A uniform reliable multicast protocol with guaranteed response times *

Laurent George[1] and Pascale Minet[2]

[1] ESIGETEL, Département Télécom, 1 rue du port Valvins, 77210 Avon, France
george@esigetel.fr
[2] INRIA Rocquencourt, Projet Reflecs, 78153 Le Chesnay Cedex, France
pascale.minet@inria.fr

Abstract. Uniform reliable multicast protocols with a guaranteed response time are of concern in numerous distributed real-time systems (e.g. distributed transactional systems, high available systems). We focus on uniform reliable multicast protocols in processors groups. The source of a message can belong or not to the destination group. A bounded number of processors crashes and network omissions is tolerated. The uniform reliable multicast protocol, designed and proved in this paper, works with Earliest Deadline First (EDF) scheduling, proved optimal in the uniprocessor context. A worst-case response time analysis for sporadic tasks subject to release jitter is given for non-preemptive EDF on a uniprocessor and extended to the distributed case using the holistic scheduling approach.

1 Introduction

Contrary to a broadcast that is targeted to the set of all the processors in the system, a multicast is targeted to a subset of processors. In the following such a subset is called a group. Informally, a reliable multicast service ensures that for any message m_i, all the correct processors (i.e. processors that do not crash) in the destination group agree either to deliver m_i or not to deliver m_i. Notice the distinction between *deliver* and *receive*: a message is received from the network (lower interface of the reliable multicast module). A message is delivered in the user queue (upper interface of the reliable multicast module).
We can notice that only the behavior of correct processors in the destination group is specified. Just before crashing, a processor in the destination group can deliver messages which are not delivered by correct processors in that group. Such a behavior is prevented by the following requirement: all messages delivered by a processor, whether correct or not, in the destination group, are delivered by all the correct processors in that group. That requirement is captured by the uniform reliable multicast.
The paper is organized as follows: in section 2 the problem to solve is specified: the assumptions made and the properties to be achieved are described. Section 3

* Published in the Proceedings of LCTES'98, June 1998, Montreal, Canada

contrasts our approach with related work. In section 4, the reliable multicast algorithm is given. Section 5 is concerned with the real-time dimension. A worst case analysis based on EDF scheduling [11], is given that enables to establish the worst case end-to-end response time for the reliable multicast of a message.

2 The problem: assumptions and required properties

According to the TRDF (Real-Time, Distribution and Fault-Tolerance) method, a method for designing and dimensioning critical complex computing systems [12], a problem is specified by a pair $< assumptions, properties >$. The assumptions are given by five models defined hereafter.

2.1 Assumptions

a) The structural model
The structural model consists of K network controllers NC_1,NC_2,..,NC_K interconnected by means of a communication network. Each network controller is connected to all the others. The network controllers provide the uniform reliable multicast service. The network controllers are organized in groups, in which messages are multicast. A network controller can belong to one or several groups. For a sake of simplicity, groups are assumed to be static (i.e. no voluntary departure, no join). Each network controller has a local clock used to timestamp the messages it multicasts.

b) The computational model
The computational model defines the assumptions made concerning the transmission delays and the processing delays. In this study, the computational model taken is the synchronous one. The network transmission delay has a known upper bound *Max* and a known lower bound *Min*. The controller processing delay has a known upper bound. Moreover the controllers clocks are assumed to be ε-synchronized: the maximum difference between two clocks of non-crashed controllers is not higher than the precision ε.

c) The processing model
We now focus on the network controllers. Any network controller only knows to which groups it belongs, but it does not know the other controllers in these groups: it does not know the membership of these groups. With each message m_i is associated a relative deadline D_i: the maximum response time acceptable for the reliable multicast of message m_i in the destination group *dest(m_i)*. The timestamp of message m_i, denoted *ts(m_i)*, is the time, given by the network controller clock, when the reliable multicast of message m_i is requested by the host. Consequently, the absolute deadline of message m_i, denoted *d(m_i)*, is equal to *ts(m_i)* + D_i. At their upper interface, the network controllers offer two primitives (see figure 1):

• *R_Multicast* (m_i, *dest*(m_i), *src*(m_i), *ts*(m_i), *d*(m_i)): where the network controller *src*(m_i) performs the reliable multicast of message m_i in the network controllers group *dest*(m_i), the message m_i having *ts*(m_i) for timestamp and *d*(m_i) for absolute deadline. Notice that there is no constraint on *dest*(m_i) and *src*(m_i) : *src*(m_i) can belong to *dest*(m_i) or not.

• *R_Deliver (mi, dest(mi), src(mi), ts(mi), d(mi))*: where a network controller in *dest*(m_i) notifies its host of the reliable delivery of message m_i, the message m_i having *src*(m_i) for source, *ts*(m_i) for timestamp and *d*(m_i) for absolute deadline.

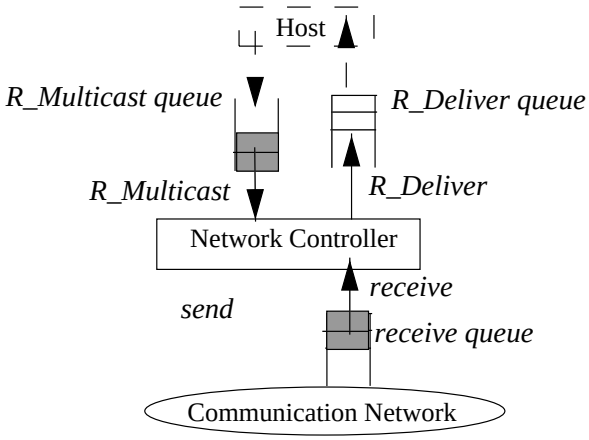

Fig. 1. The primitives provided / used by the uniform reliable multicast service.

At their lower interface (the communication network), the network controllers use two basic primitives (see figure 1):

• *send* (m_i, *dest*(m_i), *src*(m_i), *ts*(m_i), *d*(m_i)): where a network controller requests the network to send the message m_i, originated from *src*(m_i), to all network controllers in *dest*(m_i) except itself; the message m_i being timestamped with *ts*(m_i) and having *d*(m_i) for absolute deadline. Notice that a network controller does not send a message m_i to itself, even if it belongs to *dest*(m_i). The primitive *send()* returns only when the message has been sent by the network.

• *receive* (m_i, *dest*(m_i), *src*(m_i), *ts*(m_i), *d*(m_i)): where a network controller in *dest*(m_i), receives from the network a message m_i, originated from *src*(m_i), timestamped with *ts*(m_i) and having *d*(m_i) for absolute deadline.

d) The failure model

The failures model defines, for each system component, the possible failures:

- A controller can fail by crash. The execution of a reliable multicast must tolerate up to f_c controller crashs.

- The network can fail by omission. More precisely the network can omit to put a message in one receive queue. The execution of a reliable multicast must tolerate up to f_o network omissions.

- The clocks of non-crashed controllers are assumed to be monotonically increasing (see for instance [14]) and their drift rate is assumed to be bounded by a constant.

e) The events model

The events model defines the events originated from the system environment and triggering a processing by the system. In our case such an event is the insertion of the *R_Multicast* request of an occurrence of a message m_i, in the *R_Multicast* queue. In the following, "the message m_i" stands for "the occurrence of message m_i whose *R_Multicast* is requested at time *ts(m_i)*". Events are assumed to occur according to a sporadic law: T_i is the minimum interarrival time between two successive *R_Multicast* requests of message m_i. In the following, T_i is called the period.

2.2 Required properties

Before defining the properties that must be achieved by the uniform reliable multicast service, we first define the notion of correct network controller. A controller that does not crash is said *correct*, otherwise it is said faulty. A controller is said non-crashed if it has not yet crashed. The properties given here are the ones defined in [8], adapted to our context. Properties (P1) to (P3) specify a uniform reliable multicast in any destination group. Notice that according to our processing model, property (P1) does not require that the source of the message belongs to the destination group.

- **(P1) Validity**: if a correct controller R_Multicasts a message m_i, then some correct controller in *dest(m_i)* eventually R_Delivers m_i.

- **(P2) Uniform Agreement**: if a controller, whether correct or not, R_Delivers a message m_i, then all correct controllers in *dest(m_i)* eventually R_Deliver m_i.

- **(P3) Uniform Integrity**: for any message m_i, every controller NC_j, whether correct or not, R_Delivers m_i at most once and only if NC_j is in *dest(m_i)* and m_i was previously R_Multicast by the network controller *src(m_i)*.

- **(P4) Uniform Timeliness**: no message m_i, whose R_Multicast has been requested at real time t, is R_Delivered after its absolute deadline $t+D_i$.

3 Related work

Our Uniform Reliable Multicast protocol relies on the concept of relays. The relays are introduced to mask the source crash and the network omissions: the relays are in charge of repeating the messages initially sent by the source. That is the main difference with the protocol given in [8] where all controllers relay any message received for the first time. The relays limit the number of messages generated by the algorithm. Another difference is that the choice of the message to be processed is specified: the one with the earliest absolute deadline. Earliest Deadline First scheduling has been proved optimal in the uniprocessor non-preemptive context [9], when the release times of tasks are not known in advance. The worst case response time analysis of EDF is done before implementation, based on the identification of worse case scenarios.
In [3], the network omissions are masked by redundant channels. Our protocol does not need redundant channels. As defined in [6], our protocol is a genuine multicast protocol: the execution of the algorithm implementing the reliable multicast of a message m_i to a destination group *dest(m_i)* involves only the source processor and the processors in *dest(m_i)*. It does not use group membership services. However contrary to [8] and [1], our reliable multicast protocol works only in synchronous systems.
End-to-end delay guarantee has been studied in packet switching networks, where each traffic flow is allocated a reserved rate of server capacity [17], [13]. The guarantee provided by the virtual clock server [17] is conditional, it works only for on-time packets arrivals. In [10], the computation of the worst case response time of a message is based on fixed priority scheduling. Our worst case response time analysis is innovative. The analysis consists of three parts: 1) the computation of the worst case response time of a task set with a jitter for a uniprocessor non-preemptive EDF scheduling, 2) the extension of those results to the distributed case, where clocks are ε-synchronized, 3) the holistic approach [16] for the computation of the end-to-end response time. To our knowledge, the holistic approach has always been applied to a sequential processing of message [16], [15], [7]. It has never been applied to parallel processing: here the parallel processing by the relays. In [16], the scheduling is based on fixed priorities. In [15] the scheduling is preemptive EDF with a blocking factor. Here we use non-preemptive EDF.

4 The reliable multicast algorithm

In this section we design a Uniform Reliable Multicast protocol. In subsection 4.1 we first give the assumption made and an intuitive idea of the algorithm. The algorithm is given in subsection 4.2.

4.1 Assumption and principles

Informally a message m_i whose reliable multicast is requested is timestamped and put in the R_Multicast queue. The processing of that message by the Uniform Reliable Multicast module generates send and receive operations over the

network, and finally the message is R_Delivered by all non-crashed controllers in *dest(m_i)*: the message is put in the R_Deliver queue of each non-crashed controller in *dest(m_i)*. We recall that the reliable multicast must tolerate up to f_c controller crashes and f_o network omissions. More precisely, we assume:

Assumption 1: any message m_i sent by any controller NC_j to *dest(m_i)* is received by at least *card[dest(m_i)-{NC_j}]-f_o-f_c* correct controllers in *dest(m_i)* distinct from NC_j.
The reliable multicast algorithm relies on the concept of relays. They are introduced to mask the source crash and the network omissions: the relays of message m_i are in charge of repeating the message m_i initially sent by *src(m_i)*. In section 4.2 the reliable multicast algorithm is given with the number of relays per message. In the next subsection, properties *P1*, *P2* and *P3* are proved. We first give an idea of the algorithm. Informally the reliable multicast of message m_i is performed as follows:

- the source sends m_i to all controllers in *dest(m_i)* except itself;
- if the source belongs to *dest(m_i)*, it delivers m_i.
- upon first receipt of m_i from *src(m_i)*, a relay sends the received message to all the controllers in *dest(m_i)* except itself.
- upon first receipt of m_i, any controller in *dest(m_i)* delivers m_i.

4.2 The reliable multicast algorithm

We recall that NC_j receives only messages m_i such that NC_j is in *dest(m_i)*, and when NC_j sends a message it does not send it to itself. NC_j runs the following algorithm:

upon R_Multicast queue not empty/* at step 0 */
{
  extract the msg. m_i with the earliest deadline *d(m_i)* in R_Multicast queue
  $send(m_i, dest(m_i), src(m_i), ts(m_i), d(m_i))$
  if NC_j is in $dest(m_i)$ then $R\_Deliver(m_i, dest(m_i), src(m_i), ts(m_i), d(m_i))$
}
upon "receive queue not empty"
{
  extract the message m_i with the earliest deadline $d(m_i)$ in the receive queue
  if NC_j is $relay(m_i)$ and first receipt of m_i from $src(m_i)$ then
    $send(m_i, dest(m_i), src(m_i), ts(m_i), d(m_i))$
  if first receipt of the message m_i and $NC_j \neq src(m_i)$ then
    $R\_Deliver(m_i, dest(m_i), src(m_i), ts(m_i), d(m_i))$
  else discard (m_i)
}
to execute $R\_Deliver(m_i, dest(m_i), src(m_i), ts(m_i), d(m_i))$
{
  insert m_i in the R_Deliver queue
}

We now establish the minimum number of relays per source.

Lemma 1. *Each message m_i has, in dest(m_i), f_c+f_o+1 relays distinct from src(m_i). A feasibility condition is card(dest(m_i) - {src(m_i)})$\geq f_c+f_o+1$.*

Proof. Any message m_i sent by its source ($src(m_i)$) is received, according to assumption 1, by at least $card[dest(m_i)-\{src(m_i)\}]-f_o-f_c$ correct controllers distinct from the source. To ensure that there is at least one correct controller, we must have $card[dest(m_i)-\{src(m_i)\}]-f_o-f_c \geq 1$. Hence the feasibility condition is $card[dest(m_i)-\{src(m_i)\}] \geq f_o-f_c+1$. That is why we associate with each message f_c+f_o+1 relays in $dest(m_i)$ (distinct from the source). Indeed at most f_c+f_o relays can have not received the message. Relays are chosen to share the workload in the destination group $dest(m_i)$.
The maximum number of messages sent over the network, is:

- in case of a multicast network, a total of f_c+f_o+2 messages per reliable multicast sent over a multicast network.
- in case of a point-to-point network, $(f_c+f_o+2)[card(dest(m_i))-1]$ messages per reliable multicast of message m_i sent over a point-to-point network, when $src(m_i)$ belongs to $dest(m_i)$.

4.3 Proofs of properties

We now prove that properties (P1), (P2) and (P3) are met with regard to primitives R_Multicast and R_Deliver: in other words, the algorithm achieves a Uniform Reliable Multicast.

Lemma 2. *The Validity property (P1) is met.*

Proof. If a correct source R_Multicasts a message m_i, it completes the execution of the R_Multicast algorithm because by definition of correct, that source does not fail. If it belongs to $dest(m_i)$, it eventually R_Delivers m_i. If it does not belong to $dest(m_i)$, at least one correct relay in $dest(m_i)$ eventually receives m_i by assumption 1 and Lemma 1 and hence R_Delivers m_i.

Lemma 3. *The Uniform Agreement property (P2) is met.*

Proof. Let us assume that there exists a message m_i such that a controller NC_j in $dest(m_i)$, whether correct or faulty, R_Delivers m_i. According to the algorithm run by any controller, a message cannot be R_Delivered before having been sent by its source. According to assumption 1, if p omissions have occurred, $0 \geq p \geq f_0$, message m_i has been received by at least $card(dest(m_i)-\{src(m_i)\})-f_c-p$ correct controllers distinct from $src(m_i)$. Among them there are at least f_o-p+1 correct relays. According to the algorithm, each of them sends m_i to all controllers in $dest(m_i)$ except itself. As the total number of omissions cannot exceed f_o during the execution of a reliable multicast, at most f_o-p omissions can occur during these f_o-p+1 sendings, any correct controller receives m_i, and R_Delivers it.

Lemma 4. *The Uniform Integrity property (P3) is met.*
Proof. According to the network failure model (omission) and the controller failure model (crash), a controller cannot forge a message. A source, whether correct or faulty, delivers its message m_i at most once at the end of the execution of the R_Multicast and only if it belongs to *dest(m_i)*. A controller in *dest(m_i)*, whether correct or faulty, which is not the source of the message delivers that message at most once: after the first receipt.

5 The real-time dimension

In this section we are concerned with the worst case response time of the reliable multicast of a message m_i. We present in section 5.2 the holistic approach adopted to perform the computation of that end-to-end response time. Hence as depicted on figure 2, the computation of the worst case response time, on controller NC_j in *dest(m_i)*, for the reliable multicast of a message m_i originated from $src(m_i)$ depends on the computation of the worst case response time on:

- *src(m_i)*,
- the relays of m_i,
- NC_j in *dest(m_i)*,
- the communication network.

We call direct receipt, a receipt where the message is directly received from the source. An indirect receipt corresponds to a receipt from a relay.

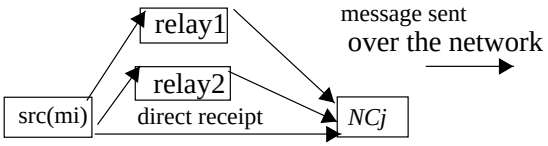

Fig. 2. Three possibilities for NC_j to receive m_i

On figure 2, there are three possibilities for message m_i to be received by NC_j in *dest(m_i)*: from *src(m_i)*, from *relay*1 or from *relay*2. In the following, all copies of m_i are considered as distinct messages, unless specified. By our assumption, the maximum transmission delay of the communication network is known and equal to *Max* and the minimum transmission delay is equal to *Min*. We could have considered a value depending on the message to transmit, but to keep the analysis simple we take the same time for all messages. We then focus on the assumptions taken for the network controllers (see section 5.1). Section 5.2 shows how the end-to-end response time between the R_Multicast request and the R_Deliver of a message m_i by a non-crashed controller NC_j can be computed using the holistic scheduling approach. Section 5.3 shows how to compute the worst case response time of a set of tasks scheduled EDF on a uniprocessor. Section 5.4 shows how the previous established results can be used to compute the end-to-end response times.

5.1 Assumptions related to network controllers

We assume that each network controller is controlled by an Application Specific Integrated Circuit (ASIC) that enables to send and receive messages in parallel (i.e receiving a message has no impact on the sending). This can be virtually seen as two processors working in parallel: one in charge of receiving messages from the network and another one in charge of sending messages in the network. The processing of a message by one of these processors is assumed to be non-preemptive. The scheduling policy of each processor of a network controller consists in serving the message with the earliest deadline in one of the two waiting queues, depicted in gray on figure 1: the R_Multicast queue and the receive queue. The processing of a message consists of basic actions. The times associated with these actions are given hereafter, the network controllers being assumed homogeneous for a sake of simplicity:

- P_i: time needed to put message m_i in the R_Deliver queue and to notify the host.
- R_i: time needed to receive message m_i from the receive queue.
- S_i: time needed to send message m_i.

5.2 The holistic approach

Tindell and Clark [16] have developed an interesting approach to establish feasibility conditions in a distributed system based on the computation of worst case end-to-end response times with fixed priority assignment. With the holistic approach, the analysis of end-to-end response times in real-time distributed systems is made easier by the concept of attribute inheritance.

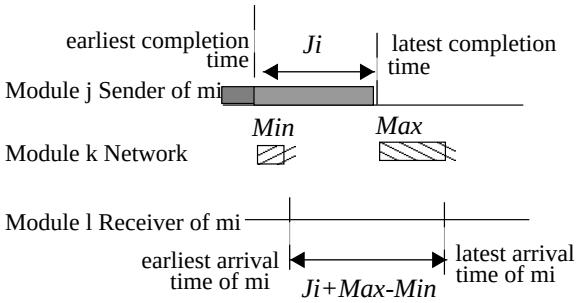

Fig. 3. Release jitter inheritance

In the following, a module is either a network controller or the communication network. Assume that a module j sends a message m_i with a period T_i to a module k. The release jitter J_i resulting from module j is equal to the maximum response time on module j minus the minimum response time. From module k point of view, the scheduling problem is to determine the worst case response

time of m_i where m_i is a message with a period T_i and a release jitter J_i. That is the principle of attribute inheritance. Figure 3 shows how a receiver, *module l*, inherits for message m_i sent by module j, a release jitter of J_i+*Max-Min*.

With attribute inheritance, it is possible to reason on any module independently. The main problem is to be able to determine the worst case response time of a message subject to a release jitter on a module, where messages are scheduled EDF. Section 5.3 addresses that problem. The holistic approach is then iterative: the response time of all messages sent or received is computed on every network controller as a function of the current release jitter of the messages; a resulting release jitter is determined which in turn leads to new worst case response times. Those steps are iterated until the release jitters of messages in every queue of the network controllers stabilize. Notice that the response time of a message is an increasing function of the jitter (see section 5.3). This guarantees that either the system converges or a deadline on the receipt of a message is missed. We now apply this result to the computation of the response time of a message m_i sent by $src(m_i)$ through a reliable multicast service. Let us focus on the messages processed by any network controller NC_j. We identify two types of messages: the sent messages and the received messages. For the sent messages, we have:

- let τ_s^j be the set of messages whose source is NC_j. We have $\forall m_i \in \tau_s^j$, the duration on NC_j of the associated sending is $C_i = S_i + P_i$ if $NC_j \in dest(m_i)$, and $C_i = S_i$ otherwise.

- let τ_{sr}^j be the set of messages relayed by NC_j. We have $\forall m_i \in \tau_{sr}^j$, the duration on NC_j of the associated sending is $C_i = S_i$.

A network controller NC_j has to send messages belonging to $\tau_s^j \cup \tau_{sr}^j$. For the received messages, we have:

- let τ_r^j be the set of messages received by NC_j. Notice that a message m_i can be received either directly from *src(m_i)* or indirectly from *relay(m_i)*. We have $\forall m_i \in \tau_r^j$, the duration on NC_j of the associated receipt is $C_i = R_i + P_i$ if $NC_j \neq src(m_i)$, and $C_i = R_i$ otherwise. Indeed we do not distinguish in the analysis the first receipt of m_i.

Section 5.3 describes how to compute the worst case response time of a set of tasks, subject to a release jitter and scheduled on a uniprocessor using EDF. This analysis will be used locally on any network controller to determine the worst case response time on the sending and on the receiving of a message and consequently the new release jitter of a message. Informally to compute the release jitter of m_i on every involved network controller NC_j, the holistic scheduling analysis proceeds by iterative steps:

- Step 0: on every network controller NC_j compute the initial jitter $J_i^{\tau_s^j \cup \tau_{sr}^j}(0)$ for every message m_i to be sent. For a message m_i whose source is NC_j, the release jitter is equal to *0*. For a message m_i such that NC_j is a relay of m_i, the release jitter is at least equal to *Max-Min*.

• Step p: compute on every network controller NC_j the jitter for every received message m_i. If NC_j is a relay of m_i then the copy of m_i which is sent by NC_j is the one received from *src(m_i)*. We can now compute on every network controller the jitter $J_i^{\tau_s^j \cup \tau_{sr}^j}(p+1)$ for every message m_i to be sent. If for any network controller NC_j and for any message m_i in $\tau_s^j \cup \tau_{sr}^j$, the jitter stabilize (i.e $J_i^{\tau_s^j \cup \tau_{sr}^j}(p+1) = J_i^{\tau_s^j \cup \tau_{sr}^j}(p)$), we have found the release jitter for any message. If there exists a network controller such that the maximum end to end response time of a received message m_i is higher than D_i, the algorithm halts: the message set is not feasible. Otherwise step *p+1* is run, and so on...

Before computing the worst case end-to-end response times (see section 5.4), we give hereafter an implementation of the holistic scheduling analysis; the function that returns the maximum response time on NC_j of a message m_i scheduled among other messages in τ^j is called $max\_rt(\tau^j, m_i)$ where τ^j is either $\tau_s^j \cup \tau_{sr}^j$ or τ_r^j. The function $max\_rt(\tau^j, m_i)$ is described in section 5.3.b. The release jitter is then equal to the maximum response time minus the minimum response time. The minimum response time for a message m_i in τ_{sr}^j is S_i and for a message m_i in τ_r^j is $R_i + P_i$ if $src(m_i)$ is not NC_j and R_i otherwise. The following procedures are a direct application of Step *0* and Step p.

• *init()* determines the initial release jitter of any message to be sent.

• *compute_sending()* determines the worst case response time of any message to be sent on every network controller. The worst case response time computation is based on the analysis proposed in section 5.3.b.

• *compute_receiving()* determines on any network controller the new maximum release jitter of any received message.

init() /* at step 0 */
{
  let $\tau_s^j = \{m_i$ such that $NC_j = src(m_i)\}$
  let $\tau_{sr}^j = \{m_i$ such that $NC_j = relay(m_i)\}$
  let $\tau_r^j = \{m_i$ such that $NC_j \in dest(m_i)\}$
  for any NC_j
  /* messages R_multicast by NC_j */
    $\forall m_i \in \tau_s^j, J_i^{\tau_s^j \cup \tau_{sr}^j}(0) := 0$
  /* relayed messages inherit an initial release jitter from the network */
    $\forall m_i \in \tau_{sr}^j, J_i^{\tau_s^j \cup \tau_{sr}^j}(0) := Max - Min$
  endfor
}
compute_sending(p) /* at step $p > 0$ */
{
  for any NC_j
/* compute the resp. time of msg. sent by NC_j with jitter $J_i^{\tau_s^j \cup \tau_{sr}^j}(p)$ */

$\forall m_i \in \tau_s^j \cup \tau_{sr}^j$, $r_i^{\tau_s^j \cup \tau_{sr}^j}(p) := max\_rt(\tau_s^j \cup \tau_{sr}^j, m_i)$

endfor

}

compute_receiving(p) /* at step $p > 0$ */

{

for any NC_j

for any $m_i \in \tau_r^j$ received from NC_k, $k \neq j$

if $NC_k \in dest(m_i)$ then

$J_i^{\tau_r^j} := r_i^{\tau_s^k \cup \tau_{sr}^k}(p) - S_i - P_i + Max - Min$

else $J_i^{\tau_r^j} := r_i^{\tau_s^k \cup \tau_{sr}^k}(p) - S_i + Max - Min$

/* response time computed with the jitter $J_i^{\tau_r^j}(p)$ */

$r_i^{\tau_r^j}(p) := max\_rt(\tau_r^j, m_i)$

if $NC_j \in relay(m_i)$ and $NC_k = src(m_i)$ then

$J_i^{\tau_s^j \cup \tau_{sr}^j}(p+1) := r_i^{\tau_r^j}(p) - R_i - P_i$

endfor

endfor

}

The following function main() is a direct application of the holistic scheduling approach.

main()

{

p:=0 /* step number */, end:=0

init()

repeat

compute_sending(p)

compute_receiving(p)

if for any NC_j and $\forall m_i \in \tau_s^j \cup \tau_{sr}^j$, $J_i^{\tau_s^j \cup \tau_{sr}^j}(p+1) := J_i^{\tau_s^j \cup \tau_{sr}^j}(p)$ then

end:=1

else if there is a message m_i and a controller $NC_j \in dest(m_i)$ such that

$$max\_rt(\tau_s^{src(m_i)} \cup \tau_{sr}^{src(m_i)}, m_i) + Max +$$
$$max_{NC_k \in relay(m_i) \text{ and } k \neq j}(max\_rt(\tau_r^k, m_i) - J_i^{\tau_r^k} + max\_rt(\tau_s^k \cup \tau_{sr}^k, m_i) - J_i^{\tau_s^k \cup \tau_{sr}^k}) +$$
$$Max + max\_rt(\tau_r^j), m_i) - J_i^{\tau_r^j} > D_i$$

then end:=1 else p:=p+1

until end=1

}

At the end of the procedure, the value of the maximum release jitter of any message m_i in any queue of any network controller is known, or the system

has diverged. Section 5.4 shows how to compute $r_i^{j,rel}$, the maximum time for any message m_i between the R_Multicast request and the time where the non-crashed controller NC_j in $dest(m_i)$ has R_Delivered the message m_i. Finally F_i, the maximum time between the R_Multicast request and the R_Deliver in the group *dest(m_i)* is computed. We now focus on Non-Preemptive EDF (NP-EDF) scheduling on a uniprocessor.

5.3 Uniprocessor NP-EDF scheduling

a) Notations

We first define the notations used for the NP-EDF scheduling analysis.

- **NP-EDF** is the non-preemptive version of Earliest Deadline First non idling scheduling. We recall that EDF schedules the tasks according to their absolute deadlines: the task with the shortest absolute deadline has the highest priority.
- Time is assumed to be **discrete** (task arrivals occur and tasks executions begin and terminate at clock ticks; the parameters used are expressed as multiples of the clock tick); in [2], it is shown that there is no loss of generality with respect to feasibility results by restricting the schedules to be discrete, once the task parameters are assumed to be integers (multiple of the clock tick) i.e. a discrete schedule exists if and only if a continuous schedule exists.

We now introduce the busy period notion that identifies periods of activity of the processor. The aim of the analysis given in 5.3.b is to identify the worst case busy periods (where the maximum response time of any task can be obtained).

- An **idle time** t is defined on a processor as a time such that there are no tasks requested before time t pending at time t. An interval of successive idle times is called an **idle period**. Notice that a task subject to a release jitter is not supposed to be pending until the release jitter ends.
- A **busy period** is defined as a time interval *[a,b)* such that there is no idle time in the interval *(a,b)* (the processor is fully busy) and such that both a and b are idle times.
- $U = \sum_{i=1}^{n} C_i/T_i$ is the **processor utilization factor**, i.e., the fraction of processor time spent in the execution of the task set [11]. In the following, we assume $U < 1$.
- Let time *0* be the beginning of a busy period *bp*. $\forall \tau_i$ and $\forall a \geq -J_i$, we define the sets: $hp_i(a) = \{\tau_k, k \neq i, D_k - J_k \leq a + D_i\}$ and $\overline{hp_i}(a) = \{\tau_k, D_k - J_k > a + D_i\}$. By construction, if $\tau_k \in hp_i(a)$, at least the occurrence of task τ_k whose activation is requested at time $-J_k$ and subject to a release jitter J_k has a higher priority than task τ_i whose activation is requested at time a. All occurrences of tasks in $\overline{hp_i}(a)$ (if any) have a priority lower than task τ_i whose activation is requested at time a whatever their release time in *bp*.

b) Worst case response time

We now consider the problem of scheduling by NP-EDF n sporadic tasks on a uniprocessor. This analysis is based on [5]. The worst case response time is reached for any task τ_i subject to a release jitter J_i in a scenario described in lemma 5.

Lemma 5. *Let $\tau_i, i \in [1,n]$ be a task requested at time a, subject to a release jitter J_i and executed according to NP-EDF in a busy period bp. The worst case response time of τ_i requested at time a, $a \geq -J_i$, is obtained in the first busy period of a scenario where:*

- *any task $\tau_k \in hp_i(a)$ executed in bp is requested for the first time at time $t_k^0 = -J_k$, with an initial release jitter J_k, and is then periodic.*
- *all the previous occurrences of τ_i executed in bp are periodic from t_i^0 to a, where $t_i^0 \in [-J_i, -J_i + T_i - 1]$.*
- *a task in $\overline{hp_i}(a)$ whose duration is maximum over $\overline{hp_i}(a)$ (if any) is released at time -1.*

Proof. see [5] for a detailed proof.

The following schema illustrates the worst case scenario of lemma 5 for a task requested at time a in a busy period *bp*.

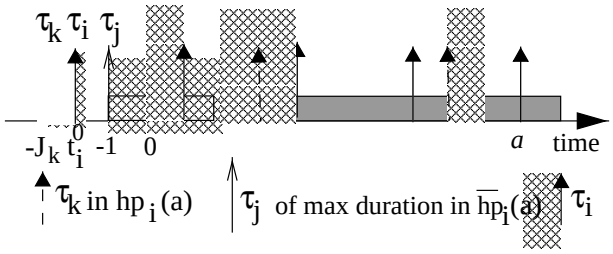

Fig. 4. Worst case scenario for task τ_i

Lemma 6. *If $U < 1$ then the longest busy period L is obtained in the first busy period of the scenario where all tasks $\tau_k, k \in [1,n]$ are subject to a release jitter J_k and are requested for the first time at time $-J_k$ and then at their maximum rate.*

Proof. Suppose there exists a busy period *BP* longer than L the first busy period of the scenario where all tasks $\tau_k, k \in [1,n]$ are requested at time -J_k. Let time *0* be the beginning of that busy period. Let t_k^0 be the first time task τ_k, is requested in the busy period. If $\forall k \in [1,n]$, t_k^0 is changed to -J_k and task τ_k is subject to a release jitter J_k then the resulting busy period can only see more or the same number of tasks and thus its size cannot decrease invalidating the former hypothesis.

L is the first solution of the equation: $L = \sum_{i=1}^{n} \lceil \frac{L+J_i}{T_i} \rceil C_i$. It is shown in [4] that L always exists when $U < 1$.

- Let $L_i(a)$ denotes the time when τ_i's instance requested at time $a \geq -J_i$ can start its execution in a scenario of lemma 5.

- The response time of the instance of τ_i requested at time a is: $r_i(a) = max\{C_i, L_i(a) + C_i - a\}$

The value of $L_i(a)$ can be determined by finding the smallest solution of the equation (1)

$$t = max_{\tau_k \in \overline{hp}_i(a)}\{C_k - 1\} + \overline{W}_i(a,t) + \lfloor \frac{a + J_i}{T_i} \rfloor C_i \tag{1}$$

The first term on the right side accounts for the worst-case priority inversion w.r.t. the absolute deadline $a+D_i$. The second term is the time needed to execute the instances of tasks other than τ_i with absolute deadlines smaller than or equal to $a + D_i$ and request times before the execution start time of τ_i's instance. Finally, the third term is the time needed to execute the τ_i's instances requested before a. The rationale of the equation is to compute the time needed by the τ_i's instance released at time a to get the processor: every other higher priority instance released before this event will be executed earlier, thus its execution time must be accounted for. For the same reason, the function $\overline{W}_i(a,t)$ must account for all higher priority instances of task $\tau_j \in hp_i(a)$ requested in the interval $[-J_j, t]$ (according to scenario of lemma 5), thus also including those possibly requested at time t. For any task τ_j, the maximum number of instances requested in $[-J_j, t]$ is $1 + \lfloor \frac{t+J_j}{T_j} \rfloor$. However, among them at most $1 + \lfloor \frac{a+D_i-D_j+J_j}{T_j} \rfloor$ can have an absolute deadline less than or equal to $a+Di$. It follows that :

$$\overline{W}(a,t) = \sum_{\tau_j \in hp_i(a)} (1 + \lfloor \frac{min(t, a + D_i - D_j) + J_j}{T_j} \rfloor) C_j \tag{2}$$

$\overline{W}_i(a,t)$ being a monotonic non-decreasing step function in t, the smallest solution of Equation (1) can be found using the usual fixed point computation:

$$L_i^{(0)} = 0 \tag{3}$$

$$L_i^{(p+1)}(a) = max_{\tau_j \in \overline{hp}_i(a)}\{C_j - 1\} + \overline{W}_i(a, L_i^p(a)) + \lfloor \frac{a + J_i}{T_i} \rfloor C_i \tag{4}$$

Having computed $L_i(a)$, we obtain $r_i(a)$. Notice that according to the lemma 6, $L_i(a)$ is upper bounded by L. Hence we conclude that the computation of $r_i(a)$ can be coherently limited to values of a smaller than L. That is, the worst-case response time of τ_i is finally

$$r_i = max\{r_i(a) : -J_i \leq a \leq L\}. \tag{5}$$

5.4 End-to-end response times

We show how the notations used in the uniprocessor case are extended to the distributed case. We can then compute $r_i^{j,rel}$, the maximum time between the R_Multicast request and the R_Deliver of a message m_i by the non-crashed network controller NC_j in *dest(m_i)*.

We first study the effect of the clock precision ε:
Two messages timestamped by the same source are not affected by the clock precision. If two messages are timestamped by two different network controllers then the precision of the clock synchronization has an effect equivalent to an extra release jitter ε to add in the worst case analysis of section 5.3. In the following, we denote $\varepsilon_{i,k}$ the function that returns for any message m_i and m_k in the same waiting queue, ε if $src(m_k) \neq src(m_i)$ and *0* otherwise.
In the uniprocessor case, a task τ_i is defined by the parameters (C_i, T_i, D_i, J_i). In the distributed case, task τ_i is associated to a message m_i. The values of the former parameters must be adapted on any network controller NC_j as follows:

The execution time C_i: already given in section 5.2.

The release jitter J_i: if $J_i^{\tau^j}(p-1)$ is the release jitter obtained for message m_i in the waiting queue τ^j at step *p-1* in the holistic scheduling approach (see section 5.2) then for any message m_k in τ^j, $J_i^{\tau^j} = J_i^{\tau^j}(p-1)+\varepsilon_{i,k}$. Indeed, the worst case behavior of sources different from *src(m_i)* is to have timestamped their messages with a clock ε late compared to the local clock of *src(m_i)*.

The parameters Ti and Di are left unchanged.
With those adaptations, we can determine the worst case response time of any task τ_i in either the send queue or the receive queue. When the holistic scheduling iteration ends, the system has either converged or diverged. If the system has converged then the worst case release jitter of any message m_i in τ^j, with $\tau^j = \tau_s^j \cup \tau_{sr}^j$ or $\tau^j = \tau_r^j$, is determined on any network controller NC_j.
We now show how to compute $r_i^{j,rel}$ defined in section 5.2 as the maximum time between the R_Multicast request of message m_i and the R_Deliver by the non-crashed network controller NC_j in *dest(m_i)*. According to figure 2, a message m_i can be received from different network controllers, either directly or through a relay. Then $r_i^{j,rel}$ is the maximum between:

- the maximum time between the R_Multicast request and the R_Deliver of message m_i by controller NC_j in $dest(m_i)$, after a direct receipt of message m_i;

- the maximum time between the R_Multicast request and the R_Deliver of message m_i by controller NC_j in $dest(m_i)$, after an indirect receipt of message m_i.

If m_i is received directly by NC_j, then the worst duration between the R_Multicast request of m_i and the R_Deliver of m_i on NC_j is equal to:

$$max\_rt(\tau_s^{src(m_i)} \cup \tau_{sr}^{src(m_i)}, m_i) + Max + max\_rt(\tau_r^j, m_i) - J_i^{\tau_r^j} \qquad (6)$$

If m_i is received indirectly by NC_j, then the worst duration between the R_Multicast request of m_i and the R_Deliver of m_i on NC_j is equal to:

$$\begin{aligned} & max\_rt(\tau_s^{src(m_i)} \cup \tau_{sr}^{src(m_i)}, m_i) + Max + \\ & max_{NC_k \in relay(m_i)}(max\_rt(\tau_r^k, m_i) - J_i^{\tau_r^k} + \\ & max\_rt(\tau_s^k \cup \tau_{sr}^k, m_i) - J_i^{\tau_s^k \cup \tau_{sr}^k}) + Max + \\ & max\_rt(\tau_r^j, m_i) - J_i^{\tau_r^j} \end{aligned} \qquad (7)$$

$r_i^{j,rel}$ being computed for any message m_i. Finally the worst case end to end response time for the reliable multicast of message m_i is $F_i + \varepsilon$, where $F_i = max_{j \in dest(m_i)} r_i^{j,rel}$.
Hence the system is feasible if the worst case end to end response time of every message is less than its relative deadline, i.e.:

$$F_i + \varepsilon \leq D_i \qquad (8)$$

where D_i is the end-to-end deadline required for the reliable multicast of message m_i in the group $dest(m_i)$.

6 Conclusion

In this paper, we have designed and proved a uniform reliable multicast protocol tolerating up to f_c processor crashes and up to f_o network omissions during the execution of a reliable multicast. The total number of messages generated per reliable multicast of message m_i is $f_c + f_o + 2$ in a broadcast network and $(f_c + f_o + 2)[card(dest(m_i) - 1]$ in a point-to-point network, when the source of the multicast belongs to the destination group $dest(m_i)$. The fault-tolerance feasibility condition is $card[dest(m_i) - \{src(m_i)\}] \geq f_c + f_0 + 1$. The network transmission delay is assumed to be bounded. The end-to-end worst case response time is analysed using the holistic approach. We have established a necessary and sufficient condition for the feasibility on a uniprocessor, of a set of tasks subject to a release jitter and scheduled EDF. These results are then applied to the computation of the end-to-end response time. In [4] the complexity is shown to be pseudo-polynomial when $U < 1$, U being the processor utilization factor. This uniform reliable multicast service ensures that for any message m_i whose reliable multicast is requested at time t, no controller delivers m_i after time $t + F_i + \varepsilon$, where F_i is a constant depending on m_i and on the workload of processors involved by the multicast. The real-time feasibility condition is then $F_i + \varepsilon \leq D_i$, where D_i is the deadline of message m_i.

References

[1] B. Charron-Bost, C. Delporte, H. Fauconnier, E. Ledinot, G. Le Lann, A. Marcuzzi, P. Minet, J.M. Vincent, "Spécification du problème ATR (Accord Temps Réel)", rapport du contrat DGA- DSP/MENSR/CNRS, Juin 1997.

[2] S. K. Baruah, R. R. Howell, L. E. Rosier, "Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic Real-Time tasks on one processor", Real- Time Systems, 2, p 301-324, 1990.

[3] F. Cristian, "Synchronous atomic broadcast for redundant channels", IBM Research Report RJ 7203, Dec. 1989.

[4] L. George, "Ordonnancement en ligne temps réel critique dans les systèmes distribués", PhD Thesis, Université de Versailles Saint-Quentin, Janvier 1998.

[5] L. George, N. Rivierre, M. Spuri, "Preemptive and Non-Preemptive Real- Time Uniprocessor Scheduling" INRIA Rocquencourt, France, Research Report 2966, Sept. 1996.

[6] R. Guerraoui, A. Schiper, "Total Order Multicast to Multiple Groups", ICDCS'97, Baltimore, Maryland, pp 578- 585, May 1997.

[7] J. F. Hermant, M. Spuri, "End-to-end response times in real-time distributed systems", 9th int. conf. on Parallel and Distributed Computing Systems, Dijon, France, pp 413-417, Sept. 1996.

[8] V. Hadzilacos, S. Toueg, "A modular approach to fault-tolerant broadcasts and related problems", Technical Report TR 94-1425, Dept. of Computer Science, Cornell University, Ithaca, NY14853, May 1994.

[9] K. Jeffay, D. F. Stanat, C. U. Martel, "On Non-Preemptive Scheduling of Periodic and Sporadic Tasks", IEEE Real-Time Systems Symposium, San-Antonio, December 4-6, 1991, pp 129-139.

[10] D. D. Kandlur, K. G. Shin, D. Ferrari, "Real-time communication in multihop networks", IEEE Trans. on Parallel and Distributed Systems, 5(10), October 1994.

[11] C.L Liu, J. W. Layland, "Scheduling Algorithms for multiprogramming in a Hard Real Time Environment", Journal of the Association for Computing Machinery, 20(1), Jan. 1973.

[12] G. Le Lann, "A methodology for designing and dimensioning critical complex computing systems", IEEE Engineering of Computer Based Systems, Friedrichshafen, Germany, March 1996, pp 332-339.

[13] A. Parekh, R. Gallager, " A generalized processor sharing approach to flow control in integrated services networks- the single node case", INFOCOM92, Florence, Italy, 1992, pp915-924.

[14] F. Schmuck, F. Cristian, "Continuous clock amortization need not affect the precision of a clock synchronization algorithm", Research Report RJ7290, IBM Almaden, Jan. 1990.

[15] M. Spuri, "Holisitic analysis for deadline scheduled real-time distributed systems" Research Report 2873, INRIA, France, April 1996.

[16] K. Tindell, J. Clark "Holistic Schedulability Analysis for Distributed Hard Real-Time Systems", Microprocessors and Microprogramming 40, 1994.

[17] G Xie, S. Lam, "Delay Guarantee of virtual clock server", IEEE/ACM Trans. on Networking, 3(6), pp 683-689, Dec. 1995.