance concerning jitter) to shift some fragments between micro-cycles (they are fragments of IP packets which need to be reassembled in the receiving side) the schedule could be optimised in order to minimise the value for T_{IPH}. In fact several policies like best-fit or worst-fit could be used in order to modify the schedule illustrated in Table 2.

Cycle	1	2	3	4	5	6	7	8	9	10	11	12
IPH1		2	1	1		2	1	1		2	1	1
IPH2		1		1			1			1		
IPH3		1		1			1			1		
IPH4		1				1				1		
IPH5	1				1				1			
Load (µs)	**1000**	**1000**	**100**	**500**	**1000**	**600**	**500**	**100**	**1000**	**1000**	**100**	**100**

Table 3

A schedule can be supplied to the scheduler through a table, which is defined during system planning (off-line schedule). In this case the scheduler will only perform dispatching: for each micro-cycle, it will execute the related schedule for that micro-cycle. Then it will increment the micro-cycle counter and perform the next schedule (another entry in the table column) till the micro-cycle which finishes the macro-cycle. The disadvantages are that off-line schedules (made available through a static table) may be memory consuming. This is particular acute if the value for the macro-cycle (which only depends on the periodicity of the IPH fragments) is very high. Therefore, we propose the use of an on-line scheduler, which we describe in the next sub-section. This on-line scheduler minimises the required TIPH value, while maintaining strict periodicity in each IPH stream flow. This on-line scheduler is based on the off-line deferred-release algorithm proposed in [9] for building a Bus Arbitrator Table for WorldFIP industrial networks.

4.2. The Proposed Scheduling Algorithms

For the example of Table 1, it is easy to see (from Table 2) that some of the lines of Table 2 could be shifted in order to try to minimise the value T_{IPH} for while maintaining the strict periodicities the frames for each IPH flow are served. Table 4 illustrates this new schedule.

Cycle	1	2	3	4	5	6	7	8	9	10	11	12
IPH1	1	1	1	1	1	1	1	1	1	1	1	1
IPH2	1			1			1			1		
IPH3		1			1			1			1	
IPH4	1				1				1			
IPH5		1				1				1		
Load (µs)	700	1300	100	300	700	1100	300	300	500	1300	300	100

Table 4

With this schedule, the minimum value for T_{IPH} will be 1300µs.

The following algorithm can be an algorithm performed during system planning for both obtaining the schedule table (if off-line schedule is used) and the value concerning T_{IPH} for a given RFieldbus master. First, there is the need to obtain the value for the macro-cycle (which in any case will be a parameter required by the scheduler).

```
function macro-cycle;
input:  niph            /* number of IPH flows (number of RelationshipEntities) */
        tp[i]           /* vector containing the periodicity of the fragments */
        Tdcy            /* value for TDCY, which is also the scheduler cycle */
output:Mcy              /* number of cycles of the macro-cycle */

begin
        1:      max = 0;
        2:      for i = 1 to niph do
        3:              if tp[i] > max then
        4:                      max = tp[i]
        5:              end if
        6:      end for;
        7:      N = max - 1;
        8:      ctrl = FALSE;
        9:      while ctrl = FALSE do
        10:             N = N + 1;
        11:             ctrl = TRUE;
        12:             for i = 1 to niph do
        13:                     if N mod (tp[i]/Tdcy) <> 0 then
        14:                             ctrl = FALSE
        15:                     end if
        16:             end for
        17:     end while;
        18:     MCy = N;
return MCy;
```

An algorithm for obtaining a schedule like that as given by Table 4 for the traffic as given by Table 1 is presented below. This algorithm determines also the value for T_{IPH}. The output will be a table containing the schedule (*sched* [,]), the value for T_{IPH} and a vector for the shift (*offset* [ ]). These two last parameters are not relevant for a scheduler based on a off-line schedule (table).

```
function schedule;
input:  niph            /* number of IPH flows (number of RelationshipEntities)*/
        tp[i]           /* vector containing the periodicity of the fragments */
                        /* ORDERED by periodicities; i ranging from 1 to niph */
        cp[i]           /* vector containing the transaction duration of the fragments */
                        /* i ranging from 1 to niph */
                        /* (Target Message Cycle Time) */
        Tdcy            /* value for TDCY, which is also the scheduler cycle */
        Mcy             /* number of cycles in the macro-cycle */

outputs:
        sched[i,cycle]/* i ranging from 1 to niph */
                        /* cycle ranging from 1 to Mcy */
        offset[i]       /* shift in the line pattern */
                        /* i ranging from 1 to niph */
        tiph            /* value for the parameter TIPH */
```

```
begin
        1:      /* obtains the shift */
        2:      for i = 1 to niph do
        3:              min_load = MAXINT;
        4:              for cycle = 1 to (tp[i] div Tdcy)
        5:                      cycle1 = cycle;
        6:                      max_load = 0;
        7:                      repeat
        8:                              if load[cycle1] > max_load then
        9:                                      max_load = load[cycle1];
       10:                              end if;
       11:                              cycle1 = cycle1 + (tp[i] div Tdcy)
       12:                      until cycle1 > Mcy;
       13:                      if max_load < min_load then
       14:                              cycle_min = cycle;
       15:                              min_load = max_load;
       16:                      end if
       17:              end for
       18:      end for;
       19:      cycle = cycle_min;
       20:      offset[i] = cycle_min - 1;
       21:
       22:      /* updates the load in each cycle and builds the schedule */
       23:      repeat
       24:              load[cycle] = load[cycle] + cp[i];
       25:              sched[i,cycle] = 1;
       26:              cycle = cycle + (tp[i] div Tdcy);
       27:      until cycle > Mcy;
       28:
       29:      /* obtains the value for T_IPH */
       30:      tiph = 0;
       31:      for i = 1 to Mcy do
       32:              if load[i] > tiph then
       33:                      tiph = load[i];
       34:              end if
       35:      end for;
return sched, offset, tiph;
```

Providing the schedule table as a parameter to the scheduler (which in fact will be more like a dispatcher), the scheduler algorithm will be as follows (note that the scheduler must run each T_{DCY}):

```
function scheduler (based on a off-line schedule);
input:  niph            /* number of IPH flows */
                        /* (number of IPH RelationshipEntities) */
        nipbe           /* number of IPBE flows */
                        /* (number of IPBE RelationshipEntities) */
        Mcy             /* number of cycles in the macro-cycle */
        sched[i,cycle] /* i ranging from 1 to niph */
                        /* cycle ranging from 1 to Mcy */

outputs:

begin
        1:
        2:
        3:      /* this runs only first time */
        4:      if start = TRUE then
        5:              cycle = 1;
        6:              start = FALSE
        7:      end if;
        8:
        9:      cycle = cycle + 1;
       10:      if cycle > Mcy then
       11:              cycle = 1
       12:      end if;
       13:
       14:      /* i identifying iph_Entities */
       15:      for i = 1 to niph do
       16:              if sched[i,cycle] > 0 then
       17:                      /* sched_queue gets a pending request, if any, from the */
       18:                      /* i Relationship Entity, and passes it to the FDL */
       19:
       20:                      sched_queue(i)
       21:              end if;
```

```
        22:     end for;
        23:
        24:     /* Part for the IP-BE Traffic */
        25:
        26:     repeat
        27:             flag = FALSE;
        28:             /* i identifying ipbe_entities */
        29:
        30:             for i = 1 to nipbe do
        31:                     /* in the following case sched_queue returns 0 */
        32:                     /* if either there are no pending requests in the */
        33:                     /* relationship entity queue or BE dispatcher queue */
        34:                     /* does not accept */
        35:
        36:                     if sched_queue(i) <> 0 then
        37:                             flag = TRUE
        38:                     end if;
        39:             end for;
        40:     until flag = FALSE;
return;
```

For the on-line scheduler (the scheduler obtains the actual schedule for each cycle) performing the scheduling as illustrated in Table 4, parameters such as *tp*[*i*] and *offset* [ ] are also required from system planning (through System Management). The algorithm is as follows (note that the scheduler must run each T_{DCY}):

```
function scheduler (based on a on-line schedule);
input:  niph            /* number of IPH flows */
                        /* (number of IPH RelationshipEntities) */
        nipbe           /* number of IPBE flows */
                        /* (number of IPBE RelationshipEntities) */
        Mcy             /* number of cycles in the macro-cycle */
        tp[i]           /* vector containing the periodicity of the fragments */
                        /* ORDERED by periodicities; i ranging from 1 to niph */
        offset[i]       /* shift in the line pattern */
                        /* i ranging from 1 to niph */

outputs:

begin
1:
2:      /* this runs only first time */
3:      if start = TRUE then
4:              cycle = 1;
5:              for i = 1 to niph do
6:                      num_sch[i] = 0
7:              end for;
8:              start = FALSE
9:      end if;
10:
11:     cycle = cycle + 1;
12:     if cycle > Mcy then
13:             cycle = 1
14:             for i = 1 to niph do
15:                     num_sch[i] = 0
16:             end for;
17:     end if;
18:
19:     /* i identifying iph_Entities */
20:     for i = 1 to niph do
21:             hit = ((cycle - offset[i]) div tp[i]);
22:             if ((cycle - offset[i]) mod tp[i]) <> 0 then
23:                     hit = hit + 1;
24:             end if;
25:             hit = hit - num_sch[i];
26:             if hit > 0 then
27:                     /* sched_queue gets a pending request, if any, from the */
28:                     /* i Relationship Entity, and passes it to the FDL */
29:
30:                     sched_queue(i)
31:
32:                     num_sch[i] = num_sch[i] + 1;
33:             end if;
```

```
34:     end for;
35:
36:     /* Part for the IP-BE Traffic */
37:
38:     /* AS DESCRIBED IN THE OFF-LINE SCHEDULER */

return;
```

Fragmentation of IP packets (at the IP-mapper) and the timing behaviour of the IP applications will imply a non-periodic behaviour of the arrival pattern of IP fragments to the RE queues. This will not be the only reason why even with a IPH stream going on, when the scheduler executes *get_queue*(*i*), the queue in the Relationship Entity *i* is empty.

Therefore, in some cycles the scheduled transaction will not actually take place. Table 5 illustrates this situation:

Cycle	1	2	3	4	5	6	7	8	9	10	11	12
IPH1	0	0	1	1	1	1	1	1	1	1	1	1
IPH2	1			1			1			1		
IPH3		1			1			1			1	
IPH4	1				1				1			
IPH5		0				1				1		
Load (µs)	700 -100	1300 -1100	100	300	700	1100	300	300	500	1300	300	100

Table 5

"Compensation" for this can be achieved in the next macro-cycle (another resolution can be used), by scheduling the "failed" transactions in next micro-cycles with available T_{IPH}.

Therefore, if in the next macro-cycle all queues have fragments for the schedule of Table 4, and IPH1 and IPH5 have additional fragments, then, the actual schedule in the next macro-cycle would be as given in Table 6.

Cycle	1	2	3	4	5	6	7	8	9	10	11	12
IPH1	3	1	1	1	1	1	1	1	1	1	1	1
IPH2	1			1			1			1		
IPH3		1			1			1			1	
IPH4	1				1				1			
IPH5		1	1			1				1		
Load (µs)	700 +200	1300	100 +1000	300	700	1100	300	300	500	1300	300	100

Table 6

Both the off-line and the on-line scheduler algorithms previously presented can be updated to incorporate this feature. Below, we exemplify with the on-line scheduler version. This algorithm requires as additional parameters, *cp*[*i*] and T_{IPH}. Note that for each cycle the load actually used must be computed. Note also, in the algorithm description, shaded parts corresponding to additions in order to perform the "compensation".

```
function scheduler_with_compensation (based on a on-line schedule);
input:  niph            /* number of IPH flows */
                        /* (number of IPH RelationshipEntities) */
        nipbe           /* number of IPBE flows */
                        /* (number of IPBE RelationshipEntities) */
        Mcy             /* number of cycles in the macro-cycle */
        tp[i]           /* vector containing the periodicity of the fragments */
```

```
                        /* ORDERED by periodicities; i ranging from 1 to niph */
        offset[i]       /* shift in the line pattern */
                        /* i ranging from 1 to niph */
        tiph            /* value for the parameter T_IPH */
        cp[i]           /* vector containing the transaction duration of the fragments */
                        /* i ranging from 1 to niph */
                        /* (Target Message Cycle Time) */
```

outputs:

```
begin
1:
2:      /* this runs only first time */
3:      if start = TRUE then
4:              cycle = 1;
5:
6:              for i = 1 to niph do
7:                      num_sch[i] = 0;
8:                      num_disp[i] = 0;
9:                      req[i] = Mcy div tp[i]; /* normal number in macro-cycle */
10:                     req_fail[i] = 0;
11:             end for;
12:             start = FALSE
13:     end if;
14:
15:     cycle = cycle + 1;
16:     if cycle > Mcy then
17:             cycle = 1
18:             /* i identifying iph_Entities */
19:             for i = 1 to npih do
20:
21:                     num_sch[i] = 0
22:
23:                     /* at each macro-cycle updates the number of failed */
24:                     /* transactions from the previous macro-cycle */
25:
26:                     req_fail[i] = req_fail[i] + (req[i] - num_disp[i]);
27:
28:                     /* limits the number of failed to support the */
29:                     /* when a stream is not active */
30:
31:                     if req_fail[i] > req[i] then
32:                             req_fail[i] = req[i]
33:                     end if
34:             end for;
35:
36:             /* i identifying iph_Entities */
37:             for i = 1 to niph do
38:                     num_disp[i] = 0
39:             end for;
40:     end if;
41:
42:     load_cycle = 0;
43:
44:     /* i identifying iph_Entities */
45:     for i = 1 to niph do
46:             hit = ((cycle - offset[i]) div tp[i]);
47:             if ((cycle - offset[i]) mod tp[i]) <> 0 then
48:                     hit = hit + 1;
49:             end if;
50:             hit = hit - num_sch[i];
51:             if hit > 0 then
52:                     /* sched_queue gets a pending request, if any, from the */
53:                     /* i Relationship Entity, and passes it to the FDL */
54:
55:                     if sched_queue(i) <> 0 then
56:                             load_cycle = load_cycle + cp[i];
57:                             num_disp[i] = num_disp[i] + 1;
58:                     end if;
59:
60:                     num_sch[i] = num_sch[i] + 1;
61:             end if;
62:     end for;
63:
64:     /* at this point, the actual load is calculated for the current cycle */
65:     /* now do the required compensation in the remaining tiph */
```

```
66:
67:     /* i identifying iph_Entities */
68:     for i = 1 to npih do
69:             if req_fail[i] <> 0 then
70:                     j = req_fail[i];
71:                     for k = 1 to j do
72:                             if (load_cycle + cp[i]) <= tiph then
73:                                     if sched_queue(i) <> 0 then
74:                                             load_cycle = load_cycle + cp[i];
75:                                             req_fail[i] = req_fail[i] - 1;
76:                                     end if;
77:                             end if;
78:                     end for;
79:             end if;
80:     end for;
81:
82:     /* AS PREVIOUSLY DESCRIBED FOR THE IPBE TRAFFIC */

return;
```

5. Conclusions

In this paper, a real-time industrial communication network based on the PROFIBUS standard is proposed. Such network supports both control-related and TCP/IP-related traffic, and is able to provide the adequate Quality of Service to the supported TCP/IP applications, while guaranteeing that the timing requirements of the control-related traffic are always satisfied. Several real-time issues were explored within the proposed architecture. One of those issues was a pre-run-time strategy that enables performing allocation in the master token holding time for different types of traffic. This pre-run-time analysis permits to parameterise the proposed Dispatcher sub-layer. Additionally, scheduling algorithms are proposed to provide the desired Quality of Service for multimedia applications

References

[1] EN 50170, "General Purpose Field Communication System", European Standard, CENELEC, 1996, Vol. 2/3.

[2] DIN 19245, "PROFIBUS-DP - Process Field Bus Decentralised Periphery (DP) - Part 3", Draft Standard DIN 19245, issue 1994.

[3] Grow, R.: “A Timed Token Protocol for Local Area Networks”, Proceedings of Electro’82, Token Access Protocols, Paper 17/3, May 1982.

[4] Tovar, E. and Vasques, F.: “Cycle Time Properties of the PROFIBUS Timed-Token Protocol”. Computer Communications 22 (1999), pp. 1206-1216, Elsevier.

[5] Tovar, E. and Vasques, F.: “Real-Time Fieldbus Communications Using Profibus Networks”. IEEE Transactions on Industrial Electronics, vol. 46, nº 6, December 1999.

[6] Pacheco, F., Tovar, E., Kalogeras, A. and Pereira, N., “Supporting Internet Protocols in Master-Slave Fieldbus Networks”, Technical Report, Polytechnic Institute of Porto, HURRAY-TR-0124, April 2001.

[7] Alves, M., Tovar, E. and Vasques, F., “On the Adaptation of Broadcast Transactions in Token-Passing Fieldbus Networks with Heterogeneous Transmission Media”, Technical Report, Polytechnic Institute of Porto, HURRAY-TR-0121, April 2001.

[8] Raja, P. and Noubir, G., “Static and Dynamic Polling Mechanisms for Fieldbus Networks”. ACM Operating Systems Review, Vol. 27, No. 3, pp. 34-45, 1993.

[9] Tovar, E. and Vasques, F., “Distributed Computing for the Factory-floor: a Real-Time Approach Using WolrdFIP Networks”. Computers in Industry, Elsevier Science, Vol. 44, No. 1, pp. 11-30, January 2001.