Causal Deliveries in Unreliable Networks With Real-Time Delivery Constraints

Roberto Baldoni, Achour Mostefaoui, Michel Raynal

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Causal Deliveries in Unreliable Networks With Real-Time Delivery Constraints

R. Baldoni, A. Mostefaoui and M. Raynal

N° 2427

Décembre 1994

PROGRAMME 1

ISSN 0249-6399

Causal Deliveries in Unreliable Networks With Real-Time Delivery Constraints

R. Baldoni*, A. Mostefaoui and M. Raynal

Programme 1 — Architectures parallèles, bases de données, réseaux
et systèmes distribués
Projet Adp

Rapport de recherche n°2427 — Décembre 1994 — 18 pages

Abstract: *Causal order* states that for any process the order in which it is delivered messages cannot violate the *happened-before* relation of the corresponding sendings. The aim of this communication abstraction is to cope with the *asynchrony* of communication channels in distributed systems. This abstraction has been defined for distributed systems without real-time delivery constraints. In this paper we extend this abstraction to cope with unreliable communication networks with real-time delivery constraints: messages have a *lifetime*, Δ, after which their contents can no longer be used, moreover some of them can be lost. This new abstraction, called Δ-*causal order*, requires to deliver as much messages as possible within their lifetime in such a way that these deliveries respect causal order. A simple efficient implementation is proposed. Examples of distributed multimedia real-time applications, in which scheduling messages deliveries respecting Δ-causal order is a crucial point for the quality of the service, are given.

Key-words: Causal order, distributed systems, multimedia, real-time.

(Résumé : tsvp)

Work partially supported by the following Basic Research Action Programs of the European Community: the HCM project under contract 3702 "CABERNET" and the ESPRIT project under contract 6360 "BROADCAST".

*On leaving from the Dipartimento di Informatica e Sistemistica, University of Roma "La Sapienza", Italy. This author is also supported by a grant of the *Consiglio Nazionale delle Ricerche* under contract No.93.02294.CT12.

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : (33) 99 84 71 00 – Télécopie : (33) 99 84 71 71

Livraisons causales dans les réseaux non fiables avec contraintes temps-réel

Résumé : L'extension aux livraisons de messages de la relation de précédence existant entre leurs émissions s'appelle ordre causal. Ce mode de communication permet de réduire l'asynchronisme de la communication dans les systèmes distribués. Cette abstraction a été définie, à l'origine, pour les systèmes distribués fiables et sans contraintes temps-réel sur les livraisons. Dans cet article, elle est étendue afin de prendre en compte les pertes de messages et les contraintes temps-réel sur les livraisons : les messages ont une durée de vie Δ après laquelle leurs contenus deviennent sans aucune utilité pour le destinataire. Cette nouvelle abstraction, appelée Δ-ordre causal, consiste à délivrer le maximum de messages avant l'expiration de leur durée de vie tout en respectant l'ordre causal sur les livraisons. Un protocole simple est proposé et des exemples d'applications temps-réel multimédia nécessitant des livraisons de messages respectant le Δ-ordre causal sont présentés.

Mots-clé : multimédia, ordre causal, systèmes distribués, temps-réel.

1 Introduction

Introduced by Birman and Joseph [BJ87], *causal order* is one of the most interesting and known communication modes defined in order to cope with the *asynchrony* of communication channels in distributed systems. Such a communication mode has been proved to be very useful as soon as one is interested in taking snapshots of distributed applications [AV94], in controlling distributed applications, in managing replicated data [BJ87] and in allowing consistent observations of distributed computations [RST91]. Basically it frees the programmer from inconsistencies due to transmission delays over the distributed system.

Causal order means that if two sends are causally related [Lam78] and concern the same destination process, then the corresponding messages are delivered in their sending order. Such a communication mode has been defined for distributed systems without real-time delivery constraints. In particular, it has been assumed that messages cannot be lost and transmission delays are unpredictable but finite. Protocols implementing this communication mode have been proposed in [BSS91, RST91]: as pointed out by [Ver94] these protocols are not concerned by real-time issues. In this paper we are interested in defining a notion of causal order, called Δ-*causal order*, for unreliable distributed systems with real-time delivery constraints. In this setting messages can be lost and have a *lifetime*, Δ, after which their data can no longer be used. In other words, messages whose transmission delays are greater than Δ are considered as actually lost. Hence, for a message m the instant *sending time of* $m + \Delta$ can be seen as a deadline after which m's content is obsolete. Δ-*causal order* requires to deliver as much messages as possible before their deadlines in such a way that these deliveries respect causal order.

To capture the causality relation among messages without real-time delivery constraints, vector timestamps or matrices of sequence numbers have been used in pure causal order protocols [BSS91, RST91]. Such logical information cannot cope with real-time systems in which the notion of physical time is required in order to check timing constraints on messages deliveries [Ver94]. Although, unfortunately, it is impossible to have a common physical clock in a distributed system, there exist clock synchronization protocols that allow all the participants across a distributed system to agree on a common clock value whose drift with respect to physical time remains always bounded [GZ89, Mil91].

Δ-causal order communication mode is particularly useful for *multimedia real-time collaborative applications* [Yav92] or *groupware real-time applications* [GM94] such as teleconferencing, shared window systems, videophone systems and catalog browsing services, where participants need to exchange real-time *audio* and *video* in-

formations (messages) over a communication network. In such applications a source generates a flow of messages with which is associated a maximun transmission delay that must be respected to save the real-time interaction and the quality of the service [Wak93]. Moreover, such applications may tolerate a number of lost messages as long as these losses do not cause a major degradation of their services [HB94, Fer90]. For example in teleconferencing systems, that is one of the multimedia services that needs the strongest delivery constraints, an analogic video signal is sampled at the source site and produces a continuous flow of messages. In order to reconstruct a high quality analogic signal at the destination site and to have a good interactivity, messages should be delivered in the order they were sent and within a well-defined deadline (under 250 msec. for video signals [JSS94]). Messages out of order or out of their lifetime must be discarded since their deliveries could hardly deteriorate the quality of the service [HB94, Yav92]. Hence, it has been argued in [Wak93] that, from the point of view of communication networks, one of the more challenging problems in developing multimedia real-time applications is to cope with loss of messages, ordering of messages and arbitrary transmission delays. Δ-causal order is a very powerful tool to cope with such a problem.

The paper is structured in four main sections. Section 2 presents the model of asynchronous distributed executions. Section 3 gives the formal definition of Δ-causal order. Section 4 presents a protocol implementing this communication mode. Its formal proof of correctness is shown in Section 5.

2 Model of distributed executions

2.1 Distributed programs

A distributed program is a set P of n sequential processes $\{P_1, P_2, \ldots, P_n\}$ that communicate and synchronize only by exchanging messages. The underlying system, where distributed programs execute, is composed of n processors (for simplicity's sake, we assume one process per processor) that can exchange messages. When a message arrives on a channel, it can be delivered as soon as a delivery condition becomes true; in an asynchronous system with no special constraints on deliveries* this condition is always true. We assume that each pair of processes is connected by an asynchronous and unreliable logical channel (messages can be lost and transmission delays of non lost messages are unpredictable). Processors do not have a shared memory and there is no bound for their relative speeds.

*Examples of special constraints in deliveries are fifo order, Causal Order and Δ-causal order.

2.2 Distributed executions

Execution of a process produces a sequence of events which can be classified as: *send* events, *delivery* events and *internal* events. An internal event may change only local variables; send or delivery events involve communication. The causal ordering of events in a distributed execution is based on Lamport's *happened-before* relation [Lam78] denoted $\rightarrow$. If a and b are two events then $a \rightarrow b$ iff one of these conditions is true:

(i) a and b are produced on the same process with a first;

(ii) $a = send(m)$ is the send event of a message m and $b = delivery(m)$ is the delivery event of the same message;

(iii) there exists an event c such that $a \rightarrow c$ and $c \rightarrow b$.

Such a relation allows to represent a distributed executions as a partial order of events, called $\widehat{E} = (E, \rightarrow)$ where E is the set of all events. Hereafter, we call $M(\widehat{E})$ the set of all messages exchanged in $\widehat{E}$ and we do not consider internal events since actually they do not affect the causal ordering of events.

2.3 Lifetime of a message

The lifetime, Δ, of a message is the physical time duration, after its sending time, during which such a message can be used by its destination process. A message that arrives at its destination after its lifetime is useless and must be discarded; for processes such messages are actually lost. A message that arrives at its destination within its lifetime must be delivered at the target process within the expiration of its lifetime. For simplicity's sake we assume a unique lifetime for all the messages (in case of types of messages with different lifetimes we can assume the minimum among them as the unique lifetime to be respected by all the messages). As indicated in the introduction, for a message m the instant *sending time of* $m + \Delta$ is actually a deadline after which this message is useless for the destination process.

In multimedia systems, the lifetime of a message is the maximum transmission delay that an application can tolerate before delivering a message before the quality of its service degrades. This delay is strictly connected to the type of multimedia applications. For example, in teleconferencing the maximum lifetime for audio and video messages is 250 msec.[JSS94] (an acceptable one is 100 msec. [RB93]), videophones systems may tolerate a lifetime of 350 msec., non-interactive video systems

have a lifetime of 1 sec. and catalog browsing services have a lifetime of 2 sec. [RB93, Fer90].

2.4 Global clock

In order to ensure the respect of real-time delivery constraints, processes are endowed with a global clock value whose drift with respect to physical time is bounded and whose granularity and precision [KO87, Ver94] are such that all the causally dependent events are produced at different times. Many clock synchronization protocols have been described in the literature and some of them are currently used and provide a global clock synchronization that bounds the clock drift value, ϵ, within a few milliseconds (5-10 msec.) [GZ89, Mil91]. Without loss of generality, we assume in the following that ϵ is incorporated in the lifetime Δ of the message.

3 Δ-causal order abstraction

3.1 Causal order

Notion of causal order has been introduced by Birman and Joseph in [BJ87]; this notion relies on a reliable communication system (all the messages will be eventually delivered) and no bound is imposed on messages deliveries (transmission delays are unpredictable but finite). *Causal order* states that for any process the order in which it is delivered messages cannot violate the *happened-before* relation of the corresponding sendings. More formally,
Definition [Causal Order]: *A distributed computation $\widehat{E}$ respects causal order if for any two messages $m1$ and $m2 \in M(\widehat{E})$ we have:*

> **if** $send(m1) \rightarrow send(m2)$ **and** $m1$, $m2$ *have the same destination process,*
>
> **then** $deliver(m1) \rightarrow deliver(m2)$.

A first implementation of such an abstraction has been embedded in the ISIS system [BJ87] and recently, more refined protocols implementing causal order have been proposed [BSS91, RST91]. Such implementations add a protocol over the original underlying system such that causal order is never violated at the process level. Basically, deliveries of messages to the process level are done by delaying those messages arrived too early at the underlying system level.

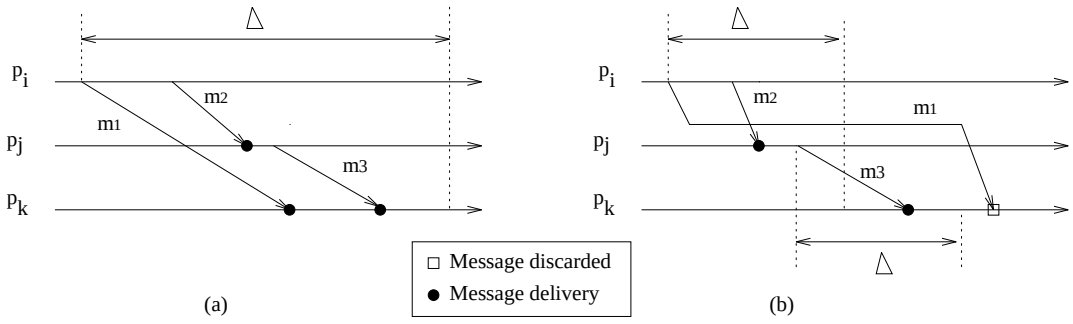

Figure 1: Δ-causally ordered communications

3.2 Δ-causal order

As indicated previously causal order can always be offered to processes in a reliable non real-time system. Each message is associated with a delivery condition that will become true sooner or later. In a system with real-time delivery constraints, we have to take into account that messages can be discarded if they arrive at their destinations after the expiration of their deadlines. This motivates the following definition of Δ-causal order.

Definition [Δ-Causal Order]: *A distributed computation $\widehat{E}$ respects Δ-causal order if:*

> *i. all messages in $M(\widehat{E})$ that arrive within Δ are delivered within Δ, all the others are never delivered (they are lost or discarded);*
>
> *ii. all delivery events respect causal order.*

Note that Δ-causal order meets causal order definition when channels are reliable and Δ is greater than any message transmission delay over the distributed system. Moreover, Δ can be practically considered as a tuning knob that influences the number of messages discarded while ensuring deliveries respect causal order. Figure 1 shows two examples of Δ-causally ordered communications, in Figure 1.b events are Δ-causally ordered since $m1$ is discarded. Figure 2 presents examples of violation of Δ-causal order: condition i (respt. ii) is violated in Figure 2.a (respt. Figure 2.b).

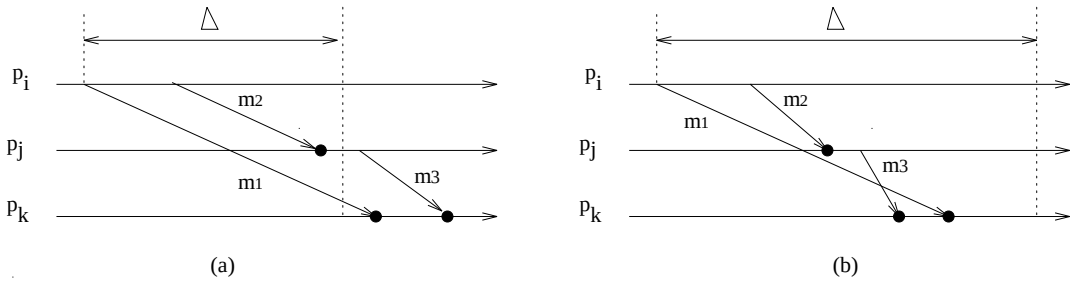

Figure 2: Non-Δ-causally ordered communications

4 A protocol implementing Δ-causal order

An implementation of Δ-causal order communication mode consists in adding a protocol over the original underlying system such that events of distributed computations be Δ-causally ordered at the process level. At the underlying system level three types of events are associated with each message: *send*, *arrival* and *delivery* (only *send* and *delivery* events are visible to processes). Due to delivery constraints imposed by Δ-causal order, when a message arrives either it is discarded (if it has missed its deadline) or it can be delayed until its delivery condition becomes true (if the message arrives within its deadline but some, not yet arrived, messages causally preced it and their deadlines have not yet expired). An example of scheduling of messages deliveries, due to Δ-causality, is shown in Figure 3.a.

The core of the protocol then consists in defining the delivery condition DC(m) associated with each message m in such a way that m *is delivered as soon as possible* i.e., when all messages that causally precedes m either have been delivered (Figure 3.a) or have been lost or they are still in transit when their deadlines expire (Figure 3.b). At this end, data structures managed by each process and control information carried by each message are defined.

Global information

As described in Section 2.4 we assume a clock synchronization protocol providing processes with a unique value representing the global physical time: so each process has a variable *current_time* of type *time* that is continuously updated by this underlying clock synchronization protocol. The domain of type *time* is the set of natural integers.

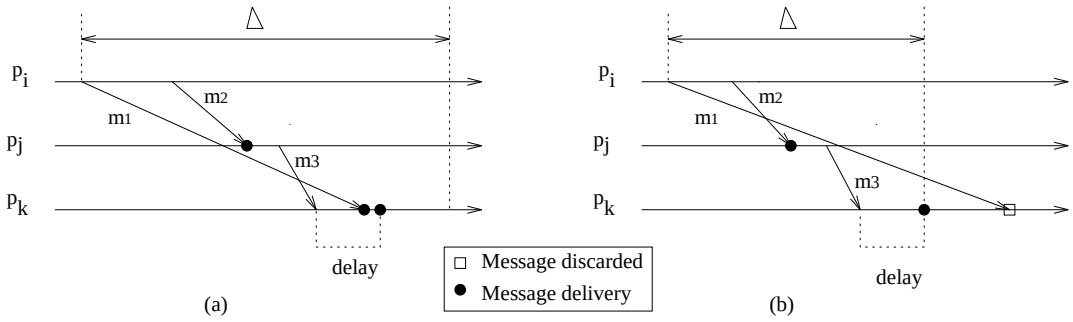

Figure 3: Examples of "as soon as possible" deliveries

procedure SEND(m, i, j): % m is the message to be sent by P_i to P_j %
begin % this procedure is executed atomically %
  $send\_time_m := current\_time$; $ST_m := SENT_i$; (1)
  send $(m, ST_m, send\_time_m)$ **to** P_j; (2)
  $SENT_i[i, j] := send\_time_m$; (3)
end. (4)

when $(m, ST_m, send\_time_m)$ **arrives at** P_i **from** P_j:
begin
  if $send\_time_m + \Delta < current\_time$ (5)
  then discard m (6)
  else wait $(\forall x : (ST_m[x, i] \leq DEL_i[x]) \vee (ST_m[x, i] < current\_time - \Delta))$; (7)
    % The following statements are executed atomically %
    % as soon as the delivery condition DC(m) becomes true %(8)
    % If DC($m1$) and DC($m2$) become simultaneously true %
    % then, if $ST_{m1} < ST_{m2}$, $m1$ is delivered before $m2$ %(9)
    $SENT_i[j, i] := send\_time_m$; (10)
    $DEL_i[j] := send\_time_m$; (11)
    $\forall x, y : SENT_i[x, y] := max(SENT_i[x, y], ST_m[x, y])$; (12)
    deliver(m); % (13)
  fi (14)
end.

Figure 4: the Δ-causal order protocol

Local information of a process Each process manages the following two variables:

$SENT_i$: $array[1..n, 1..n]$ *of time*;

DEL_i : $array[1..n]$ *of time*;

The variable $SENT_i[x, y]$ represents the knowledge of process P_i about the sending time of the last message sent by P_x to P_y. The variable $DEL_i[j]$ represents the sending time of the last message sent by P_j to P_i and delivered to P_i. These arrays are initialised to a value lower than the initial value of the variable *current_time*.

Information carried by each message
Each message m piggybacks the following control information:

ST_m is a copy of the matrix $SENT$ of the sending process at m's sending time;

$send\_time_m$ is the time at which m has been sent (occurrence of the associated send event).

Moreover, let ST_{m1} and ST_{m2} be the matrices piggybacked respectively by messages $m1$ and $m2$. By definition: $ST_{m1} < ST_{m2} \Leftrightarrow \forall (x, y), ST_{m1}[x, y] \leq ST_{m2}[x, y]$ and $\exists (x, y), ST_{m1}[x, y] < ST_{m2}[x, y]$.

Behavior of a process
Upon its arrival a message m will be immediately discarded if it misses its deadline. If m is not discarded, it will be eventually delivered within its deadline according to the definition of Δ-causal order. As indicated previously its delivery is determined by a *delivery condition*. Message m can be delivered as soon as the following predicate DC(m) is true: all messages that causally preced m either have been delivered or their deadlines have expired. More formally, the delivery condition of message m sent to P_i is expressed as:

$$\forall\ x : (ST_m[x, i] \leq DEL_i[x]) \vee (ST_m[x, i] < current\_time - \Delta) \qquad \text{DC}(m)$$

The whole protocol is described in Figure 4. When message m, send by P_j, is delivered to P_i, state variables DEL_i and $SENT_i$ are updated according to their definitions (remark that $SENT_i[j, i]$ and $DEL_i[j]$ have always the same value (lines 10 and 11); so $DEL_i[j]$ could be replaced by $SENT_i[j, i]$).

Note that if we assume reliable channels and Δ greater than any message transmission delay over the network, we have no longer real-time delivery constraints and then by replacing the physical clock with a logical one we obtain the protocol presented in [RST91].

5 Correctness proof

To prove that our algorithm guarantees Δ-causal order, we use two steps. In the first one we show that all messages received within their deadlines will be delivered within their deadlines and the others discarded (*liveness property*). Secondly, we prove that all delivery events respect causal order (*safety property*).

5.1 Liveness

Before proving *liveness property*, let us remark that, because of the definition of $SENT$ array and the continuous progress of the variable $current\_time$ (see Section 2.4), it follows that for each message m, we have the following relation:

$$send\_time_m > ST_m[x,y] \quad \forall\ (x,y) \tag{P}$$

Theorem 1

(i) All messages arrived after the expiration of their deadlines will be discarded and (ii) all messages arrived within their deadlines will be delivered within their deadlines.

Proof

Point i follows from the test (line 5) of the protocol of Figure 4.

Point ii is proved by contradiction. Suppose that there exists a message m that arrived within its deadline and has not been delivered within its deadline. Hence, on its deadline, from the delivery condition DC(m) (line 7), the following condition NDC follows:

$$\exists x : (ST_m[x,i] > del_i[x]) \wedge (ST_m[x,i] \geq current\_time - \Delta) \tag{NDC}$$

On the deadline of message m we have: $current\_time = send\_time_m + \Delta$. So the second term of NDC becomes: $\exists x : (ST_m[x,i] \geq send\_time_m)$. But this contradicts property P.

It follows that, at the deadline of an arrived message m, NDC is false contradicting our initial assumption. Therefore for any message m, arrived before its deadline, the delivery condition DC(m) will be verified before its deadline expires.

□

5.2 Safety

Lemma L1

Each variable $SENT_i[x,y]$ *(*$\forall i,x,y$*) does not decrease.*

The proof follows directly from the protocol (lines 3, 10 and 12).

Lemma L2

Consider a pair of messages $m1$ *and* $m2$ *sent respectively by* P_i *and* P_j *such that* $m1$ *has been sent to* P_l*. We have:* $send(m1) \rightarrow send(m2) \Rightarrow ST_{m1} < ST_{m2}$ *with* $ST_{m1}[i,l] < send\_time_{m1} \leq ST_{m2}[i,l]$.

Proof

Label the *happened before* relation between two *send* events $send(m1)$ and $send(m2)$ by a non negative integer k in the following way; k represents the number of messages that establish the causal path from $send(m1)$ to $send(m2)$ (by definition k does not include $m1$). So, we have:

- $send(m1) \stackrel{0}{\rightarrow} send(m2)$ iff $m1$ and $m2$ have the same sender or the sender of $m2$ has been delivered $m1$ before sending $m2$ (see Figure 5.a and Figure 5.b).
- $send(m1) \stackrel{k>0}{\rightarrow} send(m2)$ iff $\exists m' : send(m1) \stackrel{k-1}{\rightarrow} send(m')$ and $send(m') \stackrel{0}{\rightarrow} send(m2)$.

The lemma L2 is proved by induction on k. Remember that P_i (respt. P_j) is the sender of $m1$ (respt. $m2$) and P_l is the destination process of $m1$.

1. $k = 0$ ($m1$ and $m2$ have the same sender $P_i = P_j$, Figure 5.a).

   At the sending of $m1$, we have (definition of $send\_time_{m1}$, lines 1 and 3): $send\_time_{m1} = SENT_i[i,l] = current\_time$. So at the sending of $m2$ by P_i, as $SENT_i[i,l]$ does not decrease (lemma L1), we have: $ST_{m2}[i,l] \geq send\_time_{m1}$. From property P, it follows that $ST_{m1}[i,l] < send\_time_{m1} \leq ST_{m2}[i,l]$.

   Further we have (because $m1$ and $m2$ have been sent by the same process, and by lemma L1) $\forall\ (x,y) : ST_{m1}[x,y] \leq ST_{m2}[x,y]$.

2. $k = 0$ ($m1$ and $m2$ have distinct senders P_i and P_j, and $P_l = P_j$, Figure 5.b).

   P_j has been delivered $m1$ before sending $m2$; at the delivery of $m1$ to $P_l (= P_j)$, we have:
   $SENT_l[i,l] > send\_time_{m1}$ (line 10).

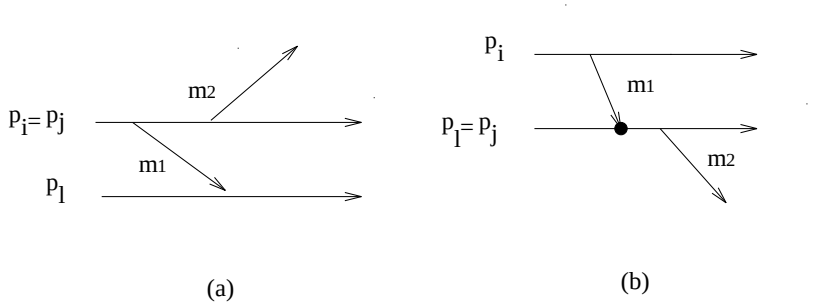

Figure 5: Proof of lemma L2

At the sending of $m2$ by P_l, as $SENT_l[i,l]$ does not decrease (lemma L1), we have: $SENT_l[i,l](= ST_{m2}[i,l]) \geq send\_time_{m1}$. So, from property P, it follows that $ST_{m1}[i,l] < send\_time_{m1} \leq ST_{m2}[i,l]$.

Moreover, as $\forall x, y : SENT_l[x,y]$ does not decrease (lemma L1) and as $m1$ is delivered to P_l (line 12) before $m2$ is sent by P_l (line 1), we have: $\forall\ (x,y) : ST_{m1}[x,y] \leq ST_{m2}[x,y]$.

3. $k > 0$.

   i. $send(m1) \stackrel{k-1}{\rightarrow} send(m')$:
   By the induction hypothesis we have: $ST_{m1}[i,l] < send\_time_{m1} \leq ST_{m'}[i,l]$ and
   $$\forall\ (x,y) : ST_{m1}[x,y] \leq ST_{m'}[x,y].$$

   ii. $send(m') \stackrel{0}{\rightarrow} send(m2)$:
   As $k = 0$, we have $\forall\ (x,y) : ST_{m'}[x,y] \leq ST_{m2}[x,y]$.

   From i and ii, it follows: $ST_{m1}[i,l] < send\_time_{m1} \leq ST_{m'}[i,l] \leq ST_{m2}[i,l]$ and
   $$\forall\ (x,y) : ST_{m1}[x,y] \leq ST_{m'}[x,y] \leq ST_{m2}[x,y]. \qquad \square$$

Lemma L3
Between two processes, delivery events respect fifo order.

Proof

Let (P_i, P_l) be a pair of processes with P_i (respt. P_j) being the sender (respt. the destination). All the messages considered in this proof are sent from P_i to P_l. We consider a couple of messages $m1$ and $m2$, *consecutive* in the following sense: both will de delivered to P_l and, among all messages sent by P_i after $m1$ and delivered to P_l, $m2$ is the first the sending order. The following proof shows that $m1$ is delivered before $m2$ to P_l.

As $m1$ is sent by P_i before $m2$, we have (lemma L2):

$$\forall\ (x,y)\ ST_{m1}[x,y] \leq ST_{m2}[x,y] \text{ and}$$
$$ST_{m1}[i,l] < send\_time_{m1} \leq ST_{m2}[i,l].$$

Moreover, at the delivery of $m2$, we have DC($m2$):

$$ST_{m2}[i,l] \leq DEL_l[i] \vee ST_{m2}[i,l] \leq current\_time - \Delta.$$

case 1: Suppose $m1$ has arrived when DC($m2$) becomes true. Due to lemma L2, DC($m1$) becomes true, in the worst case, at the same time as DC($m2$). If DC($m1$) becomes true before DC($m2$), $m1$ is delivered before $m2$ (line 8); if it comes true at the same time, as we have $ST_{m1} < ST_{m2}$, $m1$ is delivered before $m2$ (line 9).

case 2: Suppose that $m1$ has not yet arrived when DC($m2$) becomes true.

case 2.1: if $ST_{m2}[i,l] \leq current\_time - \Delta$ then we have (lemma L2):
$send\_time_{m1} \leq ST_{m2}[i,l] \leq current\_time - \Delta$, and consequently $m1$ will be discarded at its arrival (line 5). This contradicts our hypothesis (namely $m1$ is delivered).

case 2.2: if $ST_{m2}[i,l] \leq DEL_l[i] = \alpha$ then as
$ST_{m2}[i,l] \geq send\_time_{m1}$, we have $DEL_l[i] = \alpha \geq send\_time_{m1}$.

As the only update (increase) to $DEL_l[i]$ is at a message delivery (line 11), this means a message m' sent by P_i after $m1$ has been delivered to P_l. This message has been sent after $m2$ (as we are evaluating the delivery condition of $m2$, this message has not yet been delivered; Figure 6). If there are several such messages sent after $m2$ and delivered before it, let m' be the first of these messages in their sending order. We have: $ST_{m'}[i,l] \geq send\_time_{m2} > ST_{m2}[i,l]$ (Figure 6).

Now, we consider the same reasoning, with $m2$ and m' replacing respectively $m1$ and $m2$. As m' has been delivered, it follows that a message m'' sent after m' has been delivered to P_l before m'; and as m'' has been delivered to P_l, it follows that a message m''', etc. So an infinity of messages sent after $m1$ must be delivered to P_l before $m1$; this is impossible as, if $DEL_l[i] = \alpha$, only

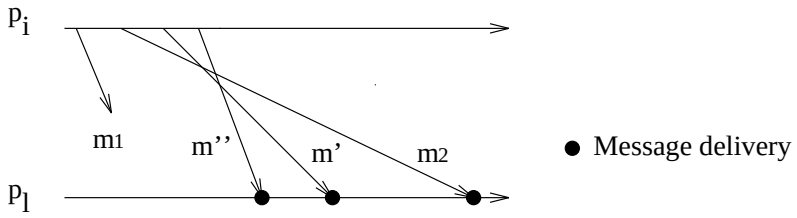

Figure 6: Proof of lemma L3

a maximum of α messages have been delivered to P_l from P_i (remember that, due to the granularity of the clock, as described in Section 2.4, two messages sent by the same sender have distinct sending times). It follows $m1$ has been delivered before $m2$ to P_l.

□

Theorem 2

Delivery events respect causal order.

Proof

Let us consider two messages $m1$ and $m2$ sent to process P_l respectively by processes P_i and P_j and let us suppose that they have been delivered out of causal order (i.e. $send(m1) \rightarrow send(m2)$ and $deliver(m2) \rightarrow deliver(m1)$). When $m2$ is delivered to P_l, its delivery condition (DC($m2$), line 7) requires one of these two conditions be true: $ST_{m2}[i,l] \leq DEL_l[i] \vee ST_{m2}[i,l] < current\_time - \Delta$.

- if $ST_{m2}[i,l] \leq DEL_l[i]$ then:

  from $ST_{m2}[i,l] \geq send\_time_{m1}$ (lemma L2), we conclude: $send\_time_{m1} \leq DEL_l[i]$.

  The last message sent by P_i and delivered to P_l was sent at $DEL_l[i]$ (see definition of $DEL_l[i]$ and line 11). Due to lemma L3, all messages sent by P_i to P_l before $DEL_l[i]$ have been either delivered (in fifo order) or discarded. As $m1$ is delivered (by hypothesis) and has been sent at $send\_time_{m1} \leq DEL_l[i]$, it has already been delivered; this contradicts the hypothesis that $m1$ was delivered after $m2$.

- if $ST_{m2}[i,l] < current\_time - \Delta$ then:

  from $ST_{m2}[i,l] \geq send\_time_{m1}$ (lemma L2), we conclude: $send\_time_{m1} < current\_time - \Delta$.

  This means that if $m1$ has not yet arrived, when $m2$ is delivered, it will be discarded if it arrives (line 5) contradicting our hypothesis (namely, $m1$ is delivered). If $m1$ has arrived when $m2$ is delivered, its delivery condition (DC($m1$), line 7) is necessarily true (as $\forall x : ST_{m1}[x,l] \leq ST_{m2}[x,l]$, lemma L2); moreover it became true, at worst, at the same time as DC($m2$). So if DC($m1$) is satisfied before DC($m2$), $m1$ is delivered first; if it is satisfied at the same time it is also delivered first as $ST_{m1} < ST_{m2}$ (line 9).

□

6 Conclusion

In this paper a new abstraction for message deliveries, called Δ-causal order, has been presented. Such an abstraction extends the notion of causal order introduced by Birman and Joseph in [BJ87] to cope with unreliable communication networks with real-time delivery constraints. We suppose messages can be lost and have a *lifetime*, Δ, after which their data can no longer be used. Δ-*causal order* requires to deliver as much messages as possible within their lifetime in such a way that these deliveries respect causal order. We have also proposed a simple implementation of Δ-causal order. The protocol, based on a global clock, associates a delivery condition with each message m. Such a condition ensures that m is delivered *as soon as possible* i.e., when all messages that causally preced it either have been delivered or are lost or are still in transit when their deadlines expire.

Scheduling messages deliveries respecting Δ-causal order is particularly useful for *distributed multimedia real-time applications* where messages have a maximum transmission delay that must be respected to save the quality and the real-time interaction of their services and where messages can be lost as long as these losses do not cause a major degradation of their services. For such distributed applications, delivery of messages out of causal order or out of their lifetime deteriorate hardly the quality of the service. Consider as an example, an analogic video signal. The video is sampled at the source site and such samples produce a *continuous* flow of messages over the communication network. In order to reconstruct a high quality analogic signal at the destination site and to have a good interactivity:

- as much samples as possible must be delivered within the maximum transmission delay and
- samples arrived within the maximum transmission delay should be delivered in their sending order.

Such signals may however tolerate a loss of information as long as these losses do not cause a major degradation of analogic signals at the destination site [Wak93, HB94, Fer90]. For example, a video signal may tolerate the loss of at most 3 frames out of 30 frames that form 1 sec. of a video image [Fer90]; in other words:

- some samples can be lost.

Previous points indicate that multimedia real-time applications need a communication protocol that copes with unreliable networks and that is able to deliver as much messages as possible within the maximum transmission delay without violating their sending order. Δ-causal order is a tool solving such a problem in a simple way.

References

[AV94] S. Alagar and S. Venkatesan. An optimal algorithm for distributed snapshots with causal message ordering. *Inform. Process. Lett.*, 50:311–316, 1994.

[BJ87] K. Birman and T. Joseph. Reliable communication in the presence of failures. *ACM Trans. Comput. Syst.*, 5(1):47–76, 1987.

[BSS91] K. Birman, A. Schiper, and P. Stephenson. Lightweight causal order and atomic group multicast. *ACM Trans. Comput. Syst.*, 9(3):282–314, 1991.

[Fer90] D. Ferrari. Clients requirements for real-time communication services. *IEEE Communication Magazine*, 65–72, 1990.

[GM94] S. Gronemberg and D. Marwood. Real time groupware as a distributed system: concurrency control and its effect on the interface. In *Proceedings of ACM conference on CSCW*, pages 207–217, ACM press, 1994.

[GZ89] R. Gusella and S. Zatti. The accuracy of the clock synchronization achieved by TEMPO in Berkeley UNIX 4.3BSD. *IEEE Trans. Software Engineering*, 15(7):847–853, 1989.

[HB94] T. Houdoin and D. Bonjour. ATM and AAL layer issues concerning multimedia applications. *Annals of Telecommunications*, 49(5):230–240, 1994.

[JSS94] K. Jeffay, D.L. Stone, and F.D. Smith. Transport and display mechanisms for multimedia conferencing across packet switched networks. *Computer Networks*, 26:1281–1304, 1994.

[KO87] H. Kopetz and W. Ochsenreiter. Clock synchronization in distributed real-time systems. *IEEE Transactions on Computers*, (8):933–940, 1987.

[Lam78] L. Lamport. Time, clocks and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[Mil91] D. Mills. Internet time synchronization: the network time protocol. *IEEE Trans. on Communications*, 39(10):1482–1493, 1991.

[RB93] K. Ravindran and V. Bansal. Delay compensation protocols for synchronization of multimedia data streams. *IEEE Trans. on Knowledge and Data Engineering*, 5(4):574–589, 1993.

[RST91] M. Raynal, A. Schiper, and S. Toueg. The causal ordering abstraction and a simple way to implement it. *Inform. Process. Lett.*, 39:343–350, 1991.

[Ver94] P. Verissimo. Ordering and timeliness requirements of dependable real-time programs. *Real-time Systems*, 7:105–128, 1994.

[Wak93] I. Wakeman. Packetized video: options for interaction between the user, the network and the codec. *The Computer Journal*, 36(1):55–66, 1993.

[Yav92] R. Yavatkar. MCP: a protocol for coordination and temporal synchronization in multimedia collaborative applications. In *Proceedings of the 12-th IEEE Int. Conf. on Dist. Comp. Systems*, pages 606–613, IEEE press, 1992.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 46 avenue Félix Viallet, 38031 GRENOBLE Cedex 1
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399