On Real-Time and Non Real-Time Distributed Computing

G. Le Lann

INRIA - BP 105
78153 Le Chesnay Cedex, France
E-mail : Gerard.Le_Lann@inria.fr

Abstract. In this paper, taking an algorithmic viewpoint, we explore the differences existing between the class of non real-time computing problems ($\aleph\Re$) versus the class of real-time computing problems ($\Re$). We show how a problem in class $\aleph\Re$ can be transformed into its counterpart in class $\Re$. Claims of real-time behavior made for solutions to problems in class $\aleph\Re$ are examined. An example of a distributed computing problem arising in class is studied, along with its solution. It is shown why off-line strategies or scheduling algorithms that are not driven by real-time/timeliness requirements $\Re$ are incorrect for class $\Re$. Finally, a unified approach to conceiving and measuring the efficiency of solutions to problems in classes $\aleph\Re$ and $\Re$ is proposed and illustrated with a few examples.

1 INTRODUCTION

Over the last 20 years, the distributed algorithms community has spent considerable effort solving problems in the areas of serializable or linearizable concurrent computing. These problems belong to a class denoted $\aleph\Re$, the class of non real-time computing problems.

Separately, over the last 30 years, the real-time algorithms community has spent considerable effort solving scheduling problems in the area of centralized computing, considering preemptable and non-preemptable resources. Only recently has distributed computing received some attention. These problems belong to a class denoted $\Re$, the class of real-time computing problems.

In this paper, taking an algorithmic viewpoint, we embark on exploring the differences between both classes. The scope of this paper is restricted to that of deterministic algorithms. Besides intellectual interest, investigation of these issues is expected to bring the two communities closer, via the clarification of a few essential concepts. In particular, we have observed that the qualifiers "real-time" and "distributed" may not carry the same meaning in both communities. Broadly speaking, with few exceptions, the distributed algorithms community does not consider problem specifications that include real-time requirements. Conversely, the real-time algorithms community often fails to understand what is implied with considering a distributed computing problem.

The remainder of this paper is organized as follows. In section 2, we introduce the distinctive attributes of class $\Re$. In section 3, we examine claims of real-time

behavior made for solutions to problems in class $\aleph\Re$. In section 4, we introduce a problem of distributed computing that belongs to class $\Re$, along with its solution. A unified approach to conceiving and measuring the efficiency of solutions to problems in both classes is proposed in section 5.

In this paper, we have purposedly put more emphasis on problems and solutions in class $\Re$, given the intended audience.

2 CLASS $\Re$ VERSUS CLASS $\aleph\Re$ PROBLEMS

Quite often, it is believed that "real-time" means "fast". For example, the distinction between "meeting specific time bounds that are specified a priori" (which is a metric-free problem) on the one hand, and "observable response times are small" (which is an implementation dependent consideration) on the other hand, does not seem to be well understood. For example, it has been recently stated that "... protocols like Isis, Horus, Transis and Totem are real-time protocols if one is concerned with real-time deadlines of minutes or tens of seconds...".

As these protocols – and the related Asynchronous Consensus problem – are reasonably well understood, it seems appropriate to use them to carry out an analysis in order to explain why statements such as the one reported above are meaningless.

We will first introduce some notations, and then present the distinctive attributes of problems in class $\Re$, before proceeding with the analysis in section 3.

2.1 Notations

Correctness Recall that, in the context of this paper, a solution is an algorithm and its associated models and proofs. The specification of any problem P comprises the following subsets :

- the specification of some assumptions, denoted λ
- the specification of the properties sought, denoted Λ.

Assumptions are equivalent to axioms in Mathematics. Essentially, assumptions cover the models considered, namely the computational models, the failure models, the models of event releases. Examples of properties of interest are safety, liveness, timeliness. Similarly, the specification of any solution A comprises the following subsets, in addition to the specification of A :

- the specification of some assumptions, denoted γ,
- the presentation of the proofs that some properties Γ hold, given γ.

Whenever a model or a set of properties x_1 dominates another model or another set of properties x_2, i.e. x_1 is more general than x_2, this will be denoted by $x_1 \supseteq x_2$, $\supseteq$ being the inclusion symbol. It is said that A solves problem P if the following correctness conditions hold :

$[cc_1]$: $\Gamma(A) \supseteq \Lambda(P)$ and $[cc_2]$: $\gamma(A) \supseteq \lambda(P)$.

For most problems, establishing correctness conditions entails the expression of feasibility conditions, denoted [fc].

A useful interpretation of these correctness conditions is as follows : with A, a system is proved to be endowed with properties that are at least equal to (as strong as) those required, assuming advance knowledge that is at most equal to (not "richer" than) that given in the exposition of P.

These conditions may seem trivial. Nevertheless, quite surprisingly, many published papers describe results that violate $[cc_1]$ or $[cc_2]$ or both. Typically, some papers explore "tradeoffs" between various solutions, and conclude with some particular decision such as A_1 is "better" than A_2, because A_1 is less "costly" than A_2, totally ignoring the fact that A_1 is less "costly" for the sole reason that A_1 is based on postulating g1, which violates $[cc_2]$ whereas A_2, assumed for A_2, does not. Such meaningless analyses are commonplace when off-line solutions are compared with on-line solutions for problems in class $\Re$.

Very often, it is useful to view the models included in λ as defining an adversary that is endowed with some bounded power. For example, the adversary contemplated with non-public concurrent data structures is more restricted than the one embodied in problems involved with public concurrent data structures [5]. Similarly, periodic or sporadic releases models characterize adversaries that are more constrained that those embodied in arbitrary event releases models.

Optimality Informally, an algorithm A is optimal for a given problem if, (i) whenever the desired properties L can hold, they do hold via A, and if, (ii) A not being able to enforce these properties, there cannot exist an algorithm that would enforce them, given λ. More precisely, a correct algorithm A is optimal for a given problem if the following optimality condition holds :

[oc]: the [fc] under which $[cc_1]$ and $[cc_2]$ hold true are necessary and sufficient.

Let us give a few examples of necessary and sufficient feasibility conditions, denoted NS[fc]. In class $\Re$, examples of NS[fc]s are, (i) those given with the proof that the centralized non-preemptive earliest-deadline-first algorithm is optimal for periodic and sporadic arrival laws, when relative deadlines are equal to periods [15], (ii) those given with the proof that the centralized D-Over algorithm is optimal for aperiodic arrival laws in the presence of overload [3].

In class $\aleph\Re$, examples of NS[fc]s are, (i) the $3t+1$ lower bound for sustaining up to t arbitrary failures in synchronous computational models [20], (ii) the $\diamond W$ unreliable failure detector semantics for solving the consensus problem in the presence of crash failures in asynchronous computational models [6].

2.2 The distinctive attributes of class $\Re$

What makes a problem belong to class $\Re$ rather than to class $\aleph\Re$? There is no general agreement on the answer. What follows in an attempt to clarify the issue. Two attributes seem to be necessary and sufficient.

(i) A first distinctive attribute of a real-time computing problem is the presence, in its specification, of arbitrary and individual time bounds to be met by every operation that can be performed by every process.

In the real-time computing community parlance, such time bounds are often referred to as timeliness constraints (earliest/latest deadlines for termination, relative to release times, bounded jitters, linear or non-linear functions of system's parameters, etc.). They are the expression of a property that is essential and specific to this class of problems, that of timeliness. Timeliness is a composition of a safety property (it should never be the case that specified time bounds are not met) and a liveness property (progress is mandatory). Operations performed by processes are triggered by the occurrence, also called the release, of events. Timeliness properties cannot be achieved for unbounded densities of event releases. Hence :

(ii) A second distinctive attribute of a real-time computing problem is the presence, in its specification, of an event releases model.

In the distributed algorithms community, "real-time" is sometimes equated with considering synchronous or timed transitions computational models (in contrast with considering partially synchronous or asynchronous or fair transitions computational models). We believe it is essential to understand that, whatever computational models are considered in λ, it is the presence or the absence of timeliness constraints in Λ that determines whether a problem belongs to class $\Re$ or to class $\aleph\Re$ (respectively).

According to the above, we argue that some papers, such as e.g. [2], do not address real-time algorithmic issues. The problems considered in these types of papers consist in demonstrating that some time independent safety property – such as, e.g. mutual exclusion – is achieved for some [fc] to be met by the synchronous computational model assumed. These problems are not equivalent to those where it is asked to demonstrate that specific and arbitrarily chosen timeliness constraints – such as, e.g., strict relative deadlines to be met by the competing processes – are satisfied for some [fc] to be met by the models considered in λ, namely the event releases model and the – possibly synchronous – computational model.

To summarize, the distinctive attributes of any problem in class $\Re$ are as follows :

- specification set Λ includes a subset, denoted $\Lambda_{\Re}$, that specifies timeliness constraints
- specification set λ includes a subset, denoted $\lambda_{\Re}$, that specifies an event releases model.

Real-time computing problems either are decision problems or are optimization problems. Decision problems arise whenever it required to meet $\Lambda_{\Re}$ while $\lambda_{\Re}$ is not violated. The [fc]s serve the purpose of telling whether or not a real-time computing problem is feasible. Such problems are often referred to as "hard real-time" problems. Optimization problems arise whenever it is accepted or anticipated that $\lambda_{\Re}$ may be violated (e.g., "overloads"). In such cases, some of the

timeliness constraints specified in $\Lambda_{\Re}$ cannot be met. Value functions are then defined for every computation. Such functions can be constants or functions of times of (computation) termination. Optimization consists in maximizing the accumulated value for every possible run – to be derived from $\lambda_{\Re}$ – or, equivalently, to minimize a value loss (e.g., minimum regret). Such problems are often referred to as "soft real-time" problems.

3 CLASS $\aleph\Re$ AND REAL-TIME COMPUTING

3.1 The Asynchronous Consensus problem

The Asynchronous Consensus problem, denoted [AC], is in class $\aleph\Re$. [AC] is the following problem : $\lambda \equiv$ a group of n processors, asynchronous computational model, up to f processor crashes, reliable message broadcast, processors have arbitrary initial values $\Lambda \equiv$ all correct processors eventually decide (termination); they decide on the same final value (agreement) ; that final value must be the initial value of some correct processor (non triviality).

Over the last 10 years, a great deal of research has been devoted to circumventing a famous impossibility result [11]. A significant number of papers contain descriptions of algorithms and extensions to λ aimed at showing how [AC] can become tractable. Let $\delta(\lambda)$ be the assumptions that need be made, in addition to λ, in order to solve [AC].

3.2 How not to solve [AC]

With few exceptions, published solutions exploit the idea of augmenting the original asynchronous model with physical or logical timers. For example, this is the approach followed to implement Atomic Broadcast in such systems as Isis, Horus, Transis or Totem. Atomic Broadcast is equivalent to [AC] in asynchronous models.

It is reasonably obvious that timers are of no help. Timers that would be arbitrary timers are not ruled out by the original asynchronous model. Even if $\delta(\lambda)$ = perfect timers, it has been shown that [AC] cannot be solved [9]. Therefore, approaches based on $\delta(\lambda)$ = arbitrary timers cannot solve [AC] either. The reason why such approaches fail simply is that the $\delta(\lambda)$ considered does not bring in more common knowledge than what is provided by the original l. Consequently, the impossibility result still applies fully.

The analysis of existing "solutions" reveals that two categories are considered, namely the primary-partition category and the multi-partition category. Simple adversary arguments can be developed to show that any of these "solutions" either violates the termination requirement of Λ (e.g., because of unjustified exclusions) or violates the agreement requirements of Λ (e.g., because any two partitions reach different decisions). Some "solutions" assume that some group membership service GMS is available. Thanks to GMS, [AC] is solved with no extra $\delta(\lambda)$. But this is a violation of correctness condition $[cc_2]$. In many instances,

assuming GMS is tantamount to assuming that [AC] is solved, which yields circular "solutions". Furthermore, it has been demonstrated that primary-partition GMS cannot be solved in asynchronous models [7].

To summarize, [AC], or any problem equivalent to [AC], has no deterministic solution that would be based on $\delta(\lambda)$ = timers. It follows trivially that claims of "real-time behavior", even in the order of centuries, are totally unfounded.

The evidence that timer-based approaches can only solve [AC] probabilistically is being acknowledged more openly than was the case previously. Having admitted this, some scientists develop the following "argument". The behavior of any real system can only be predicted with some probability – including real-time behavior. Therefore, everything being only probabilistically true, those deterministic algorithms that solve [AC] only approximately can be considered as yielding "sufficiently good" real-time behavior. Unfortunately, this superficial argument is flawed (this is discussed further in section 4.2.6). At best, this argument is void. Indeed, the real-time problem that we are told is solved is not even defined, as specifications $\Lambda_{\Re}$ and λ_{Re} are not given.

3.3 How to solve [AC]

Conversely, very few papers describe provably correct solutions to [AC]. The concept of unreliable failure detectors was first introduced in [8]. It was subsequently demonstrated that the completeness and the accuracy properties that define $\diamond W$ are the NS $\delta(\lambda)$ under which [AC] can be solved [6]. Hence, correct and optimal solutions are available.

An interesting question to ask is whether a real-time extension of [AC] could be solved with some real-time extension of $\diamond W$. This also raises the question as whether asynchronous computational models can be considered in class $\Re$. This discussion is deferred to section 5.

Let us now examine a problem in class $\Re$ and its solution. Before doing so, we will first re-state the well-known principles of partial advance knowledge and partial common knowledge that characterize problems in distributed computing. This is felt useful mainly for the reason that we keep seeing papers aimed at solving "distributed real-time computing" problems that violate these basic principles.

4 CLASS $\Re$ AND DISTRIBUTED COMPUTING

4.1 Partial knowledge

A distinctive attribute of any problem in distributed computing is incomplete information, or partial knowledge. If we look at the classes of problems considered by the distributed algorithms community over the last 20 years, we find such computational models as synchronous, partially synchronous and asynchronous models and such failure models as crash, timing and arbitrary failures, which induce a significant amount of uncertainty w.r.t. future system runs. Hence,

a first principle is that of partial advance knowledge (of future system runs). Furthermore, distributed computations have to cope with an additional source of uncertainty, that is lack of knowledge of the current system state (even if the most conservative models are assumed). Hence a second principle is that of partial common knowledge (shared by the processors).

Consider a set of processors involved in some distributed computation. At best, via some algorithm, some processors may end up sharing some common knowledge about some partial past system state. Therefore, a distributed algorithm may be viewed as the union of two algorithms, one in charge of building/maintaining some common knowledge, referred to as the dissemination algorithm, the other one in charge of acting on the system state, referred to as the decision algorithm. Let κ be a measure of the common knowledge accessible to processors, as achieved by the dissemination algorithm. Of course, κ is a (possibly complex) function of the number of the different shared partial states as well as of the number of processors sharing each partial state. Let $C(\kappa)$ be the cost of obtaining κ. Cost may be measured in various ways, e.g. a number of steps (of the dissemination algorithm). In general, C is a monotonically increasing positive function of κ. Theoretically, κ ranges between 0 (0-common knowledge) and K, the best achievable approximation of full-common knowledge.

If we look at the classes of problems considered by the real-time algorithms community over the last 30 years, we mainly find event releases models such as periodic and sporadic releases and timeliness constraints that are expressed as simple linear functions of periods (equality, very often). Furthermore, most systems considered are centralized (typically, single-processor systems or centralized multiprocessors). These models do not fit well with those considered for distributed computations. Hence the growing recognition that more general releases models should be investigated, such as aperiodic or arbitrary event releases models, as well as more general properties such as arbitrary timeliness constraints. Clearly, such models induce a significant amount of uncertainty w.r.t. future system runs. However, given the principles of partial advance knowledge and partial common knowledge, it is not at all clear that these more general releases models yield increased uncertainty compared to that resulting from considering a distributed computational model.

In the recent past, we have noticed that these more general models and objectives are sometimes perceived as being "unnecessarily complicated". It is quite surprising that such views can be taken by scientists who address real-time distributed computing issues. Indeed, these more general models and objectives reflect reality more accurately than the good old models (periodic/sporadic releases, time bounds related to periods). Note that the event releases models in $\aleph\Re$ and the timeliness constraints in $\Lambda_{\Re}$ are specified by the "clients" (e.g., the end users of the systems to be designed). As these systems can only start operating in the future, specifying these models and constraints is tantamount to predicting the future behavior of the system's environment. Which client would be foolish enough to pretend that every possible external event can only be released periodically or sporadically for the next 10 years of operation and that the appropriate

relative deadlines must always be equal to the periods ? Such assumptions are clearly unacceptable in the case of critical applications or whenever the environment is, by nature, non-cooperative. Condition $[cc_2]$ makes it mandatory for designers to consider assumptions g that do not artificially weaken the problem under consideration. Hence, any solution to a real-time distributed computing problem that would be based on clairvoyance assumptions or, more generally, on a violation of condition $[cc_2]$, is a non-solution. Let us refine this condition.

When considering a given problem P, the related λ yields the following two bounds on κ :

- $\kappa(\lambda)$, the upper bound on κ that is accessible at cost $C(\kappa(\lambda)) = 0$;$\kappa(\lambda)$ is a measure of partial advance knowledge as embodied in λ,
- $\kappa(\lambda)$, the upper bound on κ that is achievable at some non-zero cost by a dissemination algorithm (given the models considered in λ).

For example, for problem [HRTDM] considered in section 4.2, we would have $\kappa(\lambda)$ = atomic channel state transitions ; ternary channel.

For any optimal dissemination algorithm and for any given κ, this κ is reachable at a cost that matches some lower bound, denoted $C^*(\kappa)$ Depending on the types of problems considered, it may or may not be the case that $[oc]$ is met with any algorithm A that needs $K(\lambda)$, obtained at cost $C^*(K(\lambda))$ only. This is particularly true with problems in class $\Re$. More generally, this is true with any problem where A embeds some "performance" objectives. On-line job assignment, with minimization of makespan, would be an example ([B et al. 92], [DP92]). Optimality is further discussed in section 5.

4.2 The Hard Real-Time Distributed Multiaccess channel problem

This problem has been selected for the purpose of demonstrating that distributed real-time computing problems cannot be solved with solutions based on off-line computations, such as precomputed schedules or scheduling algorithms based on fixed priorities. More to the point, tc standing for the timeliness constraints specified by $\Lambda_{\Re}$, distributed real-time computing problems can only be solved with on-line tc-driven scheduling algorithms [18].

The problem The Hard Real-Time Distributed Multiaccess channel problem, denoted [HRTDM], arises when considering a broadcast communication channel that is shared by stations for transmitting messages. A station comprises a source and a sender. Messages released by a source are queued up for transmission by the local sender. Channels considered are equivalent to multi-writer/multi-reader concurrent atomic ternary objects. More precisely, [HRTDM] is as follows :

Set $\lambda - \lambda_{\Re} \equiv$ Computational model :
Synchronous. In the absence of access control, a channel can enter three states, namely "idle", "busy", "jammed". A channel is "idle" when no message transmission is attempted or under way. State "busy" corresponds to exactly one

message being transmitted. State "jammed" is an undesired state, which corresponds to many messages being transmitted concurrently (garbled transmission). Over such ternary channels, stations can only observe global channel states and global channel state transitions that result from their collective behavior. The number of stations that are active at any time is unknown. The number of stations has a finite upper bound, denoted n. The channel end-to-end propagation delay is small compared to message durations.

Failure models : Crash failures and send-omission failures allowed for stations. (Receive-omission failures can be considered, although this is not done here, for the sake of conciseness). Channel state transitions are assumed to be reliably propagated along the channel. We do not require that a message being transmitted by a correct sender (channel state = "busy") be correctly delivered to every station. However, we require that an erroneous message reception be distinguished from channel state "jammed".

Set $\lambda\Re$

Message releases model : Messages released by source i, $i \in [1, n]$, belong to a set denoted $M_i = m_{i,1}, m_{i,2}, ..., m_{i,p(i)}$. Every message has a finite transmission duration. Release times of messages follow arbitrary laws. Arbitrary laws are characterized via a sliding window model, as follows. W being the size of the sliding window considered, for every i, $p(i)$ integers $x_{i,k}$ are specified, $k \in [1, p(i)]$. Integer $x_{i,k}$ is the highest number of releases of message $m_{i,k}$ that can be found in any window of size W. Indirectly, this defines, $\varphi_i = X_i/W$, the upper bound on the density of message releases from source i, with $X_i = \sum_{k=1}^{p}(i)x_{i,k}$.

Set $\Lambda - \Lambda_{\Re}$

Message transmissions must be mutually exclusive (a safety property). Every message must be transmitted in bounded time (a liveness property).

Set $\Lambda_{\Re}$

Timeliness constraints : A relative latest deadline is assigned to each message. Relative deadline of message $m_{i,k}$ is denoted $d(m_{i,k})$. Deadline values are arbitrary. Let W be $maxd(m_{i,k})$, $k \in [1, p(i)]$, over all sets M_i.

- (T) every message released by a source must be transmitted before its deadline (a timeliness property)
- (D) the distributed algorithm selected belongs to a class that dominates every other class (a dominance property).

Note that (D) is equivalent to requiring that the feasibility conditions are "close enough" to NS[fc]s.

Preliminaries First, consider sets M_i' that are derived from sets M_i, by creating $x_{i,k}$ releases of every message $m_{i,k}$, $k \in [1, p(i)]$. A scenario is any collection comprising the n sets M_i', the set of $p(i)$ relative deadlines assigned to the X_i messages in every set M_i', as well as every possible pattern of release times that satisfies bounds φ_i. Feasibility conditions can be viewed as an (algorithm dependent) oracle that answers "yes" or "no" to any such question as "is (T) satisfied with this scenario ?".

In order to prove that [HRTDM] is solved, one must prove first that (T) holds whenever the set of senders considered is presented feasible scenarios. A feasible scenario is a scenario such that there exists a schedule that achieves property (T). At this point, it is useful to consider that the feasible scenarios embodied within the exposition of [HRTDM] are those that can be generated by an all-knowing adversary – referred to as Z in the sequel – which is free to decide on when sources will release messages over the set of stations, provided that the definition of sets M_i^t and the φ_i boundaries are not violated. Hence, we have no other choice left than to consider releases referred to as non-concrete releases in the real-time scheduling algorithms community. In particular, note that the sliding window-based arbitrary releases model is more general than the sporadic releases model, in that multiple messages may be released simultaneously. With this type of model, the distinction between time-triggered versus event-triggered computations [KV93] is meaningless.

Proving that (T) holds consists in devising some distributed multiaccess algorithm A which, when being used to play against Z, guarantees that every deadline is satisfied. Hence, $[cc_1]$ translates into having to meet the following double requirement, given A, in the presence of Z, $\forall i \in [1,n]$, $\forall j \in [1,X_i]$:

- (R_1) establish the expression of a function $B(m_{i,j})$ that gives a guaranteed upper bound on response times for message $m_{i,j}$
- (R_2) verify that $B(m_{i,j}) \preceq d(m_{i,j})$ holds true.

Proving that (D) holds consists in proving that A belongs to a class of algorithms that yield bounds $B(m_{i,j})$ that are always smaller than those obtained when considering other classes.

Adversary arguments can be developed considering Z as a global adversary or considering that Z is the union of several distinct adversaries. The former approach has been used in [HLLR95] to demonstrate that [HRTDM] cannot be solved with algorithms based on decisions made (fully, partially) off-line. The latter approach consists in proceeding as follows :

- pick up a distributed algorithm A, that defines the rule of the game played (i.e. imposed to the adversary/adversaries)
- extract one station, say i, from the set of stations considered
- consider that all other stations coalesce to "defeat" i, i.e. they are an adversary Z_i against which i is playing
- $[cc_2]$: prove that Z_i is not "weaker" than (i.e. as unrestricted as) Z
- $[cc_1]$: establish $B_A(m_{i,j}, Z_i)$ and prove that the double requirement $\{R_1, R_2\}$ is met for i (in the presence of Z_i, given A).

Doing the above for every possible value of i results into establishing conditions under which (T) holds true with A in the presence of Z (which is at most as strong as $\cup_i Z_i$).

Given the communication channels considered, a very basic issue that need be solved is how to enforce mutual exclusion among senders, i.e. how to handle contention.

How to handle contention Many existing network standards or off-the-shelf products or proposals from the research community are based on contention avoidance or on contention detection-and-resolution. Representatives of the contention avoidance category are decentralized polling/round-robin or token-passing algorithms. Representatives of the contention detection-and-resolution category are carrier-sense algorithms. It is reasonably obvious that probabilistic algorithms, such as that used in Ethernet, cannot solve [HRTDM].

Other off-the-shelf products or proposals from the research community are based on Synchronous Time Division (STDMA). It is also reasonably obvious that STDMA algorithms cannot solve [HRTDM]. Such algorithms can only work with a time slotted channel. How such slots can be instantiated is a crucial issue. If a unique (central) clock is used, then the solution is not distributed, hence it is unacceptable. If multiple clocks are used (e.g., one per station), then the question arises as how do they exchange messages so as to reach and maintain mutual synchrony. Inevitably, such message exchanges are conducted via either a contention avoidance or a contention detection-and-resolution algorithm. Hence, distributed STDMA does not solve [HRTDM]. Distributed STDMA can only be a "synchronous" extension of some underlying algorithm that solves [HRTDM].

It looks like the only correct solutions belong to the class of contention avoidance algorithms. This is a well accepted view, as demonstrated by recent survey publications such as [MZ95]. It is easy to demonstrate that such a view is mistaken. Note also that ternary channels are needed for a correct functioning of contention avoidance algorithms. Whenever structural changes occur (voluntarily or because of station failures), contention is unavoidable. State "jammed" is needed.

How not to solve [HRTDM] Deterministic contention avoidance algorithms cannot solve [HRTDM]. It is easy to show that Z can defeat any algorithm belonging to this class. This is essentially due to the fact that every such algorithm is based on static decisions, i.e. scheduling decisions made off-line. Such static decisions being known to Z, Z is able to generate feasible scenarios that will never match those assumptions made for the sake of computing scheduling decisions off-line. Let us briefly review three well known examples of contention avoidance algorithms. A detailed examination of this issue can be found in [HLLR95].

(i) Decentralized polling/round-robin

Senders are served in some predetermined order, which reflects the polling sequence. Consider that Z is given n messages exactly, 1 message per station, and assume that the corresponding scenario is feasible. For example, a valid schedule would be any schedule whereby $m_{i,1}$ and $m_{r,1}$ are transmitted first (in any order). Knowing the fixed polling sequence, Z can pick up release times that will lead senders to schedule some message(s) belonging to some station(s) $k(k_i, k_r)$ between $m_{i,1}$ and $m_{r,1}$ or ahead of $m_{i,1}$ and $m_{r,1}$. Either $d(m_{i,1})$ or $d(m_{r,1})$ or both deadlines are missed.

(ii) Token-Passing with timers

The addition of individual timers to polling algorithms does not help either to

solve [HRTDM]. A timer (THT's with the Token Bus or FDDI) is an upper bound θ on the service time granted to a sender. Let be θ_q the value of the timer associated with sender h, $h \in [1, n]$, by the virtue of some off-line computation. The same adversary argument used with polling can be invoked. In fact, decentralized polling defines individual timers implicitly. Considering the same feasible scenario as above, the only valid off-line computation of timer values should be $\theta_q = 0$, $k \neq i$, $k \neq r$. This would be, of course, a ridiculous decision. More generally, being aware of the fixed qk's, Z can easily pick up release times such that deadlines are missed with feasible scenarios.

(iii) Token-Passing with fixed priorities

With this type of algorithms, some method must be applied off-line to transform deadlines into fixed priorities. Given that we must solve [HRTDM], a "good" method would consist in defining a mapping function such that "short deadlines" are translated into "high priorities" (e.g., ranging between 0 and 7, 7 the highest, in the case of ISO-OSI 8802/5). The problem is, whatever the method, unbounded starvation can be experienced by any message assigned a fixed priority that is not the highest one. For such messages, deadlines are inevitably missed. If we now concentrate on the set of messages that, at any given time, are assigned the highest priority, it is obvious that such algorithms (e.g., the Token Ring protocol) boil down to (decentralized) polling. Conclusions drawn above fully apply.

In the real-time algorithms community, significant effort has been devoted to identifying good methods for transforming deadlines into fixed priorities off-line. It might be worth mentioning that Rate-Monotonic (Deadline Monotonic as well), which is a well publicized method in certain circles, does not help either in solving [HRTDM]. The correctness and the optimality of Rate-Monotonic have been established for a preemptable processor, considering a periodic releases model and assuming that a known relation holds between message periods and message deadlines [17]. The corresponding type of adversary is much weaker than the one embodied within [HRTDM], where releases and deadlines are arbitrary. Furthermore, a channel is a distributed resource (unlike a processor) that is not preemptable (unlike the assumptions that underly the optimality of the Monotonic methods). So called Generalized Rate Monotonic (GRM) is a method that is claimed to overcome these limitations. In particular, GRM is claimed to be applicable to distributed systems. As (unvoluntarily) demonstrated in [22], such claims are unfounded. The unsolvable problem faced with GRM is that there cannot exist a method that could be used off-line to transform the deadlines into fixed priorities, while demonstrating that the transformed problem is equivalent to [HRTDM] or without violating [cc2]. It is in fact easy to demonstrate that GRM cannot solve general distributed scheduling problems, contrary to the claims made in [22]. GRM is a typical example of an approach based on an artificially restrictive view of reality (see section 4.1).

(iv) Conclusions

Simple adversary arguments have been used to demonstrate that [HRTDM] cannot be solved with contention avoidance algorithms. Such arguments help

in avoiding the conventional byzantine debates on the hypothetized "real-time properties" of polling/round-robin or token-passing algorithms. A much often used argument developed in favor of such algorithms is the following one : ¡upper bounds $B(m_{i,j})$ – see section 4.2.2 – can be computed with such deterministic algorithms ; the double requirement $\{R_1, R_2\}$, i.e. $[cc_1]$, can be met ; hence, these algorithms are "good" for solving "real-time" computing problems¿.

The fundamental flaw of this argument is that such bounds are of no value, for the reason that they may never hold true (property (T) is not enforced) or they hold true under assumptions that violate $[cc_2]$ or they are too pessimistic (property (D) is not achieved). Distributed polling/round-robin or token-passing algorithms inevitably incorporate some off-line decisions. Such decisions match only a subset of the possible scenarios. Hence, the problem is not that the double requirement $\{R_1, R_2\}$ cannot be expressed. The problem is that the corresponding oracle may respond "yes" when it should respond "no" or it will respond "no" arbitrarily often when it should respond "yes". Adversary Z as embodied within [HRTDM] is too powerful to be mastered by a contention avoidance algorithm.

Another way of explaining why contention avoidance/off-line decision algorithms cannot solve [HRTDM] is as follows : such algorithms enforce sequential scheduling decisions, each spanning a set of multiple messages pending for transmission. Once such a decision is made, it cannot be altered. It is then easy for Z to defeat such decisions. The larger the set, the easier for Z. It then becomes obvious that the ideal way of playing (and winning) against Z is by providing oneself with the possibility of making or changing scheduling decisions on a per transmitted message basis (leaving aside the question of how fast such decisions can be made).

The fact that such decisions should be deadline-driven should not come as a surprise to readers familiar with the demonstrated optimality properties of the Earliest-Deadline-First algorithm. It should then be obvious that the only algorithms that make sense are those that make scheduling decisions based on the deadline data provided on-line.

How to solve [HRTDM] The solution builds upon the demonstrated optimality of centralized non-preemptive earliest-deadline-first (NP-EDF), in the class of non-idling algorithms, in the absence of overload, for the following models :

- non-concrete periodic and sporadic message releases, relative deadlines being equal to periods [15],
- aperiodic message releases, arbitrary deadlines [12].

The detailed solution can be found in [14]. A summary is provided below.

a) Timeliness property (T) Let us first have sets M_i' sorted by increasing relative deadlines. Let $m_{i,j}$ refer to the message ranked jth in set M_i', $j \in [1, X_i]$. Let us write $X = \sum_{x=1}^{n} X_i$. Consider set $M*$, the ordered union of sets M_i'. Any message ranked g in $M*$ is some unique message ranked jth in some specific set

$M_i^!$. Let Ψ be the bijection $g \longleftrightarrow i,j$ and let message durations be denoted by e.

A NS[fc] for centralized NP-EDF is as follows :
(G) $\forall g, 1 \preceq g \prec X : d(m_g) \succeq B(m_g)$,
with $B(m_g = \max_{h\in[g+1,X]}\{e(m_h)\} + \sum_{u=1}^{g} e(m_u)$.

Hence, conditions [cc_1], [cc_2] and [oc] are met in the case of an ideal distributed NP-EDF scheduling algorithm, denoted I, which would always be provided with instantaneous perfect global knowledge (of the senders waiting queues) at zero cost. Trivially, we have $B_I(m_{i,j}) = B_I(m_g)$, with $i,j = \Psi(g)$. Centralized NP-EDF or I being optimal, any valid schedule that would not be EDF-ordered can be transformed into an EDF-ordered (valid) schedule [12]. Hence, for any given feasible scenario, the lower bound of the rank for any message is obtained with I. Therefore, bounds BI are the lower bounds of any real bounds that can be enforced by any real distributed scheduling algorithm. For any such algorithm, denoted A, let us write $B_A(m_{i,j}) = B_I(m_{i,j}) + b_A(m_{i,j})$, where $b_A(m_{i,j})$ is an upper bound on the additional latency due to lack of perfect global knowledge.

Hence, for any algorithm A in class $D-NP-EDF$, (G) yields the following sufficient condition under which property (T) holds : $\forall i \in [1,n]$, $\forall j \in [1,X_i]$: $B_{D-NP-EDF}(m_{i,j}) \leq d(m_{i,j})$, with

$$B_{D-NP-EDF}(m_{i,j}) = \max_{h\in[j+1,X_i]}\{e(m_h)\} + \sum_{u=1}^{j} e_u + b_{D-NP-EDF}(m_{i,j})$$

b) Dominance property (D) What follows is a sketch of the proof. Consider the class of $D-NP-EDF$ algorithms based on contention detection-and-resolution and the class of contention avoidance algorithms, denoted CA. Pick up an algorithm in each class and consider a feasible scenario for which both algorithms generate a valid schedule (property (T) holds). Let us demonstrate that, for any message m in this scenario, we have :

$$b_{D-NP-EDF}(m) \prec b_{CA}(m)$$

Any distributed algorithm is bound to create deadline inversions (which occur also in our case because a channel is a non-preemptable resource). EDF ordering being optimal, it follows that $b(m)$ is an increasing function of the number of deadline inversions. Recall that $b(m)$ is a worst-case bound and that we are assuming that (T) holds. What is the magnitude of the number of deadline inversions ? Let us introduce the notion of a deadline equivalence class, which is a time window of some duration denoted v. Any two messages whose absolute deadlines differ at most by v belong either to the same equivalence class or to two time adjacent equivalence classes. Therefore, deadline inversions can only occur among messages that have absolute deadlines within v of each other.

With $D-NP-EDF$ algorithms, which are deadline-driven, parameter v is tunable. In particular, v does not depend on n, the highest number of

senders. Conversely, with CA algorithms, v is not freely tunable. The token rotation time or the polling sequence/round-robin latency, which depend on n, are lower bounds of v. Hence, except maybe for ridiculously small values of n, $v(D-NP-EDF)$ can always be chosen to be smaller than $v(CA)$. Therefore, the number of deadline inversions being smaller with D-NP-EDF algorithms, it follows that corresponding schedules are closer to ideal EDF-ordered schedules. Consequently, message ranks are closer to lower bounds (than ranks obtained with CA algorithms). This completes the demonstration.

Note that property (D) has been established without making any assumption w.r.t. the algorithm used to schedule messages in senders waiting queues when class CA is considered. This establishes that $D-NP-EDF$ algorithms always outperform CA algorithms, even if individual senders waiting queues are scheduled according to EDF.

Having demonstrated that class $D-NP-EDF$ dominates class CA when considering [HRTDM], we have demonstrated that optimal solutions cannot belong to class CA. Given that possible solutions to the (basic) contention problem belong either to the contention avoidance class or to the contention detection-and-resolution class (that of $D-NP-EDF$), we have therefore demonstrated that optimal solutions to [HRTDM] can only belong to class $D-NP-EDF$, when considering non-idling algorithms.

Ideally, beyond proving (D), condition [oc] should be proved to hold. However, proving that a distributed on-line scheduling algorithm is optimal still raises a few fundamental issues (see section 5).

c) An example DOD/CSMA-CD (Deadline Oriented Deterministic/Carrier Sense Multiple Access-Collision Detection) is an algorithm that belongs to class $D-NP-EDF$. It is a deterministic deadline driven variation of the ISO/OSI 8802/3-Ethernet standard (see [LLR93] for a more complete presentation). Deterministic deadline-driven binary tree search (called time trees) is used by DOD/CSMA-CD to implement $D-NP-EDF$. We have considered an arbitrarily devilish global adversary – referred to as Z_0 – which is allowed to release messages in a fully unrestricted manner. In other words, with Z_0, we have considered n adversaries , every such adversary being characterized as follows : $\forall i \in [1,n]$, $\forall k \neq i$, $\varphi_k = \infty$. Obviously, Z_0 dominates every possible adversary defined as per [HRTDM]. Using adversary techniques, we have established the expression of the BDOD function and given the [fc] under which (T) holds.

/subsubsectionWhere are the probabilities ? Let Z be the adversary embodied in some λ_{Re}. There are obvious differences between the following three approaches :

a) Probabilistic or randomized algorithms Worst-case behavior of adversary Z is non-deterministic. Timeliness properties are established via the expression of bounds $B_A(Z)$ that hold true with some computable probability. This probability is a function of the probability that the real future adversary matches postulated Z – the assumption coverage – as well as of the accuracy of the modelling of algorithm A. This probability depends on λ_{Re} and on the proof used to demonstrate that $[cc_1]$ holds.

b) Deterministic algorithms Worst-case behavior of (non-deterministic) adversary Z is deterministic. As shown with [HRTDM], timeliness properties are established via the expression of bounds $B_A(Z)$ that always hold true in the presence of Z. Hence, the computable probability that such bounds hold true in the future is the assumption coverage of postulated Z. Probabilities are not involved in the modelling of A or in the proof that $[cc_1]$ holds.

c) Approximately correct algorithms With such algorithms, most often, adversary Z is not defined. Furthermore, such bounds as $B(Z)$ are not given. Hence, probabilities that $[cc_1]$ holds cannot be computed, as there is no attempt made at establishing $[cc_1]$.

5 A UNIFIED ALGORITHMIC VIEW OF BOTH CLASSES

Recall that tc stands for the timeliness constraints that appear in Λ_{Re}. It should be clear by now that only those (distributed) on-line scheduling algorithms that are tc-driven can be contemplated for solving distributed real-time computing problems. Non real-time concurrency also implies that some decision algorithm is used to break ties in the case of actual simultaneity. Therefore, such algorithms enforce particular schedules whenever necessary. However, such algorithms are not tc-driven. Nevertheless, there is no reason why one could not take a problem in class $\aleph\Re$ and augment it with specification sets Λ_{Re} and λ_{Re}, so as to transform it into a problem in class $\Re$. In fact, recent work suggests that convergence of both classes is feasible. Let us illustrate this observation with a comparison of wait-freedom and timeliness.

5.1 Wait-freedom and timeliness

In asynchronous shared-memory computational models, unbounded wait-freedom is a liveness property. Bounded wait-freedom implies that there is an upper bound $U_i(op)$ on the number of (its own) steps that some process i takes in order to complete the execution of a given operation (op). Prima faciae, the fact that $U_i(op)$ holds regardless of the behavior of other processes is disturbing in light of elementary results in queueing theory, where it is held that $U_i(op)$ depends on the amount of service that process i receives, this amount being "what is left" by the other processes. (For example, other processes being released infinitely often, $U_i(op)$ could be infinite in the absence of a fair scheduling policy).

A first observation is that waiting queues are not part of the models considered when addressing problems of wait-free linearilizability. Nevertheless, the fact that processes can release one operation at a time only is a constraint on their behavior. Futhermore, even under this constraint, there must be a rule that

serves to break ties whenever real concurrency occurs on an elementary object. Such a rule is a scheduling policy, which explains why such bounds as $U_i(op)$ hold. Another explanation derives from the algorithms used to enforce wait-freedom. For example, in [H88], a general construction algorithm is described whereby a general wait-free concurrent object can be built out of multiple elementary wait-free objects. One feature of the algorithm could be characterized as "limited altruism", for the reason that "fast" processes help "slow" processes to proceed, to a certain extent. For example, "fast" process i performs the operation that "slow" process j intends to perform, before process i proceeds with its own operation. This type of rule clearly is a scheduling policy. Bounds $U_i(op)$ depend on the scheduling policy considered. Therefore, scheduling policies or algorithms being implicitly or explicitly considered, the apparent contradiction vanishes.

Furthermore, this opens the way to the concept of real-time wait-free concurrent objects. The specification of such an object is the specification of a concurrent object augmented with :

- Λ_{Re}, the specification of the (arbitrary) timeliness constraints to be met by every operation (denoted d)
- λ_{Re}, the specification of the event releases model.

As with every problem in class $\Re$, one has to find a scheduling algorithm such that, under λ_{Re}, for some [fc] to be established, the following timeliness property holds :

for every process i, for every operation $(op_{i,.})$, $\exists U_i(op_{i,.}) : U_i(op_{i,.}) \preceq d_i(opi,.)$.

The "plugging" of Λ_{Re} and λ_{Re} yields the "unplugging" of the implicit/explicit (tc-independent) scheduling algorithm yielding bounded wait-freedom, to be superseded by a tc-driven on-line scheduling algorithm.

5.2 Which computational models for class $\Re$

At first sight, asynchronous computational models do not make sense when considering a problem in class R. Unbounded delays for completing elementary operations seem to be antagonistic with the goal of enforcing timeliness properties for global operations. However, if we clearly distinguish the design phase of an algorithm from its implementation phase, there is no reason to reject asynchronous models. Consider for example the (deterministic) algorithms and constructs used to build wait-free concurrent objects or to solve [AC]. Imagine that such algorithms or constructs are "immersed" in a real system that is endowed with timeliness properties, via some other algorithm(s). For example, upper bounds are proved to hold for elementary computation/communication steps. It is then possible to establish which are the timeliness properties achieved by those algorithms or constructs that were proved correct in some asynchronous model. For example, the "immersion" of $\diamond W$ [6] in a communication system that solves [HRTDM] yields a perfect failure detector, which can be used to solve any real-time extension of problem [AC].

Of course, a similar observation applies a fortiori in the case of solutions developed for partially synchronous models. Quite systematically until now, the real-time algorithms community has considered synchronous computational models for the design phase. This a sound approach whenever assumption set λ and related [fc] cannot be violated (or whenever such violations can be ignored). However, the danger with such models is that they lead to algorithms that cannot provably keep enforcing some minimal property whenever the postulated computation/communication bounds are violated, in contrast with algorithms designed for asynchronous or partially synchronous models which, in many cases, maintain some safety property (e.g. silence rather than disagreement in the case of [AC]), would the assumption set λ be invalidated at run-time. Such concerns typically arise with critical systems.

5.3 Optimality

Not much is known yet about optimal distributed real-time scheduling. It is not even clear that we have a satisfactory definition at hand (see further). It may then be more appropriate to begin with the identification of which is the class of algorithms that necessarily contains the optimal one(s), for any given problem. This is precisely what we have accomplished with [HRTDM], in establishing the dominance property of class $D-NP-EDF$ over any other known class. Optimality within class $D-NP-EDF$ is an open issue. Idling algorithms may dominate non-idling ones. Dissemination may yield feasibility conditions closer to NS[fc]s. For example, every message transmitted (i.e. ranked first in its waiting queue) could carry the deadlines of some of the pending messages. Such a dissemination scheme may greatly reduce the likelihood of collision occurrence, which would bring bounds B closer to optimal bounds. More generally, using the notations introduced in section 4.1, one could be tempted to equate optimality with a necessary and sufficient condition for κ, denoted $\kappa*$. Any algorithm that needs $\kappa*$ only, obtained at cost $C*(\kappa*)$ would be optimal. However, the decision algorithm embedded within A must also be taken into account. The "quality" of the decisions may improve significantly with "small" increments of common knowledge (in addition to $\kappa*$).

The general issue of optimality of distributed algorithms is being addressed through various concepts and definitions that have emerged recently. Many of them resemble concepts and definitions explored in game theory. Examples are competitive analysis of on-line algorithms s explored in game theory. Examples are competitive analysis of on-line algorithms [23], competitive analysis of distributed (on-line) algorithms (e.g., [10], [1]). An early application of competitive analysis to demonstrating optimality in the case of centralized preemptive on-line scheduling, in the presence of overload, can be found in [3].

Competitive ratios are a measure of "how well" an on-line player can perform against some adversary, in worst-case conditions. Competitive ratios depend on the ratio of advance knowledge given to the player, who selects the on-line algorithm, over the knowledge given to the adversary who, knowing the algorithm selected, is able to generate those scenarios that maximize some regret function.

In the context of [HRTDM], s being the scenarios that can be generated by Z, the competitive ratio of any algorithm A with respect to timeliness is defined to be $sup_{(}\alpha$ bound $B(A,\sigma)$/inf bound $B(Z,\sigma)$.

Of course, Z also is a distributed player. In departure from the original definitions, we consider that Z also incurs some cost due to distribution. Z wins over A because Z can generate schedules that minimize its deadline inversions while maximizing deadline inversions experienced by A.

Another departure from the original definitions is explored in [1], where any algorithm A is evaluated against other algorithms called champions (denoted H), that are optimal for specific schedules in s and correct for every possible schedule in s. The competitive ratio of an algorithm A with respect to latency, using the notations proper to this paper, is defined to be sup_σ bound $B(A,\sigma)/inf_H$ bound $B(H,\sigma)$. Competitive latency is examined only for schedules and algorithms that are said compatible. This restriction is equivalent to considering event releases models such that no queueing phenomenon ever develops, or, stated differently, to ignoring sojourn times in waiting queues. This is reminiscent of the observation made relative to wait-free concurrent objects.

Equating optimality with best achievable competitive ratios is not entirely satisfactory, for the reason that competitive ratios are not an homogeneous measure. By this, we mean that an algorithm whose competitive ratio would match the optimal ratio could still be dominated by some other algorithm, when being presented scenarios other than worst-case.

Nevertheless, competitive analysis of distributed algorithms seems to be a promising analytical vehicle, powerful enough to explore problems in both classes $\aleph\Re$ and $\Re$ homogeneously. This view is backed by the recent explosion of papers in this area.

Acknowledgments

I would like to thank Vassos Hadzilacos and Sam Toueg for fruitful discussions on Asynchronous Consensus and on bounded wait-freedom.

References

1. M. Ajtai, J. Aspnes, C. Dwork, O. Waarts, "A theory of competitive analysis for distributed algorithms", 35th Symposium on Foundations of Computer Science, Nov. 1994, 401-411.
2. M. Abadi, L. Lamport, "An old-fashioned recipe for real-time", ACM Trans. on Programming Languages and Systems, vol. 16, 5, Sept. 1994, 1543-1571.
3. S. Baruah et al., "On the competitiveness of on-line real-time task scheduling", IEEE Real-Time Systems Symposium, Dec. 1991, 106-115.
4. Y. Bartal et al., "New algorithms for an ancient scheduling problem", 24th ACM STOC, May 1992, 51-58.
5. H. Brit, S. Moran, "Wait-freedom vs. bounded wait-freedom in public data structures", Proc. of the 13th ACM Symp. on Principles of Distributed Computing, Aug. 1994, 52-60.

6. T. Chandra, V. Hadzilacos, S. Toueg, "The weakest failure detector for solving consensus", Proc. of the 11th ACM Symp. on Principles of Distributed Computing, Aug. 1992, 147-158.
7. T. Chandra, V. Hadzilacos, S. Toueg, "Impossibility of group membership in asynchronous systems", in preparation.
8. T. Chandra, S. Toueg, "Unreliable failure detectors for asynchronous systems", Proc. of the 10th ACM Symp. on Principles of Distributed Computing, Aug. 1991, 325-340.
9. D. Dolev, C. Dwork, L. Stockmeyer, "On the minimal synchronism needed for distributed consensus", Journal of the ACM, vol. 34, 1, Jan. 1987, 77-97.
10. X. Deng, C.H. Papadimitriou, "Competitive distributed decision-making", 12th IFIP Congress (Elsevier North-Holland Pub.), vol. 1, 1992, 350-355.
11. M. Fischer, N. Lynch, M. Paterson, "Impossibility of distributed consensus with one faulty process", Journal of the ACM, vol. 32, 2, April 1985, 374-382.
12. L. George, P. Mühlethaler, N. Rivierre, "Optimality and non-preemptive real-time scheduling revisited", INRIA Research Rep. n 2516, March 1995.
13. M.P. Herlihy, "Impossibility and universality results for wait-free synchronization", Proc. of the 7th ACM Symp. on Principles of Distributed Computing, Aug. 1988, 276-290.
14. J.F. Hermant, G. Le Lann, N. Rivierre, "A general approach to real-time message scheduling over distributed broadcast channels, to appear in Proc. of the IEEE/INRIA Conf. on Emerging Technologies and Factory Automation, Oct. 1995.
15. K. Jeffay, D.F. Stanat, C.U., Martel, "On non-preemptive scheduling of periodic and sporadic tasks", IEEE Real-Time Systems Symposium, San-Antonio, Dec. 1991, 129-139.
16. H. Kopetz, P. Verissimo, "Real-time and dependability concepts", in Distributed Systems, chapter 16, S.J. Mullender Ed. (Addison-Wesley Pub.), 1993.
17. C.L. Liu, J.W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment", Journal of the ACM, vol. 20, 1, Jan. 1973, 46-61.
18. G. Le Lann, "Scheduling in critical real-time systems : a manifesto", Third Intl. Symp. on Formal Techniques in Real-Time and Fault-Tolerant systems, Lübeck (D), Sept. 1994, Lecture Notes in Computer Science n 863 (Springer-Verlag pub.), 511-528.
19. G. Le Lann, N. Rivierre, "Real-time communication over broadcast networks : the CSMA-DCR and the DOD/CSMA-CD protocols", INRIA Research Rep. n 1863, March 1993, 35 p.
20. L. Lamport, R. Shostak, M. Pease, "The Byzantine generals problem", ACM Trans. on Programming Lang. and Syst., vol. 4, 3, July 1982, 382-401.
21. N. Malcolm, W. Zhao, "Hard real-time communication in multiple-access networks", Journal of Real-Time Systems (Klüwer Academic Pub.), vol. 8, 1, Jan. 1995, 35-77.
22. L. Sha, S.S. Sathaye, "A systematic approach to designing distributed real-time systems", IEEE Computer, Sept. 1993, 68-78.
23. D.D. Sleator, R.E. Tarjan, "Amortized efficiency of list update and paging rules", Com. of the ACM, vol. 28, 2, Feb. 1985, 202-208.