Scheduling IP Traffic in Multimedia-Enabled PROFIBUS Networks[1]

Luís L. Ferreira, Sandra Machado, Eduardo Tovar
IPP-HURRAY Research Group
Polytechnic Institute of Porto (ISEP-IPP)
Rua Dr. António Bernardino de Almeida, 431
4200-072 Porto
Portugal
{llf, smachado, emt}@dei.isep.ipp.pt

***Abstract* – Recent technological developments are pulling fieldbus networks to support a new wide class of applications, such as industrial multimedia applications. To enable its use in this kind of applications the TCP/IP suite of protocols can be integrated within a fieldbus stack, leading to a dual-stack approach that is briefly outlined in this paper. One important requirement that must be fulfilled by this approach is that the hard real-time guarantees provided to the control-related traffic ("native" fieldbus traffic) are kept. At the same time it must also provide the desired quality of service (QoS) to IP applications. The focus of this paper is on how, in such a dual-stack approach, QoS can be efficiently provided to IP applications requiring quasi-constant bandwidth.**

I. INTRODUCTION

Local area networks (LANs) are becoming increasingly popular in industrial computer-controlled systems. LANs allow field devices like sensors, actuators and controllers to be interconnected at low cost, using less wiring and requiring less maintenance than point-to-point connections [1]. Besides the economic aspects, the use of LANs in industrial computer-controlled systems is also reinforced by the increasing decentralisation of control and measurement tasks, as well as by the increasing use of intelligent microprocessor-controlled devices. Broadcast LANs aimed at the interconnection of sensors, actuators and controllers are commonly known as fieldbus networks.

Recent technological developments are pulling fieldbus networks to support a new wide class of applications, such as industrial multimedia applications. Examples of such applications for the industrial environment include video, audio, file transfer, http, etc. These kinds of applications can be supported by the TCP/IP protocol, which is widely used, vendor independent, standardised, and interoperable with almost every operating system [2,3].

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

[1] This work was partially supported by Fundação para a Ciência e Tecnologia under project CIDER (POSI/1999/CHS/33139) and by the European Commission under the project RFieldbus (IST-1999-11316).

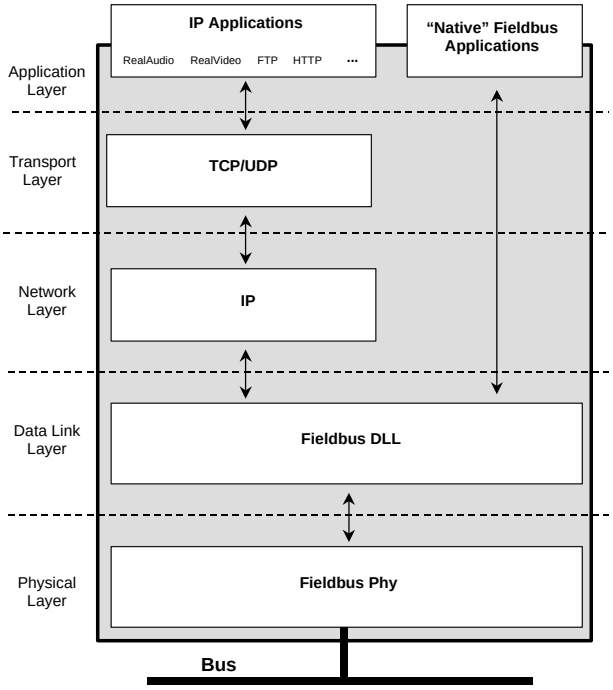

Fig. 1 Integration of the TCP/IP protocol into a generic fieldbus stack

To enable its use in industrial multimedia applications the TCP/IP suite of protocols can be integrated with the fieldbus stack, leading to a dual-stack approach as briefly outlined in Fig. 1.

However, there are some relevant aspects that such integration must take into account. In fact what is inherent to the approach of Fig. 1 is that the IP packets are to be encapsulated within fieldbus data frames. Typically this requires that the IP packets are fragmented/ de-fragmented. This functionality must be supported by an *IP Adapter Sub-layer* (IPAS), which must be placed between the IP and the Fieldbus DLL. The IPAS must also support mechanisms able to provide the tunnelling of IP traffic between fieldbus nodes which do not have communication initiative (slave nodes), as it happens in PROFIBUS [4], WorldFIP [5] or P-NET [6] protocols.

Another important requirement that must be fulfilled by the approach outlined in Fig. 1 is that the hard real-time guarantees provided to the control-related traffic ("native" fieldbus traffic) are kept. At the same time the proposed approach must also provide the desired quality of service

(QoS) to IP applications. To achieve this dual goal it is most probably required to have a sub-layer, to which we call Traffic Manager Sub-layer (TMS), between the Fieldbus DLL and the upper layers (Fieldbus Application Layer and IPAS - not directly the IP).

The rest of this paper is organised as follows. In Section II we briefly introduce a dual-stack architecture which integrates TCP/IP into a specific fieldbus (PROFIBUS). In Section III we describe how QoS guarantees can be provided to IP related traffic while at the same time keeping the hard real-time guarantees of the native PROFIBUS control-related traffic. The basic ideas concerning the IP traffic scheduler are introduced and then further detailed in Section IV. Some analogies are made to the scheduling of the WorldFIP periodic traffic. However, and contrarily to the case of WorldFIP, IP fragments to be scheduled can suffer some communication jitter. In Section V we describe how scheduling algorithms for the IP fragments can benefit from this possibility. Finally, in Section VI we discuss the possibilities to improve the scheduling algorithms in a away that constant bandwidth is provided to IP fragments even if they arrive with a non-periodic pattern. In Section VII some conclusions are drawn.

II. AN IP-ENABLED FIELDBUS ARCHITECTURE

Fig. 1 can be further detailed to include both the IPAS and TMS sub-layers, as shown in Fig. 2.

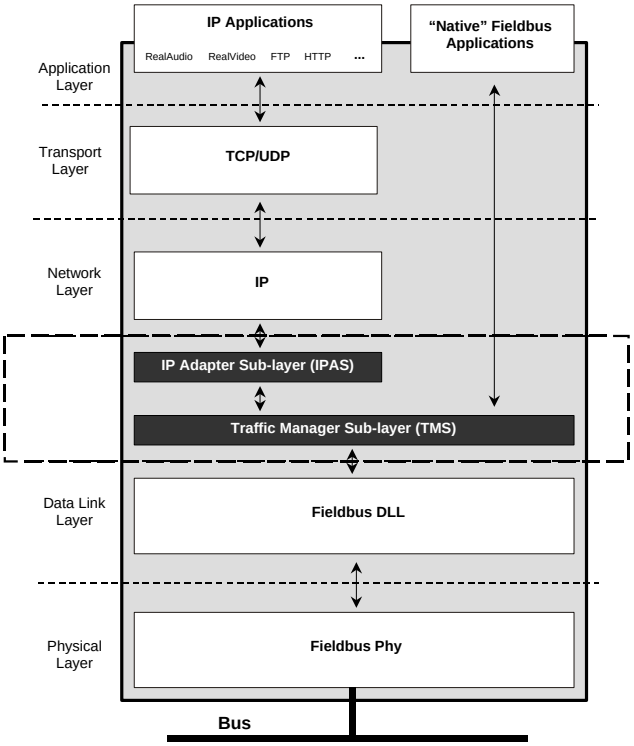

Fig. 2 Details on the IP integration

The rationale for the TMS can be briefly explained with the particular case of PROFIBUS networks, just as an example.

The PROFIBUS medium access control (MAC) protocol is based on a token passing procedure (simplified version of the timed token protocol [7]) 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 will now be 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).

In PROFIBUS, if a master 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.

This is one of the major differences to the timed token protocols used in IEEE802.4 [10] and FDDI [11], where synchronous bandwidth allocation is possible, thus allowing approaches such as those found in [12, 13, 14].

In PROFIBUS, 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 [8]:

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 [9], 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 [9] demonstrates that the first profile is a suitable approach for more responsive systems (tighter deadlines), whilst the second one allows for an increased low-priority traffic throughput. Within the context of this paper, the second approach is the basis for the TMS.

Basically, the purpose of the TMS is to provide some form of allocation of the token holding time to the different types of traffic, as shown in Fig. 3.

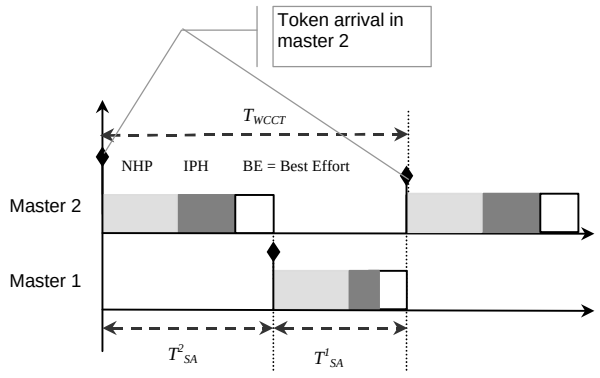

Fig. 3 Traffic allocation example for a 2-master network

The network parameters are set in a way that each master *i* is able to hold the token for T^i_{SA} time (station allocation). In this profile each type of real-time traffic, either NHP (native high priority fieldbus control-related traffic) or IPH[2] (IP traffic with QoS requirements) will have a portion of T^i_{SA} (see [15] for some details on how to set these partial allocations). This guarantees, for the case of PROFIBUS networks, that the worst-case token cycle time is bounded to T_{WCCT}. This will be an important notion throughout the rest of the paper, and will be denoted for scheduling of IP traffic as T_{IPCY}.

We denote the allocation for the IPH traffic as T_{IPH}. It will be used in each token arrival to serve the IPH traffic (fieldbus frames containing IP fragments).

Typically there will be a number of IP flows between a particular station and other stations. These IP flows can have different QoS requirements, namely bandwidth and allowed jitter. Therefore, the IPAS must schedule properly the different IP fragments related to the different IP flows.

The crucial guideline for the Scheduler is that it will have to schedule the appropriate T_{IPH} amount of IP traffic to be transferred each T_{IPCY}.

III. BASICS OF THE IP TRAFFIC SCHEDULER

Take as an example the stream set example presented in Table 1.

Table 1 – IPH stream set example

	Periodicity (T_{IPCY})	Transaction Duration (ms)
IPH1	1	1.3
IPH2	2	1.1
IPH3	4	0.3
IPH4	6	0.1
IPH5	6	1.3

An IPH stream is a temporal sequence of message cycles conveying IP fragments. The notion of message cycle results from the underlying fieldbus data link layer. In fieldbus networks message requests have typically immediate replies. Therefore, a transaction (message cycle) will have a time length corresponding to the time to send the request frame up to completely receive the response frame. One could wonder why the different values presented in table 1 for the transaction duration. The differences do not concern the amount of bits involved in the transactions (typically each frame will contain the same number of IP-fragment bits) but rather due to different propagation and turnaround times, as happens in broadcast wired/wireless networks [16, 17].

Also inherent to Table 1 is the notion of multicycle operation [18, 19]. In our case, the primary cycle will be the cycle at which the scheduler will operate (T_{IPCY}), which in turn corresponds to the worst-case token cycle time (see Section II). All other periods, called secondary cycles, are defined as integer multiples of the primary cycle. If for instance the value of T_{IPCY} = 10 ms and T_{IPH} = 5 (both parameters computed in the system planning phase - prior to run time), the scheduler could produce the following schedule (Fig. 4). Fig. 4 assumes that in each cycle the IPH queues have at least one pending fragment, so the actual dispatching corresponds to the schedule produced in each cycle.

Multicycle scheduling can be found in fieldbus networks (e.g., WorldFIP) for regulating transfers of periodic variables (control-related). The example shown in Table 1 could correspond to 5 periodic process variables to be scanned with required periodicities as

[2] This traffic is mapped onto the low priority PROFIBUS transactions

shown. An important concept in WorldFIP is that of WorldFIP BAT (Bus Arbitrator Table). Two important parameters are associated with a WorldFIP BAT: the microcycle (elementary cycle) and the macrocycle. The microcycle imposes the maximum rate at which the BA performs a set of scans. Usually, the microcycle is set equal to the highest common factor (HCF) of the required scan periodicities. It is easy to depict that the sequence of microcycles repeats each 12 microcycles. This sequence of microcycles is said to be the macrocycle, and its length is given by the lowest common multiple (LCM) of the scan periodicities.

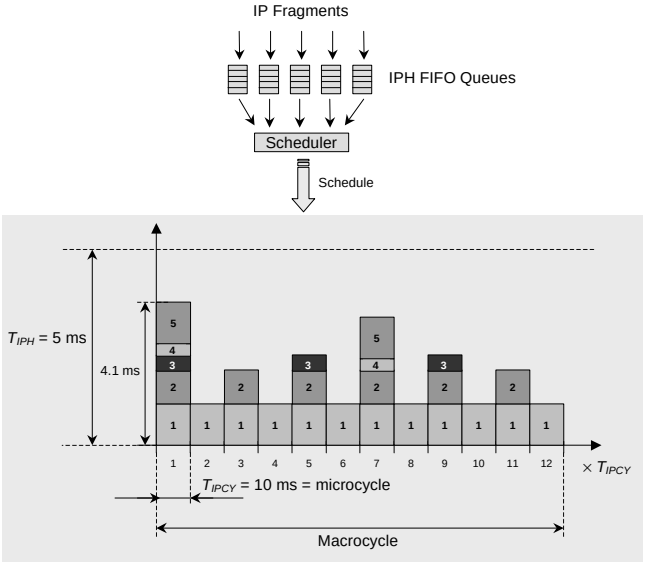

Fig. 4 Example of scheduling of the IP stream set of Table 1

One of the differences between the multicycle scheduling of periodic variables in WorldFIP and the scheduler for our IPAS is that in WorldFIP a complete cycle schedule can last up to the value of the microcycle. For our case a complete cycle schedule can only last up to T_{IPH}, which is just a portion of T_{IPCY} (microcycle).

The HCF/LCM approach for building the WorldFIP BAT has some properties that have been posing some interesting challenges to researchers [20]:

1. The variables are not scanned at exactly regular intervals (communication jitter);
2. The length of the macrocycle can induce a memory size problem, since the table parameters must be stored in the BA. For instance, if periodicities of IPH4 and IPH5 were, respectively, 5 and 7, the length of the macro-cycle would be 420 microcycles instead of only 12.

Both the communication jitter and memory size problems have been addressed in the literature. In [21] 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 [18,19], 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.

In our case there are additional aspects which triggered us to investigate other possibilities. Firstly, the problem of communication jitter (as defined in [20]) has not the same relevance. In fact the scheduler will schedule IP fragments and for obvious reasons it is not of paramount importance to schedule each of them in a strict periodic fashion. Secondly we want to consider transactions with dissimilar durations. Thirdly, it is of paramount importance that TIPH is minimised, since its length may impact the timing requirements of the fieldbus native control-related traffic (NHP in Fig. 3). Finally, streamed TCP/IP applications have particular timing behaviour, which must be taken into account by the scheduler.

IV. SCHEDULING POLICIES FOR THE IPAS

A possible schedule to the stream set in Table 1 could be the one shown in Fig. 4. Basically this schedule corresponds to a rate monotonic approach: it schedules all streams to be transferred on the first microcycle and then in subsequent microcycles according to the periodic pattern of the stream.

This schedule can be provided as a table to the scheduler, which at each time it executes (each T_{IPCY}) will process a column. Alternatively, the scheduler determines each time it runs (each T_{IPCY}) the actual schedule for that cycle (on-line scheduler, as illustrated in [15]).

One of the main drawbacks of the RM approach is that the minimum value for T_{IPH} must be equal to the sum of the duration of all transactions. And the value for T_{IPH} can eventually collide with the requirements for the control-related traffic: high values for T_{IPH} can be incompatible with short values for T_{WCCT} (see Fig. 3) required by the NHP traffic.

Minimising T_{IPH} can be achieved if the deferred release algorithm proposed in [20] for the WorldFIP case is adapted for the IPAS case, as described in [15]. Table 2 represents the schedule as shown in Fig. 4.

Table 2 – Schedule with the RM approach

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		1		1	
IPH3	1				1				1			
IPH4	1						1					
IPH5	1						1					
Load (ms)	4.1	1.3	2.4	1.3	2.7	1.3	3.8	1.3	2.7	1.3	2.4	1.3

This schedule can be improved to reduce maximum microcycle load (thus allowing a smaller value for T_{IPH}). If the pattern of both IPH2 and IPH3 are moved one microcycle ahead and the pattern of IPH5 is moved three microcycles ahead, the maximum value of a microcycle load will be reduced to 2.9 ms, as shown in Table 3. If the maximum pattern shift allowed corresponds to the stream period, there is not any negative impact in the resulting schedule, since both the stream data rate per macrocycle and periodicity between transfers (in terms of cycles) are maintained.

Table 3 – Schedule with the deferred release approach

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		1		1	
IPH3		1				1				1		
IPH4		1						1				
IPH5				1						1		
Load (ms)	2.4	1.7	2.4	2.6	2.4	1.6	2.4	1.4	2.4	2.9	2.4	1.3

Due to the dissimilar values we are considering for the transaction durations (Table 1), an improvement in the direction of the minimisation of T_{IPH} can be made by modifying the deferred release algorithm in a way that it starts to consider first the streams with bigger transaction durations rather than using a frequency ordering.

An offline version of the algorithm is presented below.

```
function deferred_realease__with_size_order;
input:
   niph    /* number of IPH flows */
   tp[i]   /* vector containing the periodicity of the fragments */
           /* ORDERED by trans. duration; i ranging from 1 to niph */
   cp[i]   /* contains the transaction duration of the fragments */
   Tipcy   /* value for T_IPCY, which is also the scheduler cycle */
   Mcy     /* number of cycles in the macrocycle */

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

begin
1:  /* obtains the shift */
2:  for i = 1 to niph do
3:     min_load = MAXINT;
4:     for cycle = 1 to (tp[i] div Tipcy)
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 Tipcy)
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 */
23: /* builds the schedule */
24:  repeat
25:     load[cycle] = load[cycle] + cp[i];
26:     sched[i,cycle] = 1;
27:     cycle = cycle + (tp[i] div Tipcy);
28:  until cycle > Mcy;
29:
30:  /* obtains the value for T_IPH */
31:  tiph = 0;
32:  for i = 1 to Mcy do
33:     if load[i] > tiph then
34:        tiph = load[i];
35:     end if;
36:  end for;
return sched, offset, tiph;
```

With this algorithm, the schedule (table) which is obtained is as shown in Table 4.

The maximum load value is down to 2.7 ms and the utilisation (considering T_{IPH} = 2.7 ms) rises up to 79.9% (compared with the 52% as given by the schedule of Table 2). From our experimental observation we found that size order with deferred release not always leads to a smaller value of the maximum load. In some specific cases the rate order leads to a smaller value.

Table 4 – Deferred release with size ordering

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		1		1
IPH3		1				1				1		
IPH4			1						1			
IPH5	1						1					
Load (ms)	2.6	2.7	1.4	2.4	1.3	2.7	2.6	2.4	1.4	2.7	1.3	2.4

V. ALLOWING "COMMUNICATION JITTER"

One of the functions of the IPAS is to fragment/de-fragment IP packets. Typically fieldbus frame length are much smaller than IP packets. Thus if the fragments are sent with a variable interval between them the timing characteristics of the IP packet are still retained.

Therefore, while in multicycle scheduling for WorldFIP networks minimising communication jitter is a major concern, in the case of the IPAS scheduler "allowing communication jitter" may be an advantage, for instance if minimising the value for T_{IPH} is an objective.

We propose now an algorithm in which IP fragments are allowed to have microcycle level jitter, meaning that the schedule of each fragment can be made with a displacement of some microcycles as compared to its schedule when the deferred release method is used.

Below we describe an algorithm that permits defining the allowable jitter (in number of microcycles) for each stream.

```
function deferred_release_size_order_w_jitter;
input:
   niph    /* number of IPH streams */
   tp[i]   /*vector containing the periodicity of the fragments*/
   cp[i]   /* contains the transaction duration of the fragments*/
   jitter[i] /* Maximum allowable jitter in multiples of Tipcy */
   Tipcy   /* value for T_IPCY, which is also the scheduler cycle */
   Mcy     /* number of cycles in the macro-cycle */

outputs:
   sched[i,cycle] /* Generated Schedule */
   tiph    /* value for the T_IPH parameter */

begin
1:  for i = 1 to niph do
2:    'Searches the the offset where tiph is minimised
3:    load_offset = MAX_REAL;
4:       for offset = 0 to tp[i] - 1
5:          load_jitter = 0;
6:          k=0;
7:          /* r – number of releases for stream i */
8:          for r = 0 to (Mcy / tp(i) - 1)
9:             k= k + 1;
10:            seq_aux[k] = best_pos_w_jitter(r, offset, tp[i],
11:            jitter[i], Mcy, load);
12:            If load(seq_aux[k]) > load_jitter then
13:                  load_jitter = load(seq_aux[k]);
14:            end If;
15:         end for;
16:
17:         If load_jitter < load_offset then
18:            load_offset = load_jitter
19:            offset_aux = offset
20:            for f = 1 to k
21:               seq[f] = seq_aux[f]
22:            end for;
23:       end If;
```

```
24:    end for;
25:
26:    /* Scheduling */
27:    for r = 1 to (Mcy / tp(i) )
28:        sched(i, seq[r]) = sched (i, seq[r]) + 1;
29:        load[seq[r]] = load[seq[r]] + cp[i]
30:    end for;
31: end for;
32: /* Determination of Tiph */
33: for cycle = 1 to Mcy
34:    If load[cycle] > tiph then
35:        tiph = load[cycle]
36:    end if;
37: end for
return sched, tiph
```

```
function best_pos_w_jitter;
input:
   r        /* release of stream */
   offset   /* offset where to apply the test */
   tp       /* periodicity of stream i */
            /* i ranging from 1 to niph */
   jitter   /* micorcycle jitter for stream i*/
   Mcy      /* number of cycles in the macro-cycle */

outputs:
   best_pos      /* nest position where the load is minimised */

begin
1: end_cycle = (r * tp + 1 + jitter + offset);
2: start_cycle = (r * tp + 1 - jitter + offset);
3: If end_cycle > Mcy then
4:    end_cycle = Mcy;
5: end If;
6: if start_cycle < 1 then
7:    start_cycle = 1;
8:  end If;
9: load_min = MAX_REAL;
10: for cycle = (r * tp + 1 + offset) to end_cycle
11:     If load[cycle] < load_min then
12:         load_min = load(cycle);
13:         cycle_pos = cycle;
14:     end if;
15: end for;
16: for cycle = start_cycle to (r * tp + 1 + offset)
17:     if load[cycle] < load_min then
18:         load_min = load[cycle];
19:         cycle_pos = cycle;
20:     end If;
21: end for;
return cycle_pos;
```

Applying this algorithm to the stream set of Table 1, and in which the jitter allowed for each stream equals 1, the resulting schedule will be as shown in Table 5.

Table 5 – Allowing Jitter in the Deferred release with size ordering

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	1		1	
IPH3				1		1						1
IPH4				1						1		
IPH5	1						1					
Load (ms)	2.6	2.4	2.4	1.7	2.4	1.6	2.6	2.4	2.4	1.4	2.4	1.6

The introduction of microcycle level jitter of 1 to all streams allows decreasing the maximum microcycle load to 2.6 ms.

For the particular case of the stream set shown in Table 1, no further improvement is possible if the allowed jitter is increased. Just as another example consider the following stream set (period, transaction duration): {{2; 1.7}; {9; 1.5}; {8; 1.4}; {8 1.3}; {6; 0.9}; {3; 0.5}; {6; 0.4}}. In Fig. 5 we represent the variation of the maximum cycle load (T_{IPH}) as a function of the allowed jitter (equal to all streams).

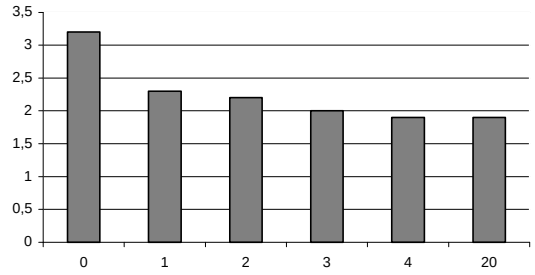

Fig. 5 Maximum microcycle load as a function of allowed microcycle jitter

VI. GUARANTEEING CONSTANT BANDWIDTH

The scheduling algorithms we have been describing so far are static algorithms, in the sense that the resulting schedule will be always the same in every macrocycle. This applies to both on-line and off-line (table provided to the scheduler) versions of the algorithms.

The idea of the algorithms is to provide constant bandwidth to multimedia applications, which correspond to the required QoS of the application. It is also important to stress that, due to the fragmented nature of IP data, the algorithms are also able to provide reduced jitter and delay to TCP/IP applications.

However, it is not true that the arrival pattern (to the related IPH queues) of the IP fragments is synchronous with the scheduling policy, far from that. This exactly means that even if some fragments are scheduled to be dispatched by the TMS sub-layer in a specific scheduler cycle (microcycle), actual transaction does not take place simply because the queue is empty. And this may happen in a number of consecutive cycles of the scheduler. The reason for this bursty behaviour resides not only in the timing behaviour of the applications but also in the timing behaviour of the operating system and, importantly in the timing behaviour of the fragmentation functionality of the IPAS. Actually, and speaking of multimedia applications, there are quite a few which will present a variable bandwidth behaviour [22]. These will not be addressed in this paper. The focus here is how to adapt the scheduling mechanisms to the pattern arrival behaviour of fragments to FIFO queues. In this section we propose "compensating" algorithms to guarantee constant bandwidth even with some bursty behaviour in the arrival pattern of IP fragments.

Theoretically speaking, compensation is only possible if there is available "spare time" in the microcycles of the schedule. For instance, and concerning Table 5, if T_{IPH} is set equal to 2.6 ms, the largest spare slot will be 1.2 ms in cycle 10, if all 5 multimedia streams are active. This may pose some problems in actually compensating "dispatching misses" concerning both IPH1 and IPH5, which in both cases have a transaction duration of 1.3 ms.

Therefore, the value for T_{IPH} may need to be traded-off to taking into account compensation. While in Section IV we discussed scheduling policies which tried to minimise the value for T_{IPH}, we can now put forward another possible requirement for the value of T_{IPH}: $T_{IPH} > min\_load + \max\{C_i\}$, where *min_load* corresponds to the

microcycle with the smallest load within a macrocycle and max{C_i} corresponds to the largest transaction duration of all streams. As it is easy to understand, scheduling policies that minimise T_{IPH} tend to result in a uniform load distribution among cycles. Therefore, when taking into account compensation requirements in the scheduling algorithms T_{IPH} can result higher if the scheduling policy of Table 5 is used instead of the one leading to Table 2.

A. An illustrative Example of the Problem

Take for example the schedule as shown in Table 5. Following the criteria as described above T_{IPH} would be set in the station equal to 2.7 ms (1.4 + 1.3). Assume that in one of the cycles both IPH1 (in the 1st cycle) and IPH3 (in the 4th cycle) queues are empty. The actual dispatching will be as shown in Table 6. This example will be used throughout this section to discuss diverse approach on how to provide on-line compensation to the scheduling policies in order to guarantee constant bandwidth to the multimedia streams.

Table 6 – Example of "missed dispatching"

Cycle	1	2	3	4	5	6	7	8	9	10	11	12
IPH1	0	1	1	1	1	1	1	1	1	1	1	1
IPH2		1	1		1			1	1		1	
IPH3				0		1						1
IPH4				1						1		
IPH5	1						1					
Load (ms)	1.3	2.4	2.4	1.4	2.4	1.6	2.6	2.4	2.4	1.4	2.4	1.6

B. A Compensation Algorithm

The algorithm described in this subsection is to run online, either the schedule is provided to the scheduler as a table or the scheduler determines each cycle schedule online.

When a scheduled fragment is not dispatched (the related queue is empty) a counter ("missed hits" counter) for the stream is incremented. The algorithm will try to schedule missing fragments in the first cycle with available spare time. Several different policies can be used in the case of several streams with missing fragments. For instance, compensation priority could be on the base of transaction duration. Streams with larger transaction duration would be compensated first once that this type of streams is scheduled with more difficulty than streams with smaller transaction times.

Alternatively inverse rate priority could be used. The rational for this is that 1 missed hit out of 2 in a macrocycle has more impact in the QoS than 1 missed hit out of 5 in a macrocycle. This is a static policy that can be improved by a dynamic one based on the quotient between the number of missed hits and the number of expected dispatches per macrocycle, thus constituting a number of relative misses criterion.

For obvious reasons the counters for the missing hits must be upper-bounded: missing hits occur while the multimedia stream is not active at all. The question is how to define the upper bound for each stream?

We consider that on average each IP packet (and their fragments) arrives to the related IPAS queue with a period of T_i^{IP} and with a jitter of J_i^{IP} (Fig. 6). This jitter includes variable latencies related to the generating multimedia application, to the operation system execution and fragmentation.

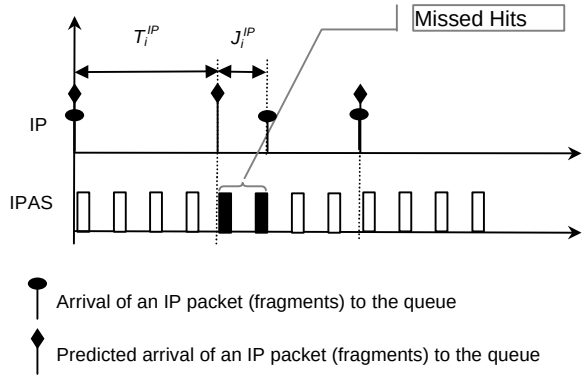

Fig. 6 Illustration how jitter in pattern arrival of fragments to a queue can induce "missed hits" in the scheduler

The value of N_i^{missed} can be set equal to the number of fragments that could be missed during the time span corresponding to the value of predicted J_i^{IP}:

$$N_i^{missed} = \left\lceil \frac{J_i^{IP}}{T_i} \right\rceil \quad (1)$$

The algorithm (using the number of relative misses criterion) is described below in pseudo-code.

```
function Dispatcher_w_Basic_Comp;
input:
  niph      /* number of IPH flows */
  tp[i]     /* vector containing the periodicity of the fragments */
            /* ORDERED by size; i ranging from 1 to niph */
  cp[i]     /* contains the transaction duration of the fragments */
  Tipcy     /* value for T_IPCY, which is also the scheduler cycle */
  Mcy       /* number of cycles in the macro-cycle */
  tiph      /* value for the T_IPH parameter */
  sched[i, c]     /* Tabe with the offline schedule */
                  /* c ranging from 1 to Mcy */
  comp_lim[i]     /* maximum number o missed releases that */
                  /* can be compensated for stream i. */

/* The algorithm runs every T_IPCY */

begin
1:  if start = TRUE then
2:    cycle = 0
3:    for i = 1 to niph
4:      req[i] = Mcy / tp[i];
5:      req_fail[i] = 0;
6:      num_disp[i] = 0;
7:      num_sch[i] = 0;
8:    end for;
9:    start = FALSE;
10: end if;
11: cycle = cycle +1;
12: load_a = 0
13:  if cycle = Mcy then
14:     cycle = 1
15: end if;
16: /* Scheduling according to the offline entry table */
17: for i = 1 to niph
18:     for f = 1 to sched[i, cycle]
19:        if get_queue(i) = 1 then
20:           load_a = load_a + cp[i]
21:           num_disp[i] = num_disp[i] +1
22:        end if;
23:        num_sch[i] = num_sch[i] +1
24:     end for;
25: end for;
```

```
26: /* Compensation */
27: for i = 1 to niph
28:     req_fail[i] = req_fail[i] + (num_sch[i] – num_disp[i]);
29:     if req_fail[i] > comp_lim[i] then
30:         req_fail[i] = comp_lim[i]
31:     end if;
32:     num_sch[i] = 0
33:     num_disp[i] = 0
34: end for;
35: /* Returns a vector ordered: the stream more far way form its */
36: /* target data-rate first. */
37: ordered_streams = obtains_sched_order(req, req_fail);
38: for p = 1 to niph
39:     i = ordered_streams(p);
40:     if req_fail[i] <> 0 then
41:         for h = 1 to req_fail[i]
42:             if (load_a + cp[i]) < tiph then
43:                 if get_queue(i) = 1 then
44:                     load_a = load_a + cp[i];
45:                     req_fail[i] = req_fail[i] -1
46:                 end if;
47:             end if;
48:         end for
49:     end if;
50: end for;
return
```

To illustrate how the algorithm would handle the situation illustrated in Table 6, assume that fragments for IPH1 and IPH3 arrive just before microcycle 10. Compensation for IPH3 can be made in microcycle 10 of the current macrocycle (Table 7) whereas for IPH1 it will only be made in microcycle 10 of the next macrocycle (Table 8).

Table 7 – Compensation of IPH3 on the current macrocycle

Cycle	1	2	3	4	5	6	7	8	9*	10	11	12
IPH1	0	1	1	1	1	1	1	1	1	1	1	1
IPH2		1	1		1			1	1		1	
IPH3				0		1				1		1
IPH4				1						1		
IPH5	1						1					
Load (ms)	1.3	2.4	2.4	1.4	2.4	1.6	2.6	2.4	2.4	1.7	2.4	1.6

Table 8 - Compensation of IPH1 on the next macro cycle

Cycle	1	2	3	4	5	6	7	8	9	10	11	12
IPH1	1	1	1	1	1	1	1	1	1	2	1	1
IPH2		1	1		1			1	1		1	
IPH3				1		1						1
IPH4				1						1		
IPH5	1						1					
Load (ms)	2,6	2,4	2.4	1.7	2.4	1.6	2.6	2.4	2.4	2.7	2.4	1.6

VII. CONCLUSIONS

To enable industrial multimedia applications the TCP/IP suite of protocols can be integrated with a fieldbus stack, leading to a dual-stack approach. In this paper we have briefly outlined how this dual stack approach can be achieved for the particular case of PROFIBUS fieldbus networks.

IP packets are to be encapsulated within fieldbus data frames, typically requiring that IP packets are fragmented/ de-fragmented. Each station have to support a number of IP flows each one having particular QoS requirements, namely bandwidth and allowed jitter. In this paper we have particularly focused the issue related to how properly schedule the different IP fragments in order to guarantee quasi-constant bandwidth to the multimedia applications.

Research that is currently in progress is addressing the issue of providing QoS guarantees to multimedia applications with variable bandwidth requirements.

VIII. REFERENCES

[1] G. Lenhart, “A Fieldbus Approach to Local Control Networks”, *Advances in Instrumentation and Control*, vol. 48, no. 1, 1993, pp. 357-365.

[2] RFieldbus Deliverable D1.3, “General System Architecture for the RFieldbus System”, Technical Report, Sep. 2000.

[3] Pacheco, F., Tovar, E., Kalogeras, A and Pereira, N., “Supporting Internet Protocols in Master-Slave Fieldbus Networks”. To appear in Proceedings of 4th IFAC FET Conference, 2001.

[4] EN 50170, “General Purpose Field Communication System. EN 50170-2 (PROFIBUS)”, 1996.

[5] EN 50170, “General Purpose Field Communication System. EN 50170-3 (WorldFIP)”, 1996.

[6] EN 50170, “General Purpose Field Communication System. EN 50170-1 (P-Net)”, 1996.

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

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

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

[10] IEEE, “IEEE Standard 802.4: Token Passing Bus Access Method and Physical Layer Specification”, IEEE, 1985.

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

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

[13] Montuschi, P., Ciminiera. L. and Valenzano, A., “Time Characteristics of IEE802.4 Token Bus Protocol”, IEE Proceedings, Vol. 139, No. 1, pp. 81-87, 1992.

[14] Zheng, Q. and Shin, K., “Synchronous Bandwidth Allocation in FDDI Networks”, IEEE Transactions on Parallel and Distributed Systems, Vol. 6, No. 12, pp. 1332-1338, 1995.

[15] E. Tovar, F. Pacheco, F. Vasques and L. Ferreira, “Industrial Multimedia over Factory-Floor Information Networks”, to appear in the 10th IFAC INCON Conference, 2001.

[16] Alves, M., Tovar, E. and Vasques, F., “On the Adaptation of Broadcast Transactions in Token-Passing Fieldbus Networks with Heterogeneous Transmission Media”, to appear in the 4th IFAC FET Conference, 2001.

[17] Alves, M., Tovar, E. and Vasques, F., “Evaluating the Duration of Message Transactions in Broadcast Wired/Wireless Fieldbus Networks”, to appear in the 26th IEEE IECON Conference, 2001.

[18] Kumaran, S. and J.-D. Decotignie, “Multicycle Operations in a Field Bus: Application Layer Implications”, Proceedings of IEEE Annual Conference of Industrial Electronics Society (IEEE) 531-536, 1989.

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

[20] Tovar, E. and Vasques, F., “Distributed Computing for the Factory-floor: a Real-Time Approach Using WolrdFIP Networks”, Computers in Industry, Vol. 44, No. 1, pp. 11-30, 2000.

[21] Kim, Y., Jeong, S. and W. Kown, “A Pre-Run-Time Scheduling Method for Distributed Real-Time Systems in a FIP Environment”, Control Engineering Practice, Vol. 6, (Pergamon) 103-109, 1998.

[22] Knops, L., “Microsoft NetMeeting 3 Resource Kit”, Microsoft Corporation, 1999.