A Mechanism to Synchronize Distributed Continuous Media in Unreliable Networks

Eduardo Lopez Dominguez, Luis A. Morales Rosales, Saul E. Pomares Hernandez

National Institute of Astrophysics, Optics and Electronics (INAOE), Luis Enrique Erro #1, 72840 Tonantzintla, Puebla, Mexico
{edominguez,lamorales,spomares}@inaoep.mx

Abstract. The synchronization of continuous media is a key issue in the development of distributed systems, such as videoconference and virtual reality. These services must be carried out in a synchronized manner with different media types, like continuous and discrete media. Due to the unreliable and asynchronous nature of networks, some problems can disrupt the synchronization. Such problems can include lost messages and delay jitter. In this paper, we aim to solve the synchronization problem of real-time continuous media in unreliable networks. The detection and recovery of lost messages is achieved by the technique of forward error detection in a distributed form, which implies avoiding retransmission. To the best of our knowledge, our work is the first to propose a forward error recovery technique for the synchronization of continuous media with causality control. The mechanism is based on our temporal model that determines logical dependencies between intervals (continuous media) to represent the basic temporal relations defined by Allen.

1 Introduction

The synchronization of continuous media (audio and video) is critical to guarantee the appropriate presentation in several applications. Such applications include, for instance: videoconferences, virtual reality, collaborative work, and parallel and distributed debugging. The problem of synchronizing continuous media relates to how one can ensure the correct temporal appearance of the media items (audio and video) [1]. Many works try to solve this problem; nonetheless, they are far from resolving it. The problem of synchronization is even more complicated when one considers real communication aspects in a distributed system, such as delayed and lost messages, which are two main problems found in unreliable networks.

In this paper, we aim to solve the problem related to the synchronization of real-time continuous media in unreliable networks. We refer by the term *real-time* to the simultaneous creation and transmission of media. This implies that neither a pre-processing step nor a previous storage exists.

We propose a dual-task mechanism to solve the synchronization problem. In the first task, we present a temporal synchronization model that describes how the synchronization is carried out. We work at two abstract levels. At the higher level, the temporal duration is taken into account by representing the continuous media segments as intervals. At the lower level, we work with intervals, considering that an interval is composed of a set of sequentially-ordered messages. We show that it suffices to ensure a partial causality between the endpoints to ensure a causal order at the interval level. The second aspect of the mechanism is focused on how to deal with the delay and loss of messages by proposing a fault-tolerant mechanism. Our proposal is based on the technique of forward detection and recovery of lost messages in a distributed form. By this, we avoid the retransmission of information. To the best of our knowledge, this work is the first to propose a *forward error correction* technique with causality control for the synchronization of continuous media. The forward recovery is achieved through the addition of redundancy. Two kinds of redundancy are used: redundancy on the type of message, and redundancy on the causal control information sent [2]. The last type of

redundancy is calculated based on the causal distance between events, and it is adaptable according to the conditions of loss in the network communication channels. Our proposal is an extension of the work developed in [3]. The extension is focused mainly on the fault tolerance mechanism.

The outline of this paper is as follows. We present in Section 2 the related works. Next, in section 3, the preliminaries and some required definitions are given. Following, the proposed Temporal Synchronization Model is presented in detail in Section 4. We introduce in Section 5 our Synchronization Mechanism. Conclusions are provided in Section 6.

2 Related Work

We classify the works that attempt to solve the synchronization problem between continuous media (streams) into two main categories: *real-time* and *on-demand*. Our work is focused on real-time synchronization. Real-time synchronization can be divided into continuous and discrete events. While continuous events concern a repetitive pattern of related events, discrete events synchronize activities in a distributed system by answering events. Our work tries to solve the problem of continuous synchronization, which in turn, can be divided into *intra-stream* and *inter-stream* synchronization. [1],[4 -6].

Intra-stream synchronization refers to the preservation of physical temporal dependencies within a single stream. Inter-stream synchronization, on the other hand, is focused on the preservation of physical and logical temporal dependencies between streams. The objective of this paper is to solve the problem of inter-stream synchronization.

One of the main mechanisms used to satisfy physical dependencies involves the use of a multiplexer to send a single synchronized stream [7,8]. These works generate a bottleneck and produce delays at the transmission and at the reception of streams. Another mechanism used is the normalization of the participants' physical clocks to achieve the synchronization [9,10].

On the other hand, some works use causal algorithms to achieve the synchronization between streams [2],[11-19]. We can divide these algorithms into two categories based on the type of communication channel used (See Figure 1). In the first category, we classify the causal algorithms that assume reliable communication channels (*i.e.*, lost message do not exist) [11-13]. In the second category, we consider causal algorithms that work in unreliable communication channels (*i.e.*, there are lost message during the transmission) [2],[14-19]. We have classified these last works into two categories based on the technique used to detect and recover lost messages.

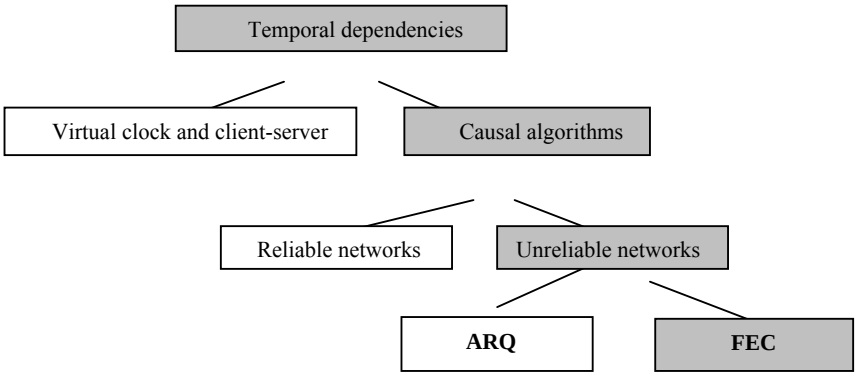

Fig. 1. Classification of multimedia synchronization based on the type of channel and recovery technique

Works in the first category use the *Automatic Repeat Request* (ARQ) technique or some variant of it [14], [16-19]. They detect a lost message in some way and recover it by retransmitting it. This mechanism is not recommended in real-time systems because it involves delays and a huge overhead. Others works, such as [15], consider the package lifetime to determine whether it is useful or not for the application. Nevertheless, these works do not address the case of lost messages.

The second category uses the *Forward Error Correction* (FEC) technique to detect and recover lost information. Our work is classified in this category. It is an algorithm that uses the causal relation at an interval level to achieve the synchronization between streams. To the best of our knowledge, our work is the first to propose a forward error correction technique to synchronize continuous media with causality control. The forward recovery is achieved by introducing redundancy. For a detailed description of our mechanism, refer to Section 5.

3 Preliminaries

3.1 The System Model

Processes: The application under consideration is composed of a set of processes $P = \{i, j \ldots\}$, organized into a group that communicates by unreliable broadcast asynchronous message passing.

Messages: We consider a finite set of messages M, where each message $m \in M$ is identified by a tuple $m=(p,x)$, where $p \in P$ is the sender of m, denoted by $Src(m)$, and x is the local logical clock for messages sent by p, when m is broadcasted. The set of destinations of a message m is always P.

Events: Let m be a message. We denote by $send(m)$ the emission event of m by $Src(m)$, and by $delivery(p,m)$ the delivery event of m to participant $p \in P$. The set of events associated to M is then the set $E = \{send(m) : m \in M\} \cup \{delivery(p,m) : m \in M \wedge p \in P\}$.

Intervals: We consider a finite set I of intervals, where each interval $A \in I$ is a set of messages $A \subseteq M$ sent by a participant $p=Part(A)$, defined by the mapping $Part: I \rightarrow P$. Formally, we have $m \in A \Rightarrow Src(m)=Part(A)$. Due to the sequential order of Part(A), we have for all $m,m' \in A$, $m \rightarrow m'$ or $m' \rightarrow m$. We denote by a^- and a^+ the unique messages of A, such that for all $m \in A$, we have $a^- \neq m$ and $a^+ \neq m \Rightarrow a^- \rightarrow m \rightarrow a^+$. The a^- and a^+ messages are the endpoints of A. We assume in this paper that $a^- \neq a^+$.

3.2 Background and Definitions

In this section we present some definitions used in the synchronization mechanism.

3.2.1 The Happened-Before Relation on Intervals

We identify and define two possible precedence relations at an interval level. These two relations are based on the possible happened-before relation for single events. The two relations identified for intervals are the causal relation and

the simultaneous relation. For more details of these definitions, refer to [3]. We begin by giving the definition of the causal relation to be applied to intervals.

Definition 1. The relation " $\rightarrow_I$ " is accomplished if the following two conditions are satisfied:

1. $A \rightarrow_I B$ if $a^+ \rightarrow_{M'} b^-$

2. $A \rightarrow_I B$ if $\exists C \mid (a^+ \rightarrow_{M'} c^- \wedge c^+ \rightarrow_{M'} b^-)$

where a^+ and b^- are the final and initial send events (or messages) of A and B respectively, c^- and c^+ are the endpoints of C, and $\rightarrow_{M'}$ is the partial causal order induced on $M' \subseteq M$, where M', in our case, is the subset composed by the endpoint messages of the intervals in I.

Definition 2. Two intervals A, B are said to be simultaneous " ||| " if the following condition is satisfied:

$$A \,|||\, B \Rightarrow a^- \,||\, b^- \wedge a^+ \,||\, b^+$$

In our work we consider Definition 2 as the complement relation to the causal relation at an interval level (Definition 1). This means that an interval can only either precede or be simultaneous to another interval at a given time.

Definition 3. Causal Broadcast Delivery for Intervals based on the endpoints

If $(a^+, b^-) \in A \times B$, $send(a^+) \rightarrow send(b^-) \Rightarrow \forall p \in P$, $delivery(p, a^+) \rightarrow delivery(p, b^-)$ then

$$\forall p \in P \Rightarrow delivery\ (p,A) \rightarrow_I delivery\ (p,B)$$

3.2.2 Causal Distance

The causal distance defines the greatest number of causal messages in a linearization between a pair of messages. The definition of causal distance was introduced in [2]. Formally, the causal distance is defined as follows:

Definition 4. Let m and m' be messages. The distance $dist(m,m')$ is defined for any pair m and $m' \in M$ such that $m \rightarrow m'$: $dist(m,m')$ is the greatest integer n such that for some sequence of messages (m_i, $i = 0...n$) with $m = m_0$ and $m' = m_n$, we have $m_i \downarrow m_{i+1}$ for all $i = 0 \ldots n-1$, where $\downarrow$ is the *immediate dependency relation* defined in [11].

4. Temporal Synchronization Model

In order to achieve the synchronization between continuous media in distributed systems, we propose a model to determine temporal relations based only on the identification of logical precedence dependencies. The model translates a temporal scenario to be expressed in terms of the precedence relations at an interval level (defined in the section 3.2.1); we call this translation *logical mapping*. In our work, a logical mapping decomposes a temporal scenario into data segments (intervals) that are arranged according to their possible precedence dependency.

4.1 Logical Mapping

The process to create logical mappings involves taking every pair of intervals in the system that compose a temporal scenario, and translating each pair into four data segments, which are determined according to the possible precedence dependency of the discrete events that compose them. These data segments, according to our definition, become new

intervals.The resulting intervals are expressed only in terms of the happened-before relation and the simultaneous relation.

In order to consider the seven basic relations and their inverses and to maintain the model simplified, we first identify the *X* and *Y* intervals for each pair of intervals in the system. The *X* interval will be the interval with the first left endpoint, and the *Y* interval will be the remaining interval. This is done in order to ensure that for every pair, $x^- \rightarrow y^-$ or $x^- \parallel y^-$ at all times. Once the *X* and *Y* intervals are identified, the model segments each pair into four subintervals (*A*, *B*, *C* and *D*) (See Table 1). When the subintervals are already identified, we proceed to construct the general causal structure $S=A \rightarrow_I W \rightarrow_I B$, where *W* determines if overlaps exist between the present pair.

Table 1. The Process of Logical Mapping[1]

$\forall(X, Y)$	$\in$	$I \times I$	
$A(X,Y)$	$\leftarrow$	$-\{ x \in X : x \rightarrow y^- \}$ $-\ \emptyset$	for $x^- \rightarrow y^- \ \vee$ for other cases
$B(X,Y)$	$\leftarrow$	$-\{ x \in X : y^+ \rightarrow x \}$ $-\{ y \in Y : x^+ \rightarrow y \}$ $-\ \emptyset$	for $y^+ \rightarrow x^+ \ \vee$ for $x^+ \rightarrow y^+ \ \vee$ for other cases
$C(X,Y)$	$\leftarrow$	$-X-(A(X,Y) \bigcup B(X,Y))$	
$D(X,Y)$	$\leftarrow$	$-Y-B(X,Y)$	
$W(X,Y)$	$\leftarrow$	$C \parallel\parallel D$	
$S(X,Y)$	$\leftarrow$	$A \rightarrow_I W \rightarrow_I B$	

According to Table 2, we are now able to express every possible temporal relation based only on the interval happened-before relation and the interval simultaneous relation. We remark that this capacity is the core of our synchronization model.

Table 2. Allen's relations and their logical mapping.

Allen's Relations	**Initial Endpoints**	**Interval Temporal Relation**	**Logical Mapping**
X *before* Y	$x^+ \rightarrow y^-$	xxxxx yyyy	$A \rightarrow_I B$
Y *after* X			
X *meets* Y	$x^+ \parallel y^-$	xxxx yyyy	$A \rightarrow_I (C \parallel\parallel D) \rightarrow_I B$
Y *meet-by* X			
X *overlaps* Y	$x^- \rightarrow y^- \rightarrow x^+, x^+ \rightarrow y^+$	xxxxxxxx yyyyyyyyy	
Y *overlap-by* X			
X *includes* Y	$x^- \rightarrow y^-, y^+ \rightarrow x^+$	yyyy xxxxxxxxx	
Y *during* X			

[1] We consider in our model that an interval must be empty, for this case we introduce the next properties:

- $\emptyset \rightarrow_I A \vee A \rightarrow_I \emptyset = A$
- $A \rightarrow_I \emptyset \rightarrow_I B = A \rightarrow_I B$
- $A \parallel\parallel \emptyset \vee \emptyset \parallel\parallel A = A$

X *starts* Y Y *started by* X	$x^- \parallel y^-$, $x^+ \rightarrow y^+$	xxxxx yyyyyyyyyyy	$(C \mid\mid\mid D) \rightarrow_I B$
X *finishes* Y Y *finished-by* X	$x^+ \parallel y^+$, $x^- \rightarrow y^-$	yyyyy xxxxxxxxx	$A \rightarrow_I (C \mid\mid\mid D)$
X *equals* Y	$x^- \parallel y^-$, $x^+ \parallel y^+$	xxxxxxxx yyyyyyyy	$C \mid\mid\mid D$

5 The Synchronization Mechanism

The present mechanism is based on the synchronization temporal model, presented in Section 4 (See Table 1). The mechanism carries out the creation of logical mappings and ensures their reproduction even when messages are lost. The correctness of the synchronization temporal model was proved in [3]. In this section, the extension is presented to provide a fault tolerant mechanism in the presence of lost messages. Internally, the synchronization mechanism uses two kinds of messages: causal messages and FIFO messages. The causal messages are divided into: *begin*, *cut* and *end* messages. The *begin* and *end* messages are the left and right endpoints of the original intervals, and *cut* is a control message used by the algorithm to inform about an interval segmentation. FIFO messages (*fifo_p*) are only used inside an interval. We note that a causal message also locally satisfies the FIFO order. We provide a description of its operation below.

5.1 The Recovery Approach

The causal messages are used either to achieve the segmentation proposed in the temporal model or to synchronize the intervals involved. The loss of causal messages disturbs the segmentation between intervals, and thus, deteriorates the synchronization. To tolerate the loss of some *begin*, *end* or *cut* causal message, we introduce a certain *redundancy*. In the algorithm, we have two kinds of redundancy: *redundancy on the type of message* and *redundancy on the causal control information*. The first type of redundancy is applied to the *begin* message and to the *cut* message. In this redundancy, some immediately consecutive *fifo_p* messages of a *begin* or *cut* message are sent as copies of the messages in question. If a *begin* or *cut* message is lost, the first *fifo_p* message is taken as the lost causal message. The *fifo_p* messages sent as copies can contain updated causal control information. The number of consecutive FIFO messages sent as copies of a causal message is not established by the mechanism, although a study carried out in [10] shows that the probability that a message can be lost diminishes considerably from five or more consecutive messages lost.

The second type of redundancy, which involves the causal control information, is based on the causal distance. The causal distance defines the greatest number of causal messages in a linearization between a pair of messages (see Definition 4). For example, for messages that have an immediate dependency, the causal distance is equal to one. For more details, refer to [2]. By considering a larger distance (more than one), we increase the redundancy in the control information sent in the system. The main advantage is that this increases the tolerance degree of lost message.

The detection and recovery of lost messages is as follows. When a *send* event occurs of a *begin, end* or *cut* causal message, the information added to its control information corresponds to the causal message identifiers that have an equal or smaller causal distance with the *send* event involved. With this redundant information, it is possible to detect and recover the control information of the lost messages in a forward way (Figure 3). We note that the redundant con-

trol information about a causal message is added only if the causal distance established is equal or smaller to the number of times that the information about this message has circulated in the system. In this way, our mechanism can adjust the redundant information needed to be sent.

5.1.1 Synchronization Mechanism when a *Begin* Message is Lost

In order to explain how the redundancy in the type of message is used to achieve the segmentation when a *begin* message is lost, we present the following faulty scenario (Fig. 2). We recall that the process of creating logical mappings in our work is made online by identifying the causal boundaries of the concerned segment(s) from left to right. In this example, segment *A* must first be determined. To achieve this, we identify the left causal boundary a^- as equal to $x^- = x_1$, and the right causal boundary as equal to $a^+ = x_k$. The right endpoint a^+ is determined by the last *fifo_p* message received by participant *j* (lines 133-136, Table 3) before the *begin* send event ($send(y^-)$). Once we know the causal boundaries of *A,* we can determine the set of messages that compose it ($A = \{x_1, x_{1+1},\ldots,x_k\}$).

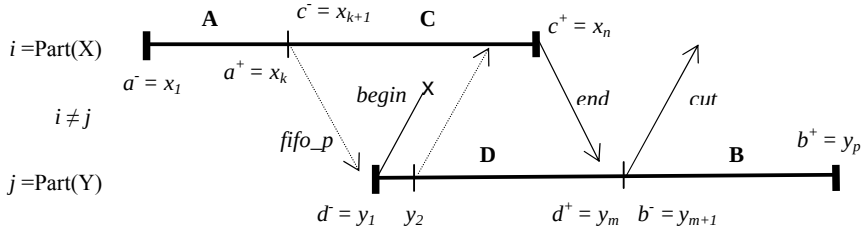

Fig. 2. Forward recovery at process i ($A \rightarrow_I (C \,|||\, D) \rightarrow_I B$).

After interval *A* is identified, we proceed to recognize the causal boundaries of *C* and *D*. At this point, we can identify the left causal boundaries $c^- = x_{k+1}$ and $d^- = y_1$, but it is only until the *send* and *delivery* event of the right endpoint x^+ that we can identify the right endpoints of *C* and *D*. In this scenario, the causal message y_1 is lost during its transmission to process *i*. The forward detection and recovery of y_1 is through the FIFO message y_2 at process *i*, which is an updated copy of y_1 (lines 35-50). Process *i* detects the loss of y_1 upon the reception of y_2 and takes y_2 as the causal *begin* message of interval *Y*. At this point, FIFO message y_2 is considered as a causal message (lines 72-140). Now with the $send(x^+ = x_n)$ we establish that $c^+ = x_n$, and consequently, $C = \{x_{k+1}, x_{k+2},\ldots,x_n\}$. At the reception of x^+ by participant *j,* our algorithm sends a *cut* message (lines 129-131), which establishes the end of interval *D* ($d^+ = y_m$) and the beginning of interval *B* ($cut = b^- = y_{m+1}$). As result, we have $D = \{y_1, y_2,\ldots, y_m\}$). Finally, with the send event of y^+ (lines 11-71), we have $b^+ = y_p$, and consequently, $B = \{y_{m+1}, y_{m+2},.., y_p\}$.

5.1.2 Synchronization Mechanism when an *End* or *Cut* Message is Lost

In order to explain how the redundancy based on causal distance is used to achieve the synchronization in the presence of lost *end* or *cut* messages, we present the following faulty scenario (figure 3). In this example, we consider segment *A,* and the left causal boundaries of *C* and *D* ($c^- = x_{k+1}$ and $d^- = y_1$, respectively) are identified. Only by the *send* and *delivery* event of the right endpoint x^+ , we can identify the right endpoints of *C* and *D*. With the $send(x^+ = x_n)$ (lines 11-71) we establish that $c^+ = x_n$, and consequently, $C = \{x_{k+1}, x_{k+2},\ldots,x_n\}$. The causal message x^+ is lost during its transmission to process *j*; therefore, we cannot identify the right endpoint of *D.* At the reception of x^+ by participant *l,* our algorithm sends a *cut* message which carries attached control information about messages x^+ and y^- (lines 18-21) with causal distances $dist=1$ and $dist=2$, respectively.

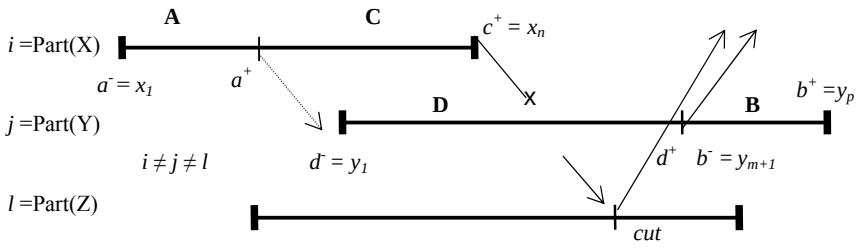

Fig. 3. Forward recovery at process *j* with d=2 ($A \rightarrow_I (C \,|||\, D) \rightarrow_I B$).

By using the control information attached to the *cut* message, process *j* is able to detect that message x^+ has been lost (lines 72-91). Because the lost message x^+ is an endpoint, *j* proceeds to send a cut message which establishes the end of interval *D* ($d^+ = y_m$) and the beginning of interval *B* ($cut = b^- = y_{m+1}$) (lines 89-91 and 125-128). As a result, we have $D=\{ y_1, y_2, \ldots, y_m\}$). Finally, with the send event of y^+, we have $b^+ = y_p$, and consequently, $B=\{ y_{m+1}, y_{m+2}, \ldots, y_p\}$. The segmentation for intervals *X* and *Y* is achieved: $A \rightarrow_I (C \,|||\, D) \rightarrow_I B$.

5.2 The Algorithm

5.2.1 Data Structures

The main data structures used in the algorithm are:

VT(*p*) is the vector time. For each process *p* there is an element *VT*(*p*)[*j*] where *j* is a process identifier. The size of *VT* is equal to the number of processes in the group. *VT*(*p*) contains the local view that process *p* has of the elements of the system. In particular element *VT*(*p*)[*j*] represents the greatest element number of the identifier *j* and 'seen' in causal order by *p*. It is through the *VT*(*p*) structure that we are able to guarantee the causal delivery of elements.

The structure of the control information *CI*(*p*) is a set of entries (*k, t, d*). Each entry in *CI*(*p*) denotes a message that is not ensured by participant *p* of being delivered in a causal order. The entry (*k, t, d*) represents a diffusion by participant *k* at a logical local timeclock $t = VT(p)[k]$, and *d* is the potential the causal distance.

The algorithm uses causal messages and FIFO messages which compose the intervals; a message *m*, in general, is composed of an identifier (*k, t, f*) and an attached causal information *H*(*m*). The intervals are identified by the tuple (*k, t*). For FIFO messages the structure *H*(*m*) is always $H(m)=\emptyset$. Formally, a message *m* is a tuple *m*=(*k,t,f,H*(*m*)), where:

- *k* is the identifier of the sender *k*=*Src*(*m*).
- *t* =*VT*(*p*)[*k*] is the (local) clock value of *p* for the identifier *k* when a causal message *m* (*begin*, *end*, or *cut*) or FIFO is sent.
- *H*(*m*) is a set of tuples (*k, t*) that represent intervals.

The structure *H*(*m*) contains identifiers of intervals causally preceding the causal message *m* (*begin*, *end*, or *cut*), which denotes the *begin* and/or *end* of other intervals. The information in *H*(*m*) is needed for the causal delivery of the causal message *m*. The causal delivery of *m* ensures the causal delivery of the interval to which *m* belongs. The structure *H*(*m*) is built before a causal message is broadcasted, and then it is attached to the causal message.

5.3 Algorithm specification

Next, we present in Table 1 the algorithm to synchronize real- time continuous media. The algorithm can manipulate *N* sources, and each source can send *M* streams.

Table 3. Synchronization mechanism tolerant faults

1. **Initially**
2. *VT*(*p*)[*j*] ←∅ ∀ *j*: 1…n
3. *VTI*(*p*)[*j*] ←∅ ∀ *j*: 1…n
4. *VTC*(*p*)[*j*] ←∅ ∀ *j*: 1…n
5. *CI*(*p*)←∅
6. *Act*=0
7. *last_fifo*(*p*)←∅
8. *num_cc* = *num_cop*
9. *con*=0
10. *causal=0*

For each *m* message diffused by *p* with process identifier *i*

11. ***Send*** (*Input*: *TP* = *begin* | *end* | *cut* | *fifo_p*))
12. *VT*(*p*)[*i*] ←*VT*(*p*)[*i*] + 1
13. If (*TP* = *fifo_p*) then
14. *Con* = *con*+1
15. Endif
16. If not (*TP* = *fifo_p* and *con* > *num_cop*) Then
17. If not (*TP* = *begin* or *TP* = *fifo_p*) Then
18. For each (*s*,*t*,*d*) ∈ *CI*(*p*) /*Construction of the *H*(*m*) for *end* and *cut* message*/
19. (*s*,*t*,*d*) ← (*s*,*t*,*d*+1)
20. *H*(*m*) ← *H*(*m*) ∪ (*s*, *t*)
21. Endfor
22. If (*TP* = *cut*) then
23. n*um_cc* = 0
24. Endif
25. Else
26. If not (*TP* = *fifo_p*) then /*Construction of the *H*(*m*) begin message*/
27. *CI*(*p*) ← *CI*(*p*) ∪ *last_fifo*(*p*) /*Adding *fifo_p* messages to CI(p)*/
28. For each (*s*,*t*,*d*) ∈ *CI*(*p*)
29. (*s*,*t*,*d*) ← (*s*,*t*,*d*+1)
30. *H*(*m*) ← *H*(*m*) ∪ (*s*, *t*)
31. Endfor
32. *reg*(*p*) ← *last_fifo*(*p*)
33. l*ast_fifo*(*p*) ← ∅
34. Else
35. c*op_CI*(*p*)←*CI*(*p*) /*construction of copies of *begin* message*/
36. If not (*last_fifo*(*p*) = ∅) then
37. ∀(*x*, *l*) ∈ *reg*(*p*)
38. If ∃ (*s*, *t*, *d*) ∈ *cop_CI*(*p*) | *x* = *s* and *l* = *t* then
39. *cop_CI*(*p*) ← *cop_CI*(*p*) / (*x*, *l*)
40. Endif
41. *cop_CI*(*p*) = *cop_CI*(*p*) ∪ *last_fifo*(*p*)
42. For each (*s*, *t*, *d*) ∈ *cop_CI*(*p*)

```
43.                         H(m) ← H(m) ∪ (s, t )
44.                       Endfor
45.                     Esle
46.                       For each (s,t,d) ∈ CI(p)
47.                           H(m) ← H(m) ∪ (s, t )
48.                       Endfor
49.                   Endif
50.               Endif
51.           Endif
52.    Else                        /*construction of copies of cut message*/
53.        If ( num_cc  < num_cop )  then
54.            For each (s,t,d) ∈ CI(p)
55.              H(m) ← H(m) ∪ (s, t )
56.            Endfor
57.            num_cc = num_cc + 1
58.          Else
59.            H(m)← ∅        /* H(m) for FIFO  messages */
60.        Endif
61.    Endif
62.    If  not( TP = end )Then     /*Determine if process p is sending or not an interval*/
63.        Act = 1
64.    Else
65.        Act = 0
66.    Endif
67.    m = (i, t = VT(p)[i], TP , H(m),data)
68.    Sending(m)
69.    If  ∃ ( k, t, d) ∈ CI(p) | d = dist_def  then
70.        CI(p) ← CI(p) /  ( k, t, d)
71.    Endif
       For each message received by p with process identifier j
72.    Receive(m) in p with i ≠ j  and  m= (i, t = VT(p)[i], TP , H(m),data)
73.    If  t =  VT(p)[k] +1 Then      /*FIFO deliver condition*/
74.        If  not ( TP =  fifo_p )    Then
75.           If not ( t' ≤ VT(p)[l]) ∀(l, t' )∈ H(m)  )  Then    /* causal delivery condition*/
76.              If (t' > VT(p)[l]) ∀ ( l, t' ) ∈ H(m)  /*Detection of lost message*/
77.                  VT(p)[l] = t'      /*update of vectors*/
78.              Endif
79.              If  ( TP = cut ) then
80.                  CPC = 1
81.               Endif
82.           Endif
83.        Endif
84.        Deliver(m)
85.        VT(p)[k] = VT(p)[k] +1
86.        If ( TP = begin ) then
87.           VTI(p)[k] =  1
88.        Endif
89.        If ( TP = cut ) then
90.            VTC(p)[k] =  1
91.        Endif
```

92. Else /*Detection of lost *begin* message */
93. If not ((*TP*=*fifo_p* and *VTI*(*p*)[*k*]!= 0) or (*TP* = *fifo_p* and *VTC*(*p*)[*k*] != 0))Then
94. If not ($t' \le VT(p)[l]$) $\forall (l, t') \in H(m)$) Then
95. If ($t' > VT(p)[l]$) $\forall$ $(l, t') \in H(m)$
96. $VT(p)[l] = t'$
97. Endif
98. Endif
99. *Causal* = 1
100. If ($VTI(p)[k] = 0$) then
101. $VTI(p)[k] = 1$
102. Else
103. $VTC(p)[k] = 1$
104. Endif
105. Endif
106. Deliver(*m*)
107. $VT(p)[k] = t$
108. Endif
109. If not (*TP* = *fifo_p* and *causal* = 0) Then
110. $CI(m) \leftarrow CI(m) \cup \{ (k, t, d=0) \}$
111. $\forall (l, t') \in H(m)$ /*Updating *CI*(*p*)with a most recent message*/
112. If $\exists (s, t, d) \in CI(m) \mid l = s$ and $t = t'$ then
113. $(s, t, d) \leftarrow (s, t, d+1)$
114. Endif
115. If $\exists (m, t, d) \in last\_fifo(p) \mid l = m$ and $t = t'$ then
116. $(m, t, d) \leftarrow (m, t, d+1)$
117. Endif
118. If $\exists (k, t, d) \in CI(p) \mid d = dist\_def$ then
119. $CI(p) \leftarrow CI(p) / (k, t, d)$
120. Endif
121. If (*TP* = *end*) then
122. $VTI(p)[k] = 0$
123. $VTC(p)[k] = 0$
124. Endif
125. If (*CPC* = 1) then
126. *Send* (*cut*) /*Sending a *cut* message by the lost message end*/
127. *CPC*=0
128. Endif
129. If *Act* = 1 and (not(*TP* = *cut*) and not (TP=*begin*)) Then
130. *Send*(*cut*) /*Sending a *cut* message by the deliver of a *end* message*/
131. Endif
132. Else
133. If $\exists$ $(x,l) \in last\_fifo(p) \mid x = k$ then
134. $last\_fifo(p) \leftarrow last\_fifo(p) / (x,l)$
135. Endif
136. $last\_fifo(p) \leftarrow last\_fifo(p) \cup (k, t, d=0)$
137. Endif /*Updating *last_fifo*(*p*) with a most recent message*/
138. *Causal* = 0
139. If $\exists (m, t, d) \in last\_fifo(p) \mid d = dist\_def$ then
140. $last\_fifo(p) \leftarrow last\_fifo(p) / (m, t, d)$
141. Endif

6 Conclusions

We have proposed a mechanism based on the technique of forward error correction to synchronize continuous media (audio and video) in unreliable networks. The mechanism avoids the retransmission of lost information. With this, we have introduced a feasible technique to synchronize real-time continuous media in the presence of lost messages. To the best of our knowledge, our work is the first to propose a forward error correction technique for continuous media synchronization with causality control. In future works, we will incorporate time restrictions to consider the lifetime of messages.

References

[1] Thomas Wahl, Kurt Rothermel: Representing Time in Multimedia Systems. Proceedings of the International Conference on Multimedia Computing and Systems, Boston, Massachusetts (1994), 538-543.

[2] Eduardo Lopez , Jorge Estudillo, Jean Fanchon, Saul E. Pomares: A Fault-tolerant Causal Broadcast Algorithm to be Applied to Unreliable Networks, The 17th IASTED International Conference on Parallel and Distributed Computing and Systems, Arizona (2005).

[3] Luis Morales, A. Algoritmo de Sincronización de flujos continuos en tiempo real. Masters Thesis in Computer Science, INAOE. Num. XM1086. Classification: XMM-M67-2005-XM1086, Tonantzintla Puebla, México. Year (2005).

[4] Haining Liu, Magda El Zarki: A synchronitation Control Scheme for Real- Time Streaming Multimedia Applications. Proc of Packet Video 2003, France (2003).

[5]Agustín J. Gonzalez and Hussein Abdel-Wahab: Light-Weight Stream Synchronization Framework for Multimedia Collaborative Applications, The Fifth IEEE Symposium on Computers and Communications (ISCC'2000), Antibes-Juan LesPins,France(2000),398-403.

[6]Apurv Gupta: Synchronization in Multimedia Applications. Technical Report on Networking Issues, Indian Institute of Technology Kanpur, Department of Computer Science & Engineering, (2000).

[7] Venkat Rangan, Shrihari Sampath, Sreerang Rajan.: Continuity and Synchronization in MPEG. IEEE Journal on Selected Areas in Communications, 14(1), (1996), 52-60.

[8] Pantelis Balaouras, Ioannis Stavrakakis, Lazaros F. Merakos.: Potential and Limitations of a Teleteaching Environment based on H.323 Audio-visual Communication Systems. Computer Networks, 34(6), (2000), 945-958.

[9] Nipun Agarwal, Sang Hyuk :A Model for Specification and Synchronization of Data for Distributed Multimedia Applications. Multimedia Tools and Applications, 3(2), (1996),79-104.

[10] Colin Perkins: RTP Audio and Video for Internet, Addison Wesley,USA(2003),pages 412.

[11] Saul Pomares, Jean Fanchon, Khalil Drira : The Immediate Dependency Relation: An Optimal Way to Ensure Causal Group Communication. Annual Review of Scalable Computing, Editions World Scientific, Series on Scalable Computing, vol 6, (2004), 61-79.

[12] Takayuki Tachikawa and MakotoTakizawa: Group Communication for Real-time Continuous Media. Proceeding of International Symposium on Multimedia Systems, Japan (1996), 118-125.

[13] Frank. Adelstein, Mukesh. Singhal: Real-Time Causal Message Ordering in Multimedia Systems, 15th IEEE International Conference on Distributed Computing Systems (ICDCS), (1995).

[14]Takayuki Tachikawa and Makoto Takizawa: Δ-Causality in Wide Area Group Communication. Proc. 1997 Int'l Conf. on Parallel and Distributed Systems , (1997),260-267.

[15]Roberto Baldoni, Achour Mostefaoui and Michel Raynal, Causal Deliveries in Unreliable Networks With Real-Time Delivery Constraints, Journal of Real-Time Systems,2(1),(1996), 308-321.

[16]Cezar Plesca, Romulus Grigoras,Philippe Quéinnec, A Flexible Communication Toolkit for Synchronous Groupware, International Conference on Sensor Networks (SENET), Montreal, Canada (2005), 6 pp.

[17] Roberto Baldoni, Roy Friedman, Robbert van: The Hierarchical Daisy Architecture for Causal Delivery, In Proceedings of the 17 th IEEE International Conference on Distributed Systems, 570-577, (1997).

[18] Akihito Nakamura and Makoto Takizawa : Causally Ordering Broadcast Protocol, In Proc. International Conference on Distributed Computing System, Australia (1994),48-55.

[19] Abdelhafid Abouaissa, Abderrahim Benslimane, M. Naimi: A group Communication Model for Distributed Real-time Causal Delivery, Seventh International Conference on Computer Communications and Networks (ICCCN '98),(1998), pp. 918.