Multimed Tools Appl (2008) 40:285–319
DOI 10.1007/s11042-008-0208-1

An RTP/RTCP based approach for multimedia group and inter-stream synchronization

Fernando Boronat Seguí · Juan Carlos Guerri Cebollada · Jaime Lloret Mauri

Published online: 25 June 2008

Abstract Most multimedia group and inter-stream synchronization techniques define or use proprietary protocols with new control messages. Many multimedia applications have been developed using RTP/RTCP as the standard for transmission of multimedia streams over IP networks. Instead of defining a new protocol, we propose the use of RTP/RTCP to provide synchronization. We take advantage of the feedback capabilities provided by RTCP and the ability to extend the protocol by extending and creating RTCP messages containing synchronization information. We have implemented our proposal and tested it in our University WAN. Our experiments have shown that network load resulting from synchronization is minimized and that asynchronies are within acceptable limits for multimedia applications.

Keywords Multimedia · Group synchronization · Inter-stream synchronization · RTP/RTCP

1 Introduction, definitions and related work

In this paper, we present a novel way to synchronize a group of receivers distributed in the network. Our approach, which we have called *RTP-based Feedback Global Synchronisation Approach (RFGSA)*, uses standard multimedia protocols (RTP/RTCP [27]) and clock synchronization protocols (for example, NTP [20] or GPS). Unlike other researchers, we do not define a new protocol with specific control messages that increase the load on the network. Instead, we use the most common protocol for multimedia delivery, RTP, and take

F. Boronat Seguí (✉) · J. Lloret Mauri
Escuela Politécnica Superior de Gandia, Universidad Politécnica de Valencia,
Ctra. Nazaret-Oliva S/N, 46730 Grao de Gandia, Valencia, Spain
e-mail: fboronat@dcom.upv.es

J. Lloret Mauri
e-mail: jlloret@dcom.upv.es

J. C. Guerri Cebollada
Escuela Técnica Superior de Ingenieros de Telecomunicación, Universidad Politécnica de Valencia,
Camino de Vera, S/N, 46022, Valencia, Spain
e-mail: jcguerri@dcom.upv.es

advantage of the feedback capabilities of RTCP to achieve synchronization. Our approach also uses a global time reference, for example, provided by NTP or GPS. The advantages of NTP include its availability (included in most current operating systems), its cost (free) and its precision (milliseconds). On the other hand, although GPS is more precise (nano-seconds), it requires additional hardware and is more expensive to use.

We have implemented and tested our proposal both objectively and subjectively, using both our University WAN and our laboratory LAN.

Network quiz shows and multimedia content distribution applications are examples of applications that require multimedia group synchronization. In a network quiz show is very important that each receiver receives information (for example, a multimedia question transmitted in several multimedia streams) at the same time and, as a result, has a fair chance of answering.

1.1 Definitions

There are three types of multimedia synchronization: intra-stream, inter-stream and group synchronization.

1. *Intra-stream synchronization* refers to the temporal relationship between the media units (MUs) of one time-dependent media stream. In [17], there is a survey of algorithms providing this type of synchronization. We will assume that intra-stream synchronization is provided by buffering methods, such as the ones proposed in [17]. Our algorithm provides the remaining two types of synchronization at the receivers' application layer.
2. *Inter-stream Synchronization* refers to the synchronization between media streams. In remote user speech, the synchronization between the user's audible words and the associated movement of the lips, referred to as *Lip synchronization* or *lip-sync* [15], is an example of this type of synchronisation.
3. *Group synchronization* refers to the synchronization of the playout of one or more media streams by several receivers at the same time. The playout of a film and commentaries sent by a teacher to all the students in a distance teaching system is one example of this type of synchronization. Another example is a simple network quiz show in which the winning contestant is the first one to answer a presenter's test question correctly. Figure 1 illustrates group synchronization for a video sequence (only one stream) showing a bouncing ball. There are two synchronization requirements: *synchronization of the Initial Playout Time* (starting point) for all the receivers and *maintaining a controlled asynchrony between receivers* (within allowed bounds decided by the application user).

1.2 Related work

We have found numerous proposed algorithms for achieving intra and inter-stream synchronization in different scenarios [4, 8, 9, 15, 17, 22–26] but very few proposals for achieving group synchronization [1, 6, 10, 30]. Our work on a qualitative comparisson of both types of synchronization algorithms can be found in [3], including the proposal presented in this paper.

There are many solutions to the inter-stream synchronization problem but all of them use specific control messages or protocols to exchange information required for synchronization. To summarise, we point out two examples which our proposal draws from: the *Feedback Protocol* [23–26] and the *Feedback Global Protocol* [8, 9]. The former uses local

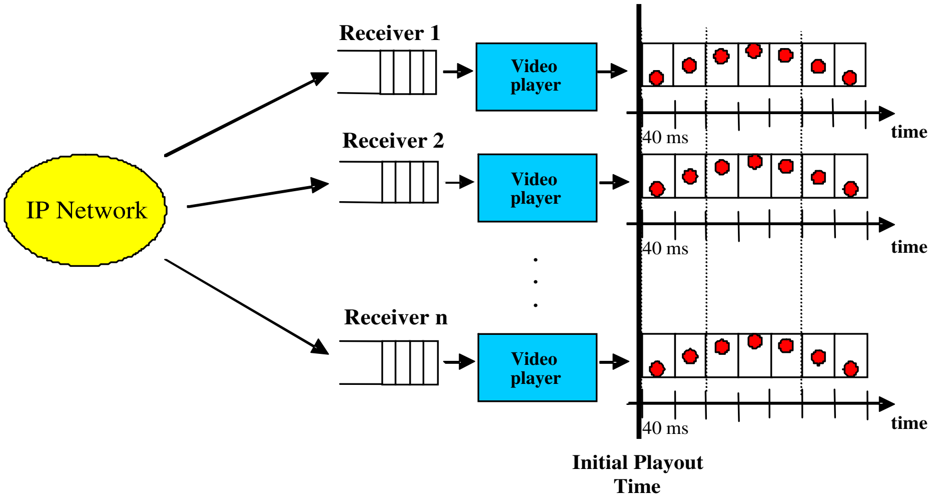


Fig. 1 Group synchronization

clocks whereas the latter uses a global time reference. Both are adaptive, valid for multicast and use a master/slave scheme and feedback techniques to exchange information between sources and receivers. They provide intra and inter-stream synchronization but not group synchronization. Both take into account the limits presented by Steinmetz in [28].

We have found very few solutions to the group synchronisation problem. All of them are receiver-based and instead of using standard protocols, they exchange specific control messages between sources and receivers, increasing the network load. There is only one solution [6] which uses RTP (and RTCP, but only for QoS and not for synchronization purposes). The solution proposed in [1] and the *VTR algorithm* [10] are worth emphasising. Both solutions use one receiver as the synchronization reference (*master/slave scheme*) and receivers synchronize with this reference by exchanging synchronization information. This results in a considerable additional network load. The algorithm proposed in [1] also synchronizes the initial playout time for all the receivers. Subsequently, Ishibashi et al. have used their solution [10] for group synchronization control with continuous and haptic media [11, 13].

Apart from these solutions for multimedia synchronization, we have found two RFCs related to our work. RFC 4585 [21] and RFC 4586 [5] define and evaluate new extensions for the Audio-Visual Profile (AVP) for RTCP-based feedback (RTP/AVPF). These extensions enable receivers to statistically provide more immediate feedback to senders and thus allow for short-term adaptation and efficient feedback-based repair mechanisms to be implemented. RFC 4585 also defines a small number of general purpose RTCP feedback messages. In our proposal we use well known RTP/RTCP capabilities: new extensions for RTCP RR packets [27] and for RTCP APP [27] packets for synchronization.

1.3 Organisation of the paper

Our group and inter-stream synchronization proposal is presented in the next section. Section 3 describes the implemented test bed and the objective and subjective evaluation of the proposal. The results obtained are also presented in Section 3. Finally, we draw conclusions in Section 4.

2 Synchronization approach

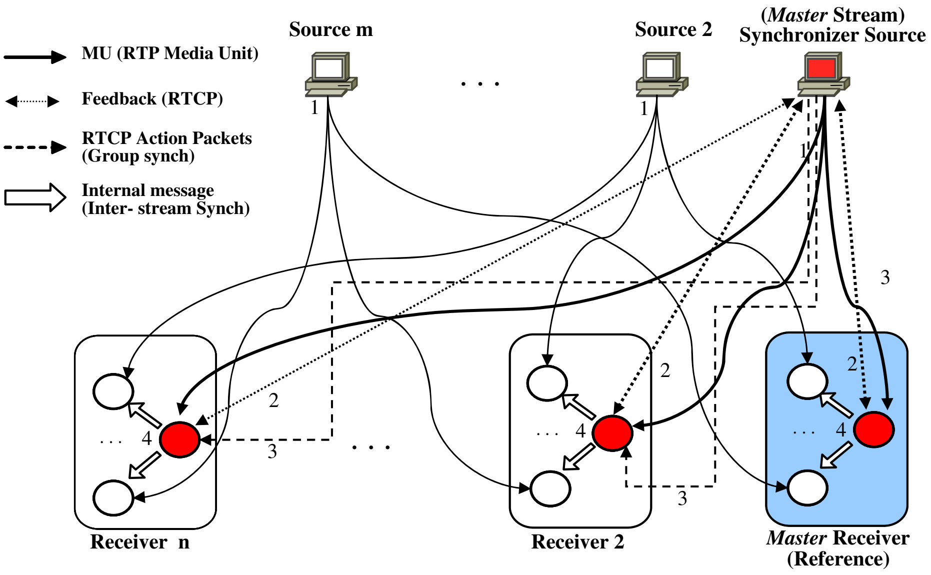
The main aims of our proposal are to:

- Guarantee the same initial playout time for all the receivers, observing the limits in [28]
- Use RTP/RTCP to implement the algorithm
- Use the most common techniques implemented by most of the analyzed synchronization algorithms, based on synchronization actions, with simple actions such as '*pause*' (stop playing) and '*skip*' (jump or move forward)
- Maintain a low control load for synchronization, in relation to the total load resulting from all of the data packets sent

We consider two master/slave schemes: one for group synchronization and another for inter-stream synchronization. Our proposal includes two separated processes (Fig. 2):

1. *Group Synchronization*. This involves the synchronization of the Initial Playout Time of the master stream by all of the receivers and, subsequently, the synchronization with the playout of the master receiver. This guarantees that all of the receivers play the master stream synchronously.
2. *Inter-stream Synchronization*: This involves the local synchronization of media streams (audio, video, text, etc.) by each receiver. We call this process *Local Inter-stream Synchronization*. Slave stream playout processes adapt their playout points to the playout process of the master stream.

Transmission of media units (MUs) can be multicast or unicast (step 1 in Fig. 2) using RTP, from one or more sources to one (requiring only inter-stream synchronization) or more



MU (RTP Media Unit)
Feedback (RTCP)
RTCP Action Packets (Group synch)
Internal message (Inter- stream Synch)
Source m
Source 2
(*Master* Stream) Synchronizer Source
. . .
1
2
3
4
Receiver n
Receiver 2
Master Receiver (Reference)

Fig. 2 Group and inter-stream synchronization

receivers. One of the streams is considered the *master stream* (thick lines) and its source will be considered the *Synchronizer Source*. Moreover, we select one receiver to be the *master receiver*, whose playout point of the master stream will become the reference for the group synchronization process. The *master receiver* can be selected statically or dynamically using several methods (for example, the slowest receiver, the fastest receiver, etc., as discussed in [2]). To make our proposal fault-tolerant, if the synchronizer source detects that the master receiver has failed or left the session, it can dynamically select a new master receiver.

Taking advantage of the feedback capabilities of RTCP (step 2), the algorithm, which runs at the Synchronizer Source, uses RTCP Receiver Report (RR) packets. These packets are extended to include the data required to determine the playout point of the master stream in each receiver. Using this data, the Synchronizer Source will be able to determine the playout point of the master stream at all of the receivers. If asynchrony exceeding a threshold value is detected, the Synchronizer Source, will send '*action messages*' (extended RTCP APP packets [27], step 3) to the slave receivers. The slave receivers adjust the master stream playout point by '*skipping*' or '*pausing*' MUs. In this way, all of the receivers will playout the master stream synchronously. Our solution is a *receiver driven solution* because the synchronization actions are performed by the receivers but in accordance with instructions sent by the source to achieve media synchronization.

When synchronization of the master stream is achieved, we also need to achieve local inter-stream synchronization at each receiver, for example, lip-sync between audio and video streams. To achieve this, we propose the use of an internal inter-process communication channel (for example, *mbus* [16] can be used). Locally and internally within each receiver, the playout process of the master stream (already synchronized between receivers) will periodically send information about its playout point to the other playout processes of the local slave streams (step 4). This allows the other playout processes to adjust their playout points by '*skipping*' or '*pausing*' MUs.

We now present in detail the main sub-processes of the global synchronization process.

2.1 Group synchronization (distributed)

When receiver playout rates are exactly equal and the MU network delay between source and receiver is deterministic, media synchronization between receivers can be guaranteed if the source firstly indicates to receivers exactly when playout should begin and then sends MUs at a constant rate or with timestamps indicating when they have to be played [26]. Receivers can loose synchronization, however, due to network jitter and deviations in playout rates.

We assume the source starts the transmission at time t_0. Network jitter can cause the receiver to start its playout process as early as $t_0+\mathrm{Del}_{\mathrm{min}}$, or as late as $t_0+\mathrm{Del}_{\mathrm{max}}$. ($\mathrm{Del}_{\mathrm{max}}$ and $\mathrm{Del}_{\mathrm{min}}$ are the maximum and minimum estimated network delay). Hence, we can find an initial asynchrony of at least $\mathrm{Del}_{\mathrm{max}}-\mathrm{Del}_{\mathrm{min}}$ between each pair of receivers. Moreover, a lack of synchronization will also result from deviations between playout rates of each receiver. Now, let us suppose the receivers have a nominal playout rate of θ MUs per second (equal to the source transmission rate) and that the deviation or *drift* in the receivers playout rate is limited by $\pm\rho$. According to these parameters, in the worst case, one receiver would be able to playout the stream at the maximum rate of $\theta * (1-\rho)$, whereas another receiver would be able to playout the stream at the minimum rate of $\theta * (1+\rho)$. The maximum possible asynchrony between the two receivers, playing the stream at the maximum and minimum rate respectively, will increase as playout progresses. The maximum possible asynchrony, A, in MUs, at the time the slowest receiver is playing the n-th MU, is given by Eq. 1.

$$A = \left\lceil \frac{(\mathrm{Del}_{\max} - \mathrm{Del}_{\min}) + 2 * \theta * \rho * n}{\theta * (1 + \rho)} \right\rceil \quad \text{Media Units} \tag{1}$$

The term $(\mathrm{Del}_{\max} - \mathrm{Del}_{\min})/(\theta * (1-\rho))$ is the contribution of network jitter to the asynchrony, whereas the factor $2 * \theta * \rho n/(\theta * (1-\rho))$ represents the effect of different playout rates in the receivers. A will increase linearly as playout progresses. This is unacceptable in practice and a method of correcting the asynchrony is required.

To achieve synchronization between a group of receivers involved in a multimedia application, two conditions must be satisfied: first, that all the receivers initiate the playout of the streams at the same time (*Initial Playout Time*), and, second, that all the receivers playout all the streams synchronously throughout the life of the application. A mechanism will be required to correct drifts in the playout processes of the receivers (which we refer to as *Fine Synchronization between Receivers*).

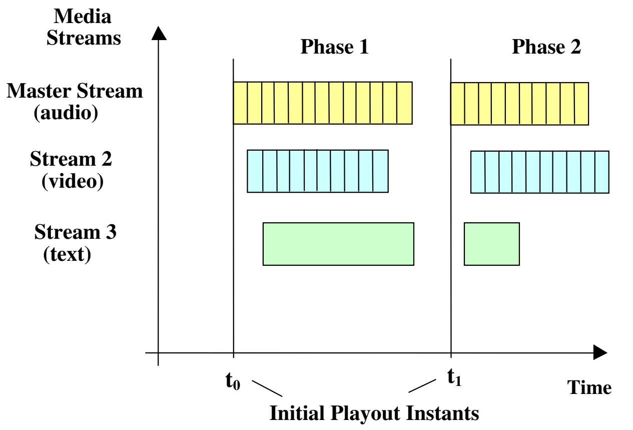
2.1.1 Initial playout instant and coarse synchronization

We begin by addressing the synchronization of the *Initial Playback Instant*. We achieve this by ensuring that all receivers agree on the Initial Playout Instant for the *master stream*. The Initial Playout Instant will be calculated by the source such that receiver buffer underflow and/or overflow is avoided and will be communicated to the receivers prior to the transmission of the first MU.

Apart from the Initial Playout Instant, a multimedia presentation can be divided into several phases (Fig. 3), each with a different *Initial Playout Instant*, which can be used to synchronize receivers. We call this process *Coarse Synchronization*. We assume that a phase ends when the transmission of the *master stream* is stopped for longer than a specified interval (that can be set by the application) and assume that a phase starts when the transmission of that stream starts again.

To solve both problems (*Initial Playout Instant* and *Coarse Synchronization*), in each phase we propose the use of a global time reference for all the members of the distributed system. Obviously, the precision of the synchronization will depend on the precision of the

Fig. 3 Initial playout instant synchronization. Coarse synchronization



protocol providing the global time reference, which, in turn, depends on the environment in which it is running and on its design.

The solution for both problems is quite simple. The source communicates (using multicast if the network permits it) to all the receivers the time when the playout of the master stream should start at the beginning of each phase. The start time is specified in relation to the global time reference. We propose the use of a specific control RTCP packet (an extended RTCP APP packet, whose format is defined in [27]) which contains the time at which all the receivers must start the playout of the master stream. The playout processes of the other slave streams will then synchronize with the master stream using inter-stream synchronization. Obviously, as all the receivers have the same global time reference, the 'real' initial playout instant for the master stream will be the same for all receivers (within the performance bounds of the global time protocol).

Initial playout instant calculation The mean network delay experienced by a master stream MU transmitted between source and each receiver must be known. Once the network delay for each receiver has been estimated [$Del_i(m)$, for the i-th receiver), the maximum value is used to calculate the Initial Playout Instant using the expression in Eq. 2 (considering n receivers).

$$T_{\text{in}}(m) = t_0(m) + \text{Del}_{\max}(m) + \Delta \tag{2}$$

$$\text{Del}_{\max}(m) = \max\{\text{Del}_i(m) \colon 0 \le i \le n\} \tag{3}$$

$t_0(m)$ is the instant at which the source starts the transmission of the master stream (m) and $\varDelta$ is the additional time added to allow the initial MUs of each stream to arrive and be buffered at all receivers. The calculation of $\varDelta$ must allow enough time to buffer a sufficient number of MUs to avoid underflow when playout starts, assuming the maximum network delay.

$$\Delta = \frac{[\text{Del}_{\max}(\mu) - \text{Del}_{\min}(\mu)]}{(1-\rho)} \tag{4}$$

The parameter Δ can be obtained at the start of the session and at the beginning of each of its phases. At the start of the session, $\varDelta$ can be obtained by taking delays calculated using the information in the RTCP APP RET packets (explained later) for the beginning of the session. Subsequently, at the beginning of each phase, information from feedback RTCP RR packets from the receivers can be used.

In this way, each receiver will have enough buffered MUs to start the playout and then playout the master stream continuously.

We now set out two ways to calculate the network delay of the i-th receiver, $\text{Del}_i(m)$:

1. The source can send several *ICMP Echo Request* messages (RFC 791) to the receivers and can calculate the mean value of the measured delays.
2. Receivers can send several control packets with timestamps to the source, before starting the media stream transmission, allowing the source to calculate the mean value of the network delay between source and each receiver.[1] We have used this second

[1]We suppose network delay from source to receiver is equal to the one from receiver to source, as ICMP or RTP/RTCP suppose.

method and the control packet used will be a RTCP APP packet extended with useful information for this purpose, which we call the *RTCP APP RET packet*.

RTCP APP RET packet, for network delay calculation This packet will have the format shown in Fig. 4, and contains the following specific fields:

- *Packet type (PT)*. Contains the constant '204' (RTCP APP packet, accordingly to [27]).
- *Name*. Sequence of four ASCII characters. We have chosen the name 'RET'.
- *NTP timestamp*. 64 bits indicating the wallclock time when this packet was sent.

Compared to the use of echo packets, the use of RTCP APP RET packets has the advantage that the source will know the maximum delay between it and the receivers more quickly, since RTCP APP RET packets are sent when the multimedia tools start. As it is an RTCP packet, the receivers will not all send the packets at the same time, which may overload the network or source. Instead, each receiver will calculate the time of RTCP packet transmission according to RFC 3550 [27] and there will be a minimal effect on scalability.

RTCP APP TIN packet, to indicate the initial playout instant To send T_{in}, the Initial Playout Instant, to the receivers, we have used another extended RTCP APP packet, which we call the *RTCP APP TIN packet*, with the same format as an RTCP APP RET packet, but with different meaning assigned to the specific fields:

- *Name.* In this case, we have chosen the name 'TIN'.
- *NTP timestamp.* 64 bits indicating the wall clock time when the receivers' playout must start.

Once the source has transmitted the RTCP APP TIN packet, it will start sending MUs, which will be buffered by the receivers until their clocks match the Initial Playout Instant.

As shown in the example in Fig. 3, the playout processes of the master stream would receive two Coarse Synchronization indications from the source (RTCP APP TIN packets) during the session: one with the NTP timestamp corresponding to t_0 (phase 1) and another with the NTP timestamp corresponding to t_1 (phase 2).

We have also considered the use of the RTCP SR (sender report) packet [27] for this purpose. However, since the source sends these packets only after the flow transmission has

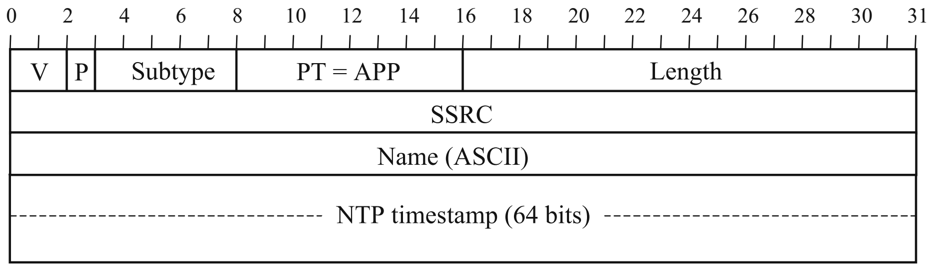


Fig. 4 RTCP APP RET and RTCP APP TIN packet formats

already started and we need to communicate the instant before the transmission starts, the RTCP SR packet was unsuitable. Moreover, the size of this packet is larger than the proposed RTP APP ACT packet.

2.1.2 Fine synchronization between receivers

Once we have chosen an Initial Playout Instant and achieved Coarse Synchronization, the algorithm must guarantee fine synchronization between the playout processes of the different receivers. We call this '*Fine Synchronization between Receivers*' because it is not a local synchronization between streams (inter-stream, like *lip-sync*), but rather a synchronization of the states of the playout processes of the master stream in each receiver participating in the session (we also refer to this as *Distributed Synchronization*, Fig. 5).

To tackle this problem, we again base our algorithm on the existence of a global time reference and on the simplicity of the RTCP feedback mechanism and the ability to define new extensions for RTCP APP packets. Moreover, the RTP packet headers carry timestamps that facilitate synchronization between receivers. This Synchronization process consists of two phases, as depicted in Fig. 6.

First phase In this phase, playout processes inform the Synchronizer Source, more or less periodically (specified in RFC 3550), of their state, using control RTCP RR packets (*feedback messages*), which we have extended to facilitate synchronization. We propose

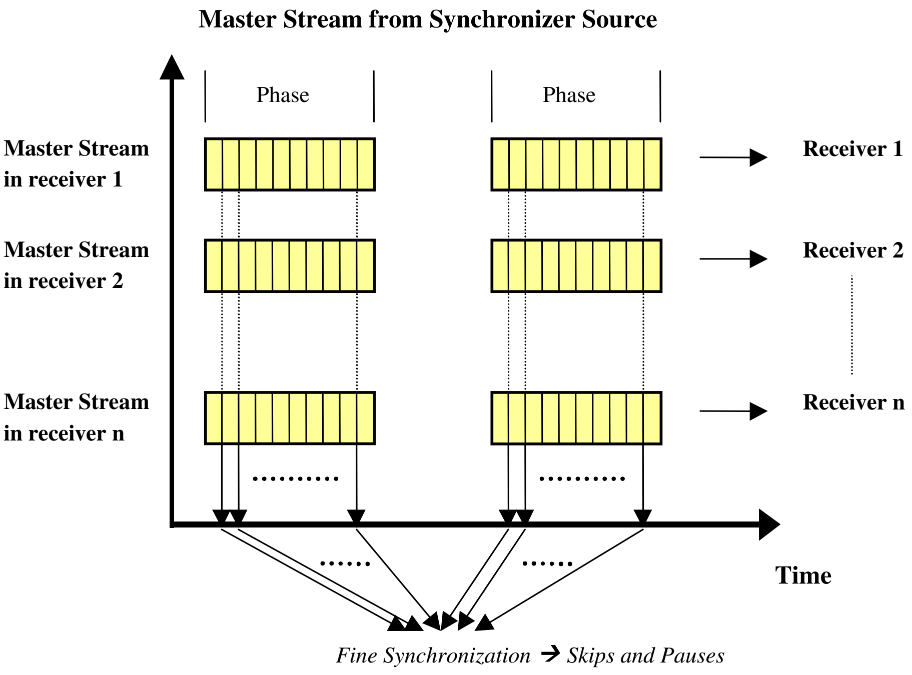

Master Stream from Synchronizer Source
Phase
Phase
Master Stream in receiver 1
Master Stream in receiver 2
Master Stream in receiver n
Receiver 1
Receiver 2
Receiver n
Time
Fine Synchronization → Skips and Pauses

Fig. 5 Fine synchronization between receivers

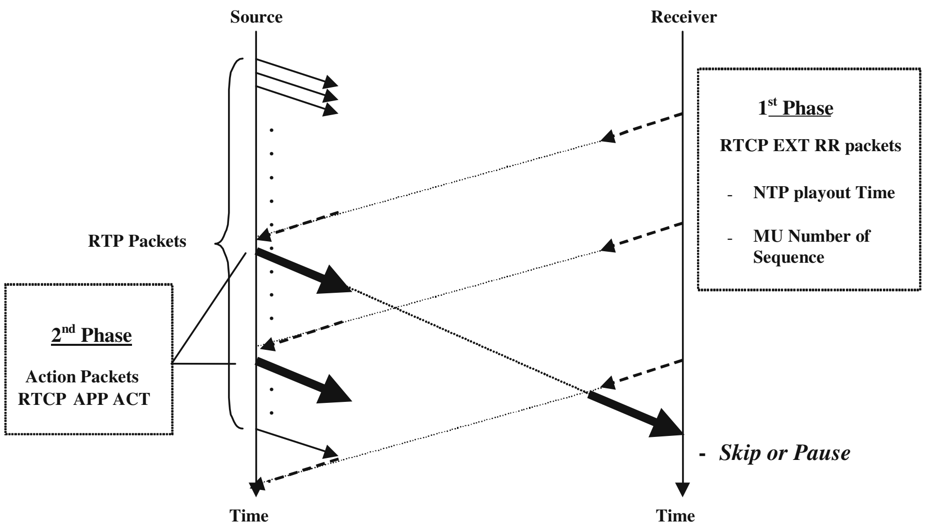


Fig. 6 Control packets for fine synchronization between receivers

adding an extension to the format of these RTCP packets, in accordance with RFC 3550. Our extension will add the following information: the sequence number of the last played MU from the Synchronizer Source and the NTP timestamp of the instant of its playout (Fig. 7).

We refer to this packet extension as an *RTCP RR EXT packet.* The packet has been extended with the following new fields:

- *NTP Timestamp*: 64 bit NTP timestamp for the instant the packet was sent.
- *Last MU played from SSRC_i*: Sequence number of the last played MU from the source with SSRC_i identifier. There will be an extra field for each source in the multimedia session.
- R_i *Bit*: Single bit ('1' or '0') used by the Synchronizer Source to check if the received feedback information is out of date and whether it should be considered in its calculations (explained later).

Since we use RTCP RR feedback packets, whose transmission rate is calculated according to RFC 3550, we do not need to calculate the optimum transmission rate for feedback information, as is required by the solutions presented in [8, 9, 23–26].

If an RTCP RR EXT packet sent by a receiver is lost, the source will not do anything until a future RTCP RR EXT packet is received. This does not significantly affect our approach since these packets are sent periodically, according to RFC 3550.

Second Phase Once the Synchronizer Source has received all of the RTCP RR EXT packets from all of the receivers, an algorithm is used to determine whether receivers are too fast or too slow, relative to the selected *Master Receiver* reference. The source will instruct fast or slow receivers to '*skip*' or '*pause*' MUs to correct the deviation. These instructions are sent to receivers in '*Action*' *Packets*, as illustrated in Figs. 6 and 8.

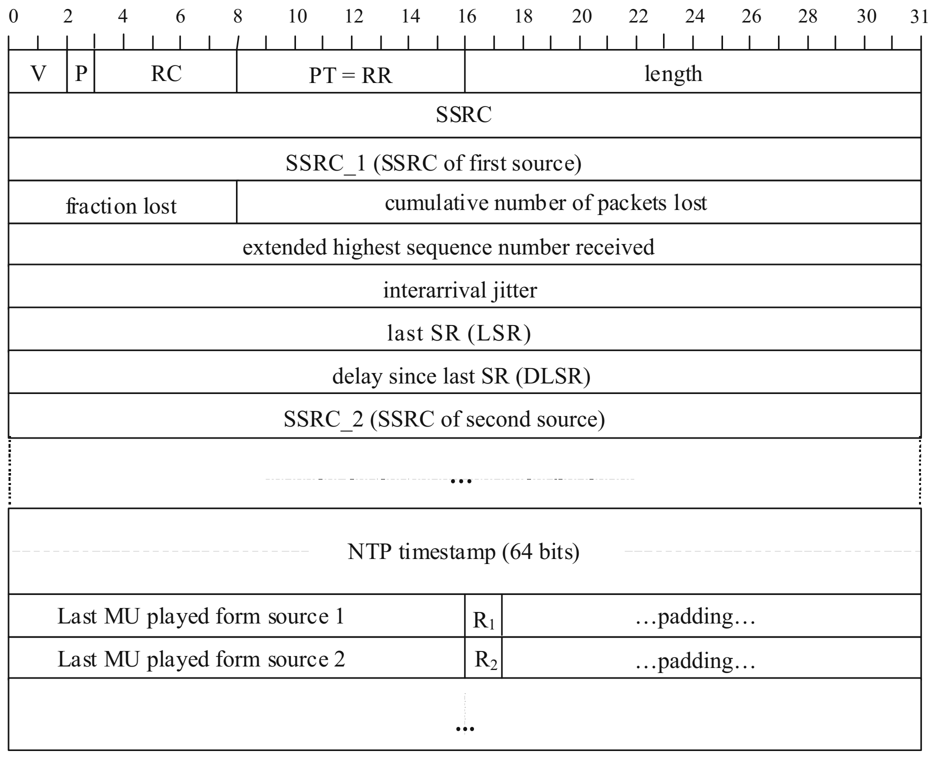

0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 31
V
P
RC
PT = RR
length
SSRC
SSRC_1 (SSRC of first source)
fraction lost
cumulative number of packets lost
extended highest sequence number received
interarrival jitter
last SR (LSR)
delay since last SR (DLSR)
SSRC_2 (SSRC of second source)
...
NTP timestamp (64 bits)
Last MU played form source 1
R_1
...padding...
Last MU played form source 2
R_2
...padding...
...

Fig. 7 RTCP RR extended packet format

During the session, the Synchronizer Source receives RTCP RR EXT packets from the receivers playing the master stream. Information required by the algorithm is extracted from these packes, specifically, receiver identifiers (SSRC [27]), the last MU played by each receiver and the NTP timestamp for the instant at which the last MU was played. This information is stored in an internal table with one row per receiver (Table 1). This table must be updated with the most recent information received from each receiver. The Synchronizer Source updates the table every time an RTCP RR EXT packet is received.

The 'playing bit' column contains a binary value ('1' or '0') and indicates whether a receiver is active (receiving data). This indicates to the source whether or not to consider the information for that receiver when determining the reference playout point. The bit is refreshed when a receiver joins or leaves the session. In this way, departed receivers do not affect the remaining receivers.

Once the table has been updated with the required information, the Synchronizer Source will select the master receiver. This selection may be static or dynamic and an existing algorithm [2] may be used.

RTCP APP ACT packet ('action' message) Once the Synchronizer Source has chosen the master receiver to be used as a reference, it will calculate the asynchrony between the slave receivers' master stream playout processes and those of the master receiver. If the asynchrony for a receiver exceeds a defined allowed maximum limit, the Synchronizer

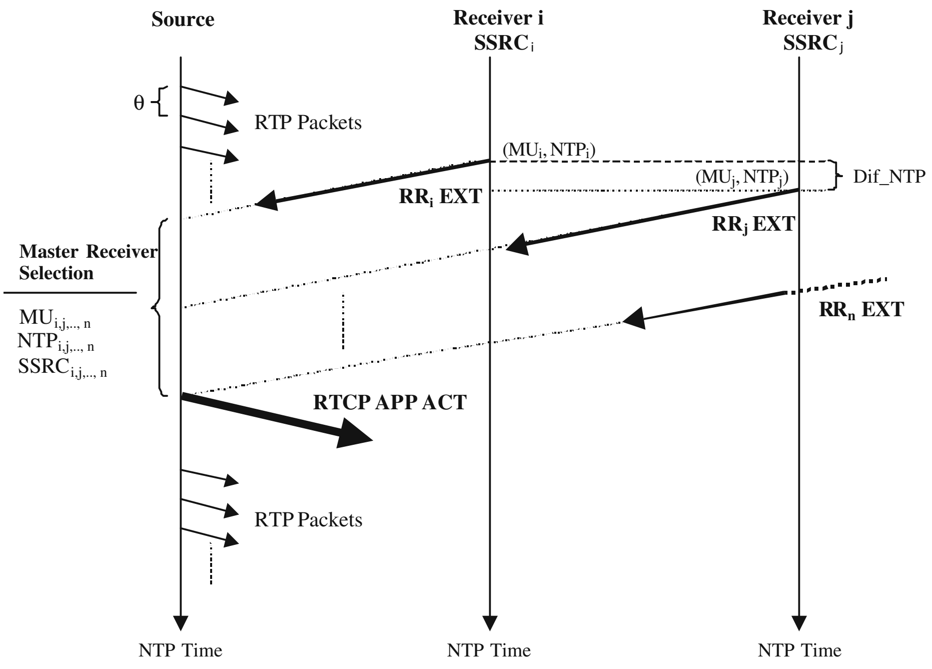


Fig. 8 Message exchange

Source will send (multicast) an '*action*' message to all the receivers causing them to adapt to the playout point of the master stream in the master receiver. Late slave receivers will '*skip*' MUs and fast slave receivers will '*pause*' MUs to achieve master stream synchronization with respect to the master receiver. Immediately after '*action*' message transmission, the source erases the information in the above table.

'*Action*' messages are also implemented using RTCP APP packets. The following fields have been added to these *RTCP APP ACT packets* (the format of which is shown in Fig. 9):

- *NTP Timestamp*: 64 bits containing the global time instant in which the receivers should play the MU corresponding to the *Sequence Number* contained in the next field.
- *A Bits*: 2 bits used to code the type of algorithm to be used by the source to select the master receiver.
- *R Bit*: This bit is necessary to solve a problem we discovered during the validation of the proposal, explained below.

In the interval between the sending of the RTCP APP ACT packet by the Synchronizer Source and the receipt of the packet and adaptation of the playout point by a receiver, the

Table 1 Required information, supposing n receivers

SSRC	Last MU	NTP timestamp	Playing Bit R_i
SSRC$_1$	MU$_1$	NTP$_1$	bit$_1$
SSRC$_2$	MU$_2$	NTP$_2$	bit$_2$
SSRC$_n$	MU$_n$	NTP$_n$	bit$_n$

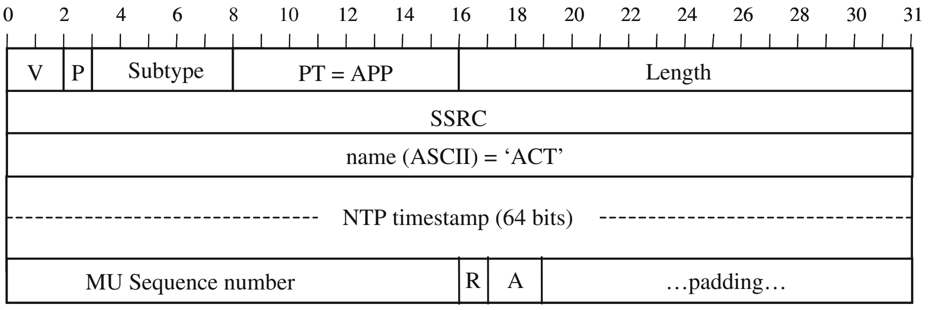


Fig. 9 RTCP APP ACT packet format

receiver may already have sent one (or more) RTCP RR EXT packets. These packets will reach the Synchronizer Source too late, with old information which should be discarded by the source. To avoid this situation, we have included an *R bit* whose value alternates between ‘1’ and ‘0’ during each cycle (period of time between two consecutive RTCP APP ACT packets sent by the same Synchronizer Source). This allows the Synchronizer Source to reject RTCP RR EXT packets sent by receivers before implementing the ‘*skip*’ or ‘*pause*’ actions to adapt their playout points.

Figure 10 illustrates this process. The initial RTCP RR EXT packets sent by receivers to the Synchronizer Source will have the R bit for that source set to ‘0’. When the source completes its table, it will calculate the values for the fields of the next RTCP APP ACT

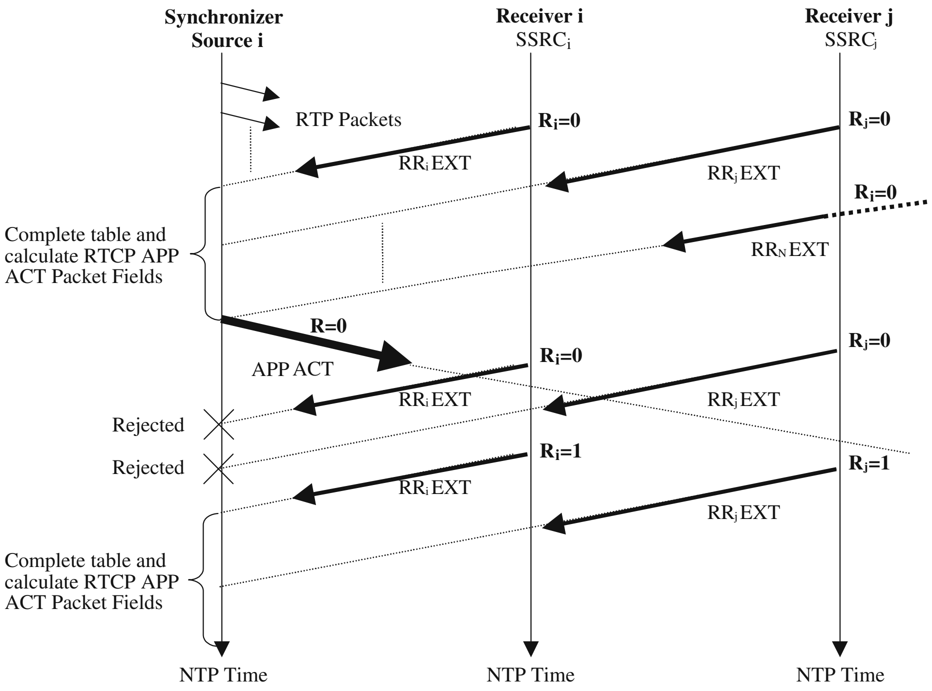


Fig. 10 RTCP APP ACT using *R* bit

packet. The first RTCP APP ACT packets sent by the source will have their *R* bit set to '0'. Any RTCP RR EXT packets with an *R* bit set to '0' which are received by the source after the RTCP APP ACT packet has been sent will be rejected. Then, only those packets with the *R* bit set to '1' will be accepted until the next RTCP APP ACT is sent. The process continues in this way, alternating the value of the *R* bit.

After receiving an RTCP APP ACT packet from the *i*-th Synchronizer Source, a receiver will set the corresponding R_i bit in its next RTCP RR EXT packet to the opposite value of the *R* bit in the RTCP APP ACT packet. The value of the *R* bit in subsequent RTCP APP ACT packets sent by the source to the receivers, will be alternated between cycles, solving the aforementioned problem.

In this case, we could also consider using the RTCP SR (sender report) packet [27], extended for this purpose. This packet, however, was not designed for group synchronization purposes (although it is useful for local inter-stream synchronization) and the Synchronizer Source would have to force its transmission when the asynchrony between receivers exceeds a defined allowed maximum limit. Furthermore, the size of these packets is larger than the proposed RTP APP ACT packet.

RTCP APP ACT packet fields calculation Once the Synchronizer Source has received data from all receivers, it has to calculate the MU sequence number, $\mathrm{MU_{act}}$, and NTP timestamp, $\mathrm{NTP_{act}}$, to be sent in the next '*action*' RTCP APP ACT packet. $\mathrm{MU_{act}}$ will be the sequence number of the next RTP packet that the source will send. Then, to calculate the NTP timestamp corresponding to the time at which $\mathrm{MU_{act}}$ must be played, the source uses the sequence number and NTP timestamp data received from the master receiver ($\mathrm{MU_{master}}$ and $\mathrm{NTP_{master}}$, respectively).

The time (in NTP units) since the MU with sequence number $\mathrm{MU_{master}}$ was transmitted is given by Equation 5, where θ(ntp) is the generation rate in NTP units.

$$\mathrm{Diff\_MU(ntp)} = (\mathrm{MU_{act}} - \mathrm{MU_{master}})^{*}\theta(\mathrm{ntp}) \tag{5}$$

The source generation rate and master receiver playout rate are assumed to be equal (with no drift or variation between the rates) and we denote this rate θ. If the time since the MU with sequence number $\mathrm{MU_{master}}$ was transmitted [Diff_MU(ntp)] is added to the global NTP time obtained from the master receiver, we obtain the time (in NTP units) at which that receiver should playout the MU with sequence number $\mathrm{MU_{act}}$ (assuming that it continues playing out at the same rate, with no drift or variation of the rate). Nevertheless, as the source generation rate may differ slightly from the master receiver playout rate, there may be a possible time interval for the consumption ($\mathrm{I_c[MU_{act}]}$), in which we can be sure that master receiver will playout that MU.

$$I_c[\mathrm{MU_{act}}] = [I_{c1}, I_{c2}] \tag{6}$$

$$I_{c1} = \mathrm{NTP_{master}} + (\mathrm{MU_{act}} - \mathrm{MU_{master}}) * \theta(ntp) * (1+\rho) \tag{7}$$

$$I_{c2} = \mathrm{NTP_{master}} + (\mathrm{MU_{act}} - \mathrm{MU_{master}}) * \theta(ntp) * (1-\rho) \tag{8}$$

These equations do not depend on network jitter, and, therefore, do not affect our algorithm. In contrast with this approach, network jitter does cause an error in the *Feedback* *protocol* [23–26]. Depending on the type of application, the error induced by the use of a global time protocol (such as NTP or GPS) can be disregarded because it is of the order of several milliseconds or lower.

In the implemention of our algorithm in the test applications, the source chooses the mean time in the above interval to configure the corresponding field of the RTCP APP ACT packet:

$$\mathrm{NTP_{act}} = \mathrm{NTP_{master}} + \mathrm{Diff\_MU(ntp)} \tag{9}$$

Thus, we assume a minimal error, limited to $\pm\rho$.

Receiver actions Immediately after receiving an RTCP APP ACT packet, the playout process of the master stream in each receiver will deduce whether it has to '*skip*' or '*pause*' MUs. This is done by comparing its $\mathrm{MU_{act}}$ and $\mathrm{NTP_{act}}$ values with the MU it is playing at that moment and the global time at that instant. These actions will result in an increase or decrease in the playout rate, with playout discontinuities. Adjusting the playout rate in this way will result in synchronous playout of the master stream by all receivers. We now present, in pseudocode, the synchronization process followed by each receiver. All times are in units of milliseconds, $\mathrm{MU_{ply}}$ is the sequence number of the last played MU and $\mathrm{NTP_{ply}}$ is the global (NTP) instant at which that MU was played).

```
IF ($MU_{act} > MU_{ply}$) THEN
    Diff_MU = $MU_{act} - MU_{ply}$
    Diff_MU(ms) = Diff_MU * $\theta$ (ms)
    IF ($NTP_{act} > NTP_{ply}$) THEN
        Diff_NTP = $NTP_{act} - NTP_{ply}$
        Diff_NTP(ms) = convert_to_ms(Diff_NTP)
        IF (Diff_NTP(ms) > Diff_MU(ms)) THEN
            Adjustment = Diff_NTP(ms) – Diff_MU(ms)
            Playout_Delay(ms) = Playout_Delay(ms)+ Adjustment
        ELSE
            Adjustment = Diff_MU(ms) – Diff_NTP(ms)
            Playout_Delay(ms) = Playout_Delay(ms)- Adjustment
        END IF
    ELSE
        Diff_NTP = $NTP_{ply} - NTP_{act}$
        Diff_NTP(ms) = convert_to_ms(Diff_NTP)
        Adjustment = Diff_MU(ms) + Diff_NTP(ms)
        Playout_Delay(ms) = Playout_Delay(ms)- Adjustment
    END IF
END IF
```

The *Playout delay* of a packet is the delay from when it was transmitted to when its content is played. When a receiver is playing too quickly, the playout time of the MU contained in the RTP packet currently being processed is delayed ('*pause*' *effect*). If the receiver is playing too slowly, it will advance the playout time of the next MU to be played ('*skip*' *effect*). In contrast with the solution proposed in [23–26], buffer overflow and underflow is avoided since the only action taken is to delay or to advance the playout of the MU that has just arrived. This is achieved simply by increasing or decreasing the value of its *playout delay.*

2.2 Inter-stream synchronization (local, several streams)

In parallel with coarse and fine synchronization of the master stream between receivers, each receiver should also maintain internal synchronization between additional streams played by the receiver. This process is called *inter-stream synchronization*.

In this case, we have again chosen a *master/slave synchronization mechanism*, in which the playout of additional streams is synchronized with a chosen *local master stream*.

Usually the local *master* stream will be the stream with the strictest QoS requirements (normally the audio stream). Alternatively the local master stream may be determined by the multimedia application. Each receiver will '*pause*' or '*skip*' slave streams to synchronize their playout with the local master stream. As a result, discontinuities in the receiver playout processes of the slave streams are inevitable.

Since each playout process may use a different time reference (e.g. the playout processes of audio and video streams [15]), the information provided by RTCP and the global time protocol should be used to negotiate their playout points.

The negotiation of a playout point starts with the exchange of messages containing the *playout delay* in NTP units (common reference), through an internal inter process communication channel, followed by an inter-stream synchronization process. There are several options for performing this synchronization. One approach (already implemented in the *vic* [19] and *rat* [14] mBone tools) is shown in Fig. 11a.

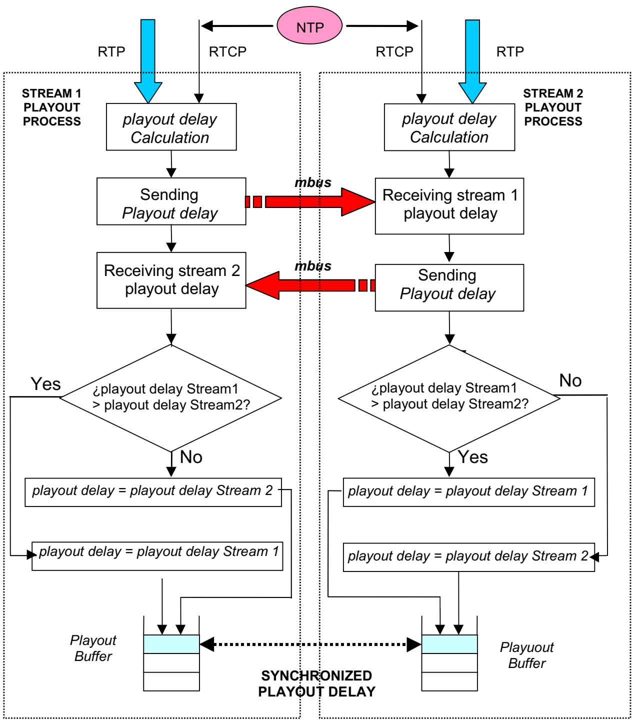
Using this approach, the playout points of all processes are synchronized with the playout point of the slowest process (with largest *playout delay*). The slowest stream is considered to be the local master for which all other processes will '*pause*' or 'wait'. This method considers all the streams and their playout processes equally (there is no fixed local master stream and at any instant the faster playout processes wait for the slower one). Nevertheless, multimedia streams have different characteristics. For example, frequent adjustment of an audio stream to achieve synchronization may be more perceptible to users and intrusive than adjustments of a video stream.

The above inter-stream synchronization scheme must be modified to integrate it with group synchronization. In particular, the group synchronization process must indirectly cause local inter-stream synchronization of slave streams. To achieve this, the master stream used for both group and inter-stream synchronization must be the same.

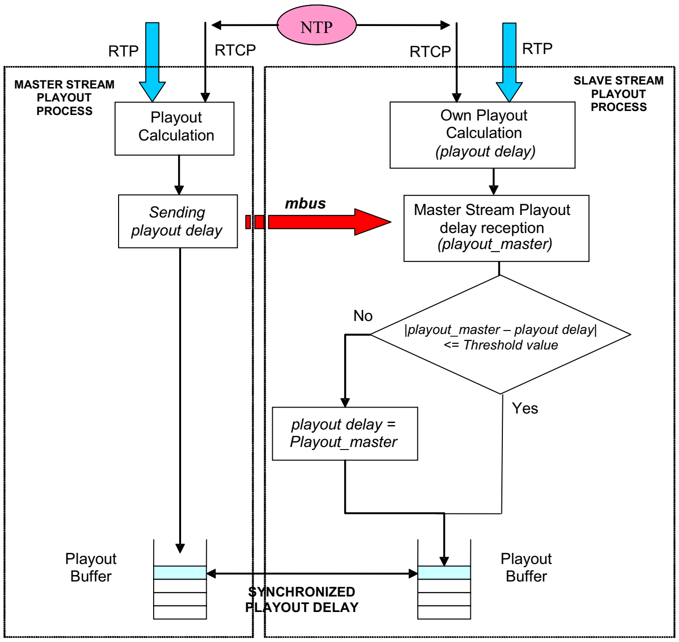
If the master stream used for group synchronization between receivers is also selected as the master stream for inter-stream synchronization, synchronization of this master stream between receivers will be automatically propagated to the slave streams in each receiver, thus achieving inter-stream synchronization. As a result, synchronization of all the playout processes of all the streams at all the receivers is achieved.

If a local master stream other than the group master stream had been selected to perform inter-stream synchronization (e.g., the video stream as the local master and the audio stream as the group master), the playout of the group master stream (e.g. the audio stream) would be adjusted both to achieve synchronization with other receivers and also to achieve synchronization with the local master stream (e.g. the video stream). Synchronizing to the local master stream may result in the loss of synchronity with the other receivers (e.g. with respect to the audio stream). This would in turn affect that synchronization process, resulting in numerous discontinuities in the playout of that stream, thus affecting its intelligibility.

Using the same stream as the master for both local inter-stream and group synchronization avoids the need for the communication (*feedback*) from the playout processes of slave streams to their sources by sending RTCP RR EXT packets. The group synchronization


NTP
RTP
RTCP
RTCP
RTP
STREAM 1 PLAYOUT PROCESS
STREAM 2 PLAYOUT PROCESS
playout delay Calculation
Sending Playout delay
Receiving stream 2 playout delay
Receiving stream 1 playout delay
Sending Playout delay
mbus
mbus
¿playout delay Stream1 > playout delay Stream2?
Yes
No
playout delay = playout delay Stream 2
playout delay = playout delay Stream 1
Playout Buffer
Playuout Buffer
SYNCHRONIZED PLAYOUT DELAY

a)


NTP
RTP
RTCP
RTCP
RTP
MASTER STREAM PLAYOUT PROCESS
SLAVE STREAM PLAYOUT PROCESS
Playout Calculation
Sending playout delay
Own Playout Calculation (playout delay)
mbus
Master Stream Playout delay reception (playout_master)
|playout_master – playout delay| <= Threshold value
No
Yes
playout delay = Playout_master
Playout Buffer
Playout Buffer
SYNCHRONIZED PLAYOUT DELAY

b)

Fig. 11 Inter-stream synchronization generic (**a**) and proposed (**b**) schemes

mechanism only needs to be implemented between the source (Synchronizer Source) and receivers of the master stream.

During the *Fine Synchronization Process between Receivers*, each time a master stream playout process receives an RTCP APP ACT packet, and a '*skip*' or '*pause*' is performed, this is communicated to the playout processes of the slave streams in an internal message, sent using a local inter-process communications channel. In this way, the master stream playout process will only modify its playout delay as a consequence of *group synchronization*, while the playout processes of the slave streams will modify their playout delays as a consequence of *inter-stream synchronization* (which is implicitly affected by *group synchronization* of the master stream). This avoids the playout of the same stream from being modified by both synchronization processes, possibly in a competing manner resulting in discontinuities.

The group synchronization master stream will communicate its playout delay to the other local playout processes. These other slave stream processes will compare their own playout delay with that of the group master and, if the asynchrony exceeds a threshold (chosen to avoid frequent adjustments), will '*skip*' or '*pause*' playback (by decreasing or increasing the value of its playout delay) if they are, respectively, behind or ahead of the master stream.

In Fig. 11b, the proposed inter-stream synchronization process is illustrated with one master stream and one slave stream and using the internal communication channel (*mbus* in the figure) for message communication. Each playout process independently calculates the playout delay for each received MU. The playout process of the master stream then communicates this value to the playout process of the slave streams using the internal communication channel. The slave stream playout processes compare the received values with their own playout delay, and determine whether playback is behind or ahead of the master, within the bounds of the maximum asynchronony allowed. If these bounds have been exceeded, the process will update its own value with the received one. By using a threshold value, the frequency of synchronization actions and discontinuities in the playout of the streams are reduced.

Using this scheme, the playout process of the slave stream will initially, in the absence of internal messages, play the stream content as RTP packets arrive. When it begins to receive internal messages from the master stream playout process, buffering will be used, if necessary, to implement '*pause*' or '*skip*' actions. Ultimately the packets from both streams, generated and sent at the same time by the sources, will be stored in the playout buffers of applications with the same playout delay (or with deviations lower than a threshold value), resulting in synchronous playback.

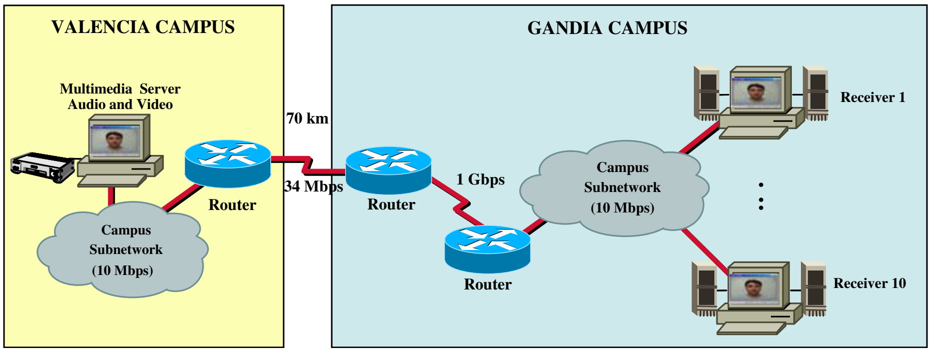

VALENCIA CAMPUS
Multimedia Server Audio and Video
Router
Campus Subnetwork (10 Mbps)
70 km
34 Mbps
GANDIA CAMPUS
Router
1 Gbps
Router
Campus Subnetwork (10 Mbps)
Receiver 1
Receiver 10

Fig. 12 Evaluation scenario

Table 2 Audio and video specifications

Parameter	Audio	Video
Codification	GSM	H.261
MU Rate (MUs/s)	8,000 samples/second	25 frames/second
Inter-MU Time (ms)	20–40	40

3 Evaluation

We have tested our proposal, both objectively and subjectively, in a LAN and in our University WAN (Fig. 12). We have modified existing open source RTP-based mBone tools, such as *vic* [19] (for video) and *rat* [14] (for audio), which include the *mbus* as the inter-processes communication facility. We present only the results for the WAN scenario (worst case). We present the objective and subjectqive evaluation of the multicast transmission of short sessions with separated audio and video streams from a videotape, located in the Valencia Campus, to 10 receivers, located in the Gandia Campus, 70 km away. The bandwidths of the links are shown in the figure. All the systems were synchronized by a stratum-1 NTP server in Red IRIS [29].

The 10 receivers were PCs with 128 MB of RAM and AMD Athlon 1600+XP 1.4 GHz processors and the Windows operating system was used. The multimedia server was also a PC with 512 MB of RAM and an Intel Pentium IV 900MHz processor and the Windows operating system was again used. Table 2 shows the audio and video stream parameters:

For *Group Synchronization*, we configured one receiver as the master receiver and the audio stream as the master stream because the synchronization requirements are stricter than those for the video stream. For *Inter-stream Synchronization* we have used *mbus*

Table 3 Measured parameters for *group synchronization*

Parameter	Explanation
Master stream Playout Delay in each receiver	If all the receivers are synchronized, this parameter will have the same value in each receiver for all the MUs
Adjustments	Adjustments (in milliseconds) made by playout processes of the master stream because of the group synchronization actions
Mean square value of the detected group asynchrony for the master stream	Mean square value of the group synchronization error
Number of control RTCP APP ACT messages	They would have been sent by the Synchronizer Source
Number of RTCP RR EXT messages	They would have been sent by the receivers (with feedback information)
Maximum allowed asynchrony between receivers	If this value is not exceeded source will not send an RTCP APP ACT message, to avoid multiple adjustments
Number of detected and corrected asynchrony situations	Number of group asynchronies
Number of times a master stream MU is 'skipped'	Number of MUs skipped to correct asynchrony situations by slave receivers
Number of times a master stream MU is 'paused'	Number of MUs repeated to correct asynchrony situations by slave receivers
Number of sent RTP packets containing master stream MUs	Total number of sent packets

([16]), already included in the *vic* and *rat* tools. The video playout process adapts its state to the one of the audio playout process by '*skipping*' or '*pausing*' video MUs (frames). In accordance with [28], we have fixed the following limits: ±120 ms as the maximum allowed value for asynchrony between receivers for the master stream (group, distributed), and ±80 ms as the maximum allowed value for asynchrony between audio and video playout processes (local).

3.1 Objective evaluation

We have tested the applications with and without our synchronization algorithm. Without synchronization, a mean asynchrony of 2.5 s between receivers was measured for the master stream, during the course of a 10-min film. Furthermore, the audio and video streams were unsynchronized (no lip-synch) from the beginning of the session, which was very unsatisfactory.

Tables 3 and 4 show the measured parameters for *group* and *inter-stream synchronization*.

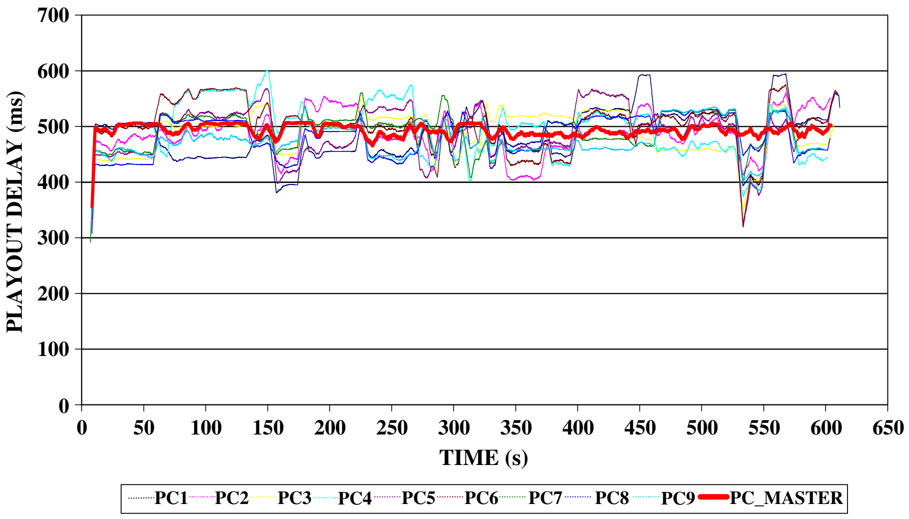
3.1.1 Group-Synchronization evaluation (Distributed Synchronization)

Figure 13a shows the playout delay of the audio (master) stream in all the receivers throughout the entire session. To smooth out short-term fluctuations, a moving average, taking groups of 100 samples, has been used.

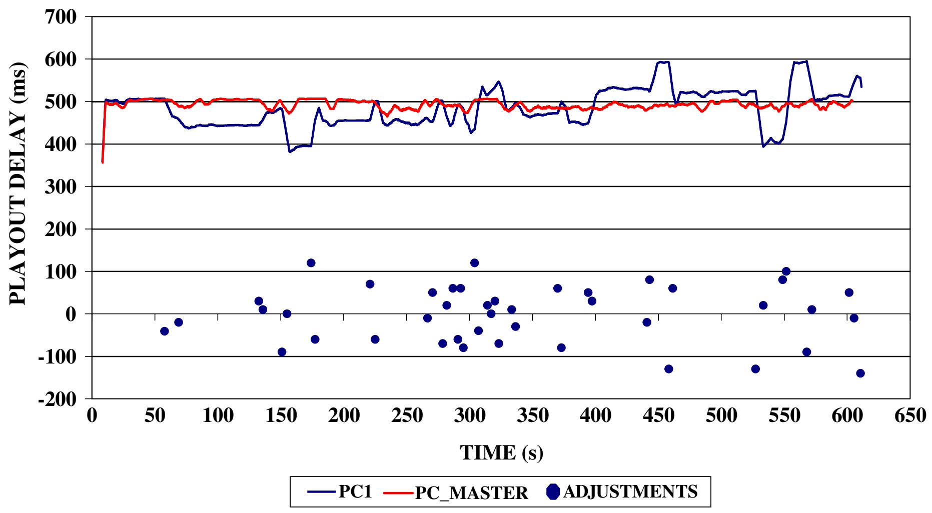
The playout delay increases at the beginning because of the start of the audio and video tools and the resulting resource consumption at the receivers. It is adjusted by each receiver at several points to the playout delay of the master receiver (PC MASTER, red thick line), whose mean value is about 500ms for this session. In Fig. 13b the playout delay for just one slave receiver and for the master receiver are shown, together with the adjustments made in the playout process of the slave receiver ('*skip*' and '*pause*' actions, corresponding respectively to negative and positive adjustments). The playout of the master stream has no adjustments resulting from the group or inter-stream synchronization processes. The most representative measurements are shown in Table 5.

Table 4 Measured parameters for *inter-stream synchronization*

Parameter	Explanation
Playout Delay of the master and slave streams in each receiver	Playout delay of both streams
Adjustments	Adjustments (in milliseconds), made by playout processes of the slave stream because of the inter-stream synchronization actions
Mean square value of the detected inter-stream (audio-video) asynchrony	Mean square value of the inter-stream synchronization error
Number of sent inter-processes messages	Number of mbus messages sent by the playout process of the master stream
Number of detected audio-video asynchronies	Number of inter-stream asynchronies
Number of times a video (slave) stream MU is 'skipped'	Number of slave MUs skipped to correct asynchrony situations
Number of times a video (slave) stream MU is 'paused' (repeated)	Number of slave MUs repeated to correct asynchrony situations



a)



b)

Fig. 13 Playout delay of the playout processes of the master (audio) stream. Adjustments for one receiver

We can see that few control packets were used. The source sent only one RTCP APP TIN to communicate the Initial Playout Instant and, subsequently, 42 RTCP APP ACT packets were sent to maintain receiver synchronization. The number of control packets (43) was only 0.14% of the total number of packets (control and data) sent by the Synchronizer Source. The 2,233 RTCP RR EXT control packets sent by the receivers represent 6.88% of the total number of packets (control and data) sent by all of the applications and 7.39% of the 30,200 RTP data packets sent.

Table 5 Most representative measurements for Group synchronization

Receiver statistics	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10 (MASTER)
Mean square value of the asynchrony (ms^2)	4,437	2,897	4,225	3,850	3,113	3,985	3,154	3,544	2,821	–
Number of RTCP APP TIN messages sent	1									
Number of RTCP APP ACT messages sent	42									
Number of RTCP RR EXT messages (Feedback)	225	222	222	225	219	230	213	227	220	230
Number of detected asynchronies audio–audio (between receivers)	3	0	1	2	0	2	2	1	2	–
Master stream MUs 'Skipped' due to group Synchronization	60	42	57	51	44	50	40	43	39	–
Master stream MUs 'Paused' due to group Synchronization	56	46	58	50	44	48	39	53	38	–
Number of sent RTP packets	30,200									

With a tolerance of ±120 ms between receivers with respect to the master stream (audio), we can see that receivers, in general, detected and corrected from 0 to 3 asynchrony situations (where this tolerance was exceeded) during the session.

The *mean square value of detected asynchrony* did not exceed the limit of 14,400 ms^2 (square of the maximum allowed asynchrony, ±120 ms) at any of the receivers. The mean value for the square of the detected asynchrony in all the receivers was 3,558.44 ms^2. These values were very low and represent good results for group synchronization.

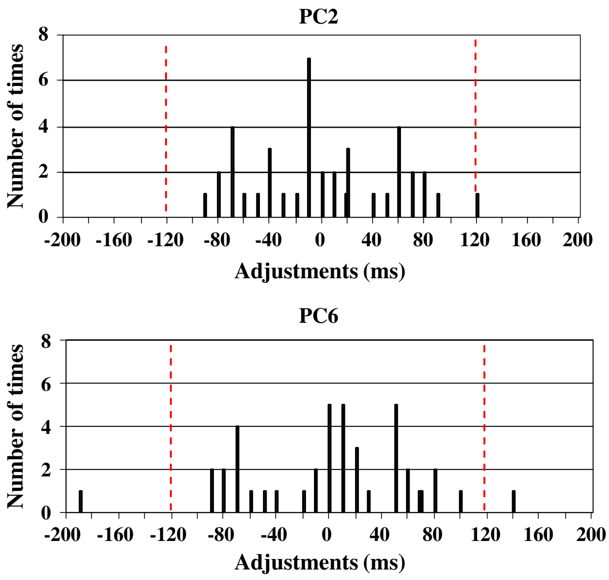
The receiver with most '*skipped*' MUs was PC1 with 60 '*skips*', while the receiver with most '*paused*' MUs was PC3, with 58 '*pauses*', which are quite low values. Moreover, the '*skips*' or '*pauses*' were not consecutive but sporadic and they were not considered annoying or intrusive by the users in the subjective evaluation, the results of which are presented later.

Figure 14 shows, for two slave receivers (the best and the worst ones), the distribution of master stream playout adjustments that result from the reception of an RTCP APP ACT packet. The limit of ±120 milliseconds is marked with red dotted lines. Figure 15 shows, for both receivers, the mean square of the adjustments during the session (mean square of the group synchronization error). The limit of 14,400 ms^2 has been marked with a red dotted line. We can see that the worst receiver (PC6) exceeded the limit (despite the mean value being much lower) on very few occasions but the asynchrony was soon detected and corrected by the algorithm. Subjective evaluation showed that those corrections were not intrusive for users.

We conclude that group synchronization mechanism worked correctly throughout the session.

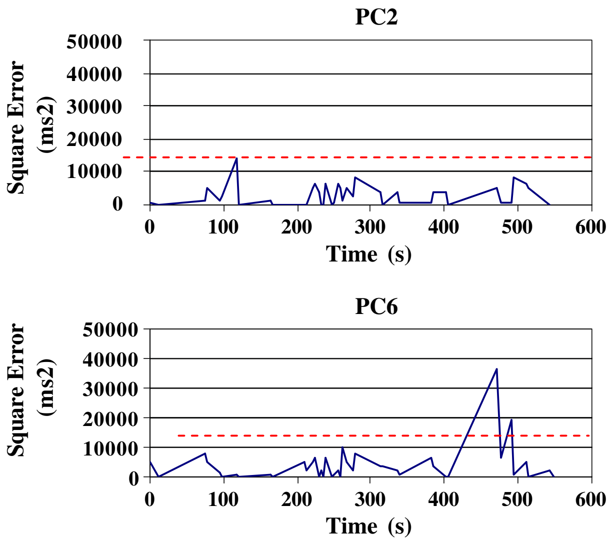
Master stream MU sequence number played in each receiver To corroborate the satisfactory performance of our proposal, we checked that all of the receivers played the MUs without exceeding the configured limits (±120 ms).

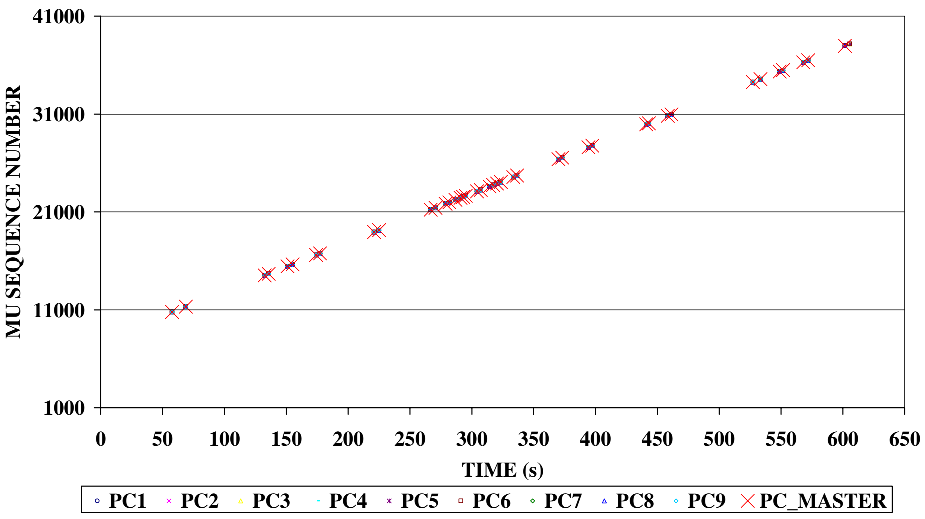
Fig. 14 Adjustments (error) distribution of the playout of the master stream in two receivers



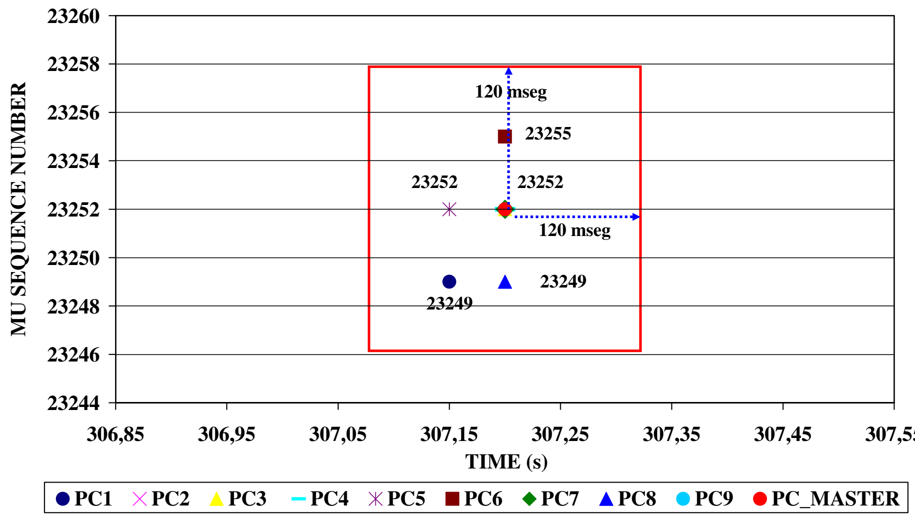
We first confirmed that they all played the first MU (with sequence number 7,895) at the same NTP instant (*Initial Playout Instant*), as indicated by the RTCP APP TIN packet sent by the Source. This confirmed that the mechanism for synchronizing the Initial Playout Instant worked correctly, with an asynchrony within the bounds of the precision of NTP. Figure 16a shows several MUs which were checked during the session for each receiver. We can see they were played at practically at the same time by all the receivers. Examining a small number of MUs, (Fig. 16b) we can observe that all the receivers played the MU without exceeding the limit of ±120 ms (red square) from the instant at which the master receiver played MU number 23,252 (centered thick red dot).

Fig. 15 Mean square value of detected asynchrony between master streams in two receivers





a)



b)

Fig. 16 Played audio MU sequence number

3.1.2 Inter-stream Synchronization evaluation (Local Synchronization)

The maximum allowed asynchrony between the playout processes of the audio and video streams at each receiver was limited to ±80 milliseconds and, if it was occasionally exceeded, with a higher limit of ±160 milliseconds, in accordance with [28]. Our proposed mechanism can be considered to operate correctly if the asynchrony between both streams is maintained below the lower limit. In order to avoid unnecessary control messages between playout processes, we configured the applications so that the master would send a control message to maintain any asynchrony within these bounds only when a threshold value of ±40 milliseconds was exceeded.

Playout delay of the master and slave streams If the playout processes of both streams are synchronized, the values of the playout delay for both streams will coincide (with a possible allowed asynchrony of ±80 ms). In this case we do not consider it necessary to show the performance for all the receivers but just for one slave (obviously the results for the master receiver were better).

Figure 17 presents the playout delay of the audio (master) and video (slave) streams for the receiver PC1 throughout the session, using a moving average with groups of 100 samples to smooth any variations. The graph also includes the measured asynchrony between both audio and video playout processes and the adjustments made by the slave (video) playout process. We can again see the large initial adjustment, as explained previously. Throughout the session, the video playout process made 76 adjustments, all of which exceeded ±40 ms (configured as the threshold value beyond which *mbus* messages would be sent). As expected, the detected asynchronies did not exceed the limit of ±80 ms and the mean square values for each receiver were below 6,400 ms^2. Moreover, the mean square of the detected asynchrony (ms^2) for all of the receivers was 432.11 ms^2. Table 6 shows the results obtained for the most representative measurements from the evaluation.

The master stream playout process of receiver PC2 sent 367 synchronization messages to the slave stream playout process. PC1 sent the least *mbus* messages (76). PC4 exhibited the most occurrences of asynchrony (with 23 occurrences), while PC7 exhibited the least occurrences of asynchrony (nine occurrences). The playout process at PC2 '*skipped*' 186 MUs during the session and '*paused*' 282 MUs. Figure 18 shows the distribution of adjustments made by the playout process of the slave stream (video) in two receivers (the

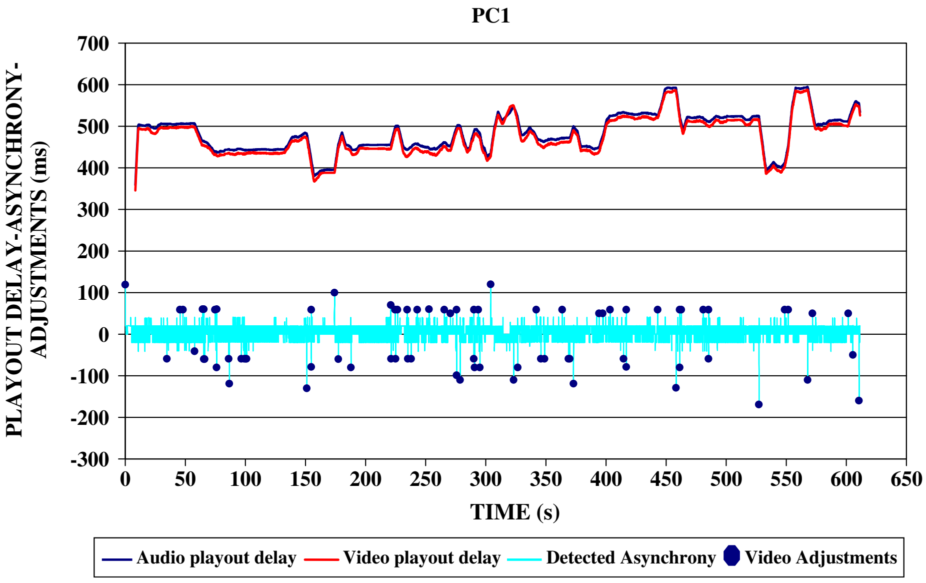


Fig. 17 *Playout delay of the streams* played by PC1, detected asynchrony and adjustments in the playout process of the slave stream

Table 6 Most representative measurements for Inter-stream synchronization

Receiver statistics	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10 (MASTER)
Mean square value of the detected asynchrony (ms^2)[a]	387.37	542.05	441	483.25	361.31	451.97	379.18	385.59	430.29	459.12
Number of sent *mbus* messages	76	367	84	177	78	180	134	126	159	163
Configured audio-video maximum allowed asynchrony (±ms)	Configured threshold: ±40[b]					Maximum allowed: ±80 ms				
Number of asynchronies audio-video	13	11	14	23	12	15	9	10	13	12
Slave (video) MUs ‘*Skipped*’ due to Inter-stream synchronization	72	186	73	120	69	109	101	104	136	101
Slave (video) MUs ‘*Paused*’ due to Inter-stream synchronization	46	282	56	131	49	137	93	79	85	136

[a] We have not considered the initial adjustments during application startup.

[b] The applications were configured to trigger resynchronization actions when they detected differences in playout processes exceeding this value, to avoid exceeding the maximum allowed value and thereby avoid asynchrony.

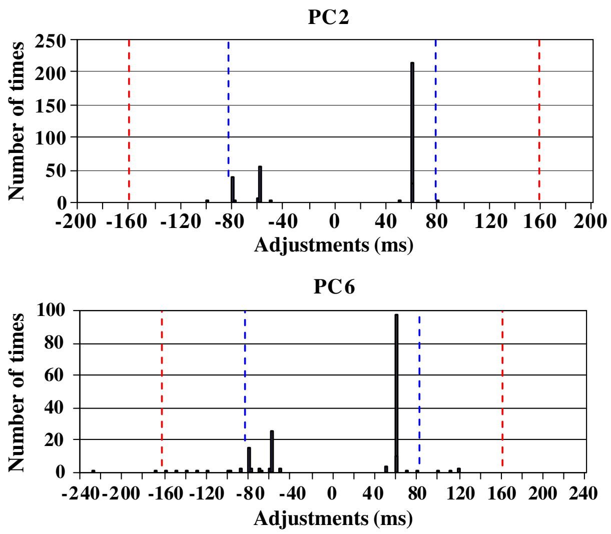


Fig. 18 Adjustment (error) distribution for the playout of the slave stream in two receivers

best and the worst ones) to synchronize with the master stream playout process. Most of the asynchronies were in the intervals [−0, −40] and [40, 80] ms, as expected.

Our algorithm maintained the *Mean Square of the detected Inter-stream asynchrony* (mean square of the inter-stream synchronization error) below 6,400 ms^2 (corresponding to an asynchrony of ±80 ms) or, in the worst case, lower than 25,600 ms^2 (corresponding to an asynchrony of ±160 ms). The above table shows that the worst obtained value was for PC2, with 542.05 ms^2, very much lower than the limit. In Fig. 19, the distributions of this measurement for the same two receivers are shown. The above limits were exceeded on

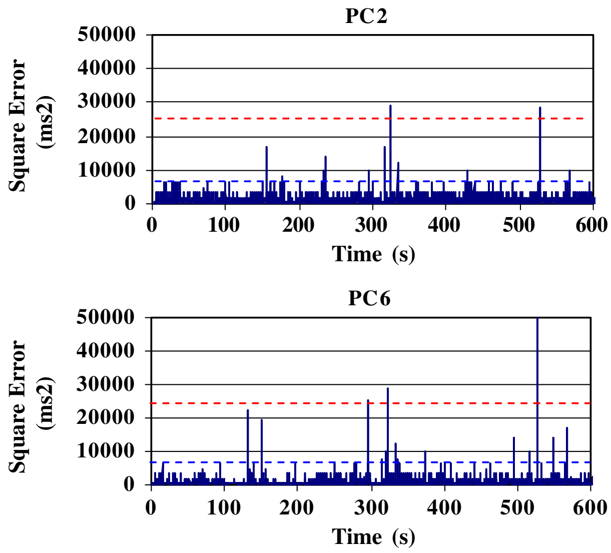
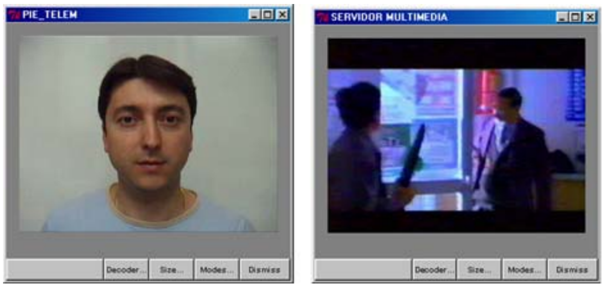


Fig. 19 Mean square value of detected asynchrony between streams in two receivers

Fig. 20 Sequence types



a) First Plane Sequence b) Action Film sequence

very few occasions. Subjective evaluation also showed, in this case, that those occurrences were not intrusive to application users.

3.2 Subjective Evaluation

We complemented the objective evaluation with a subjective evaluation. In multimedia systems design, the study of the functional qualities of the human audiovisual system [7] is very important. Such a study allows us to determine a quality interval such that the limits of the interval represent acceptable quality while performance within the upper and lower limits of the interval represents a quality level exceeding the perception of users.

This evaluation was centred on studying how a user perceived the synchronization effect, from a psychophysics point of view. We based our evaluation on the work presented by Steinmetz in [28]. We selected 20 users (8 females, 12 males) without experience in subjective evaluation and in synchronization techniques. Only 15% of them had experience as users of multimedia systems. We showed them two types of video sequence (Fig. 20): *First Plane*, representative of news programmes, and an *Action Film*; with three synchronization grades: *without synchronization* (without any synchronization algorithm), only *with inter-stream synchronization* (in each receiver independently) and *with group synchronization* (including group and inter-stream synchronization).

Table 7 Test parameters

	Without synchronization		With inter-stream synchronization		With group synchronization	
Video stream playout rate	15 frames/ second	25 frames/ second	15 frames/ second	25 frames/ second	15 frames/ second	25 frames/ second
Sequence type	First plane Action film	First plane Action film	First plane Action film	First plane Action film	First plane Action film	First plane Action film
Maximum allowed asynchrony value (*Skew*) between audio and video streams			±80 ms		±80 ms	
Maximum allowed asynchrony value between audio receivers					±120 ms	

Table 8 Scales used in the tests

Grade	Synchronization degradation	Synchronization Quality
0	Not sure of accepting it	Not sure
1	Very annoying	Totally without synchronization
2	Annoying	–
3	Slightly annoying	–
4	Perceptible but not annoying	–
5	Imperceptible	Totally with synchronization

The parameters of the tests are shown in Table 7 and Table 8 shows the evaluation scales used by the users after watching the sequences, both based on the Recommendation UIT-R BT 500-7 [12]. First, users had to evaluate the quality of the synchronization of the sequences using a 6 degree quality scale (0 to 5), where '5' indicates total synchronization and '1' indicates a total lack of synchronization between streams (inter-stream synchronization). Second, they had to evaluate the quality of the presentation also using a 6 degree quality scale (0 to 5), where '5' indicates good presentation without abnormal effects ('*skips*', '*pauses*', etc.), and '1' indicates a very annoying presentation due to abnormal effects. In both scales, a '0' value indicates user indecision.

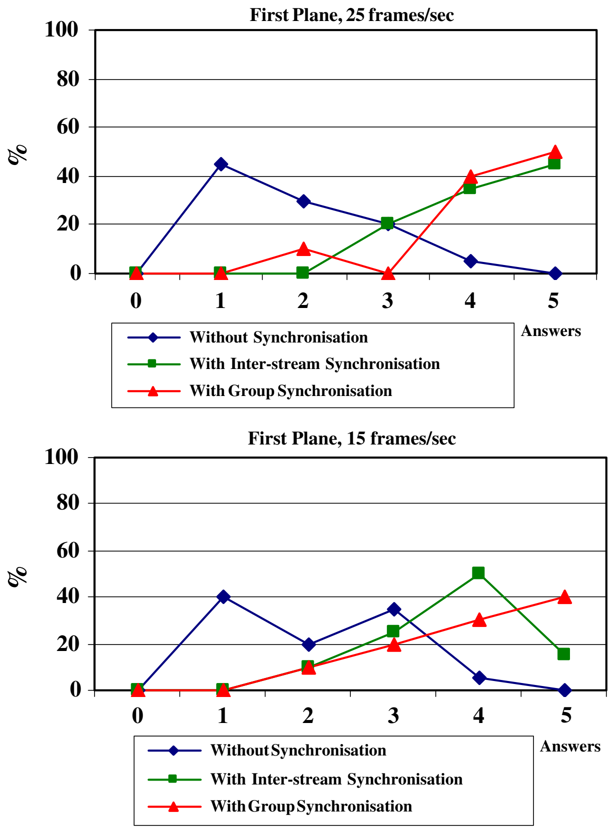


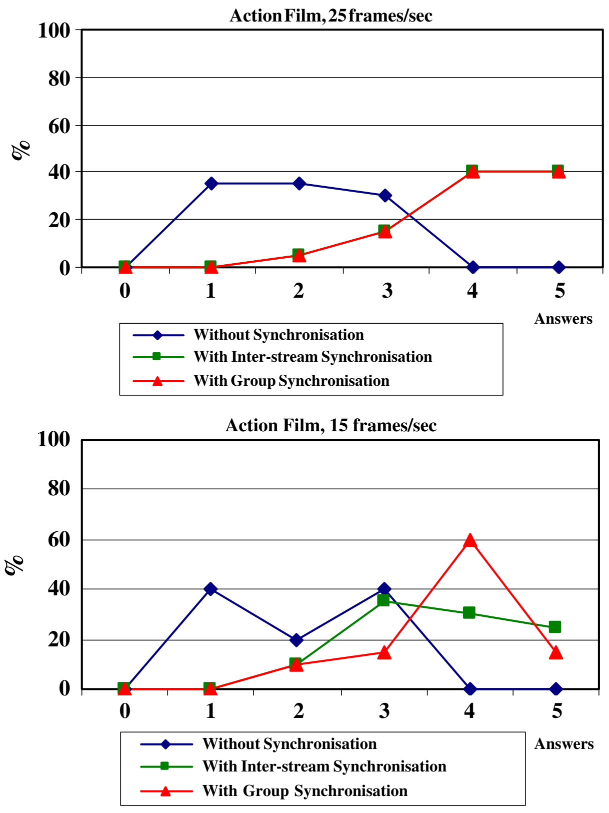
Fig. 21 Synchronization quality (first plane sequence)

Tests consisted of 3-min sessions with the two types of sequence and with separated video and audio streams. Before starting the tests, users were informed of the evaluation method and we explained the tests to them and how they had to fill in the questionnaires for each test.

Figures 21 and 22 show the results of the synchronization quality evaluation. The use of the group synchronization algorithm obtained very good marks, even better than the sequences with only local inter-stream synchronization.

Figures 23 and 24 present the degradation of synchronization perceived by the users in all the tests for both types of sequences and for both playout rates. In the sequences with group synchronization, they detected abnormal effects due to the synchronization processes but they evaluated them as imperceptible or not annoying. We can appreciate that in the first plane sequences, users could easily perceive when a situation of asynchrony between streams occurred. In this type of sequence, users could easily detect the absence of lip-sync because they were directly watching the speaker's face movements. In action film sequences, there were frequent changes of planes so the synchronization actions ('*skipping*' and/or '*pausing*' MUs) were more difficult to detect. Moreover, most of the subjects were used to watching TV foreign films which have been translated into their own languages and in those processes there are abnormal effects due to the asynchronies inherent in the dubbing process. We suppose users tolerated well the effects due to the asynchronies in these types of sequence because they were used to watching these effects almost every day and do not consider them as abnormal.

Fig. 22 Synchronization quality (action film sequence)



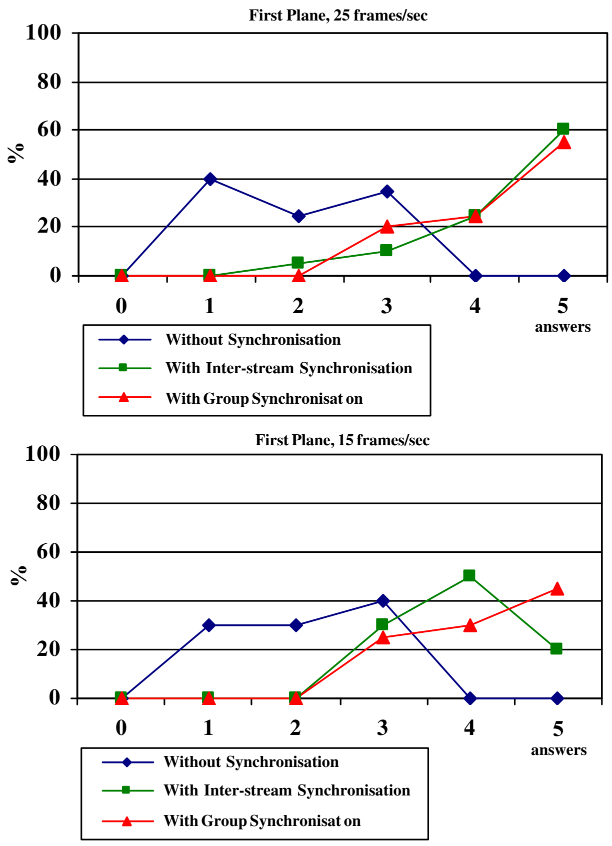


Fig. 23 Synchronization degradation (first plane sequence)

All of the figures show that playout rate did not significantly affect the appreciation of the synchronization quality by the users, in these cases. Nevertheless, as we expected, users detected more abnormal effects in sentences with rates of 15 frames/seconds, such as mechanical and artificial movements or '*skips*' between different images.

4 Conclusions and future work

A new type of multimedia synchronization called *Group Synchronization* has been addressed and a solution to achieve it has been presented. This has been done using standard well-known protocols and without defining a new protocol. Such synchronization is very important in lots of distributed multimedia applications in order to guarantee that all the receivers playout multiple media streams synchronously. Very few researchers, however, have considered this.

A modification of RTCP packets has been presented to facilitate synchronization in a simple manner. The solution is not based on a new protocol with specific packets for synchronization but instead uses standard protocols such as RTP/RTCP and global time protocols such as NTP or GPS. We propose modifying or extending the format of the RTCP RR packet and using common RTCP APP packets for synchronization, thus simplifying the implementation.

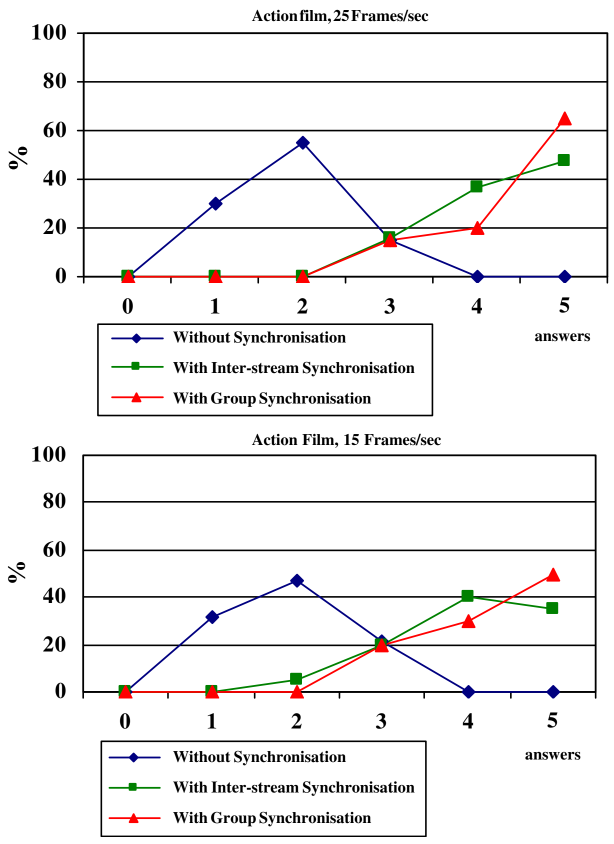


Fig. 24 Synchronization degradation (action film sequence)

Our solution takes advantage of feedback RTCP RR messages and the flexibility of RTP/RTCP to communicate the information required for synchronization. This modification results in only a small increase in network workload. Synchronization between receivers (distributed) and between streams (local) is maintained within limits that are in accordance with the results obtained by Steimetz in [28].

Our approach does not require the optimum transmission rate for feedback and control information to be calculated, as required by the solutions presented in [8, 9, 23–26], because we use RTCP RR feedback packets and extended RTCP APP packets. This approach has the benefit of avoiding a situation where all receivers send packets at the same time, overloading the network or the source. Instead, each receiver calculates the transmission time for RTCP packets according to RFC 3550, minimizing the effect on the scalability of the approach.

The solution is based on a dual master/slave scheme in which the master reference can be fixed statically or selected dynamically (if the master reference fails, another reference will be chosen), making the scheme fault tolerant.

Our solution satisfies the main aims set out at the beginning of the paper and good results have been achieved from our objective evaluation and subsequently corroborated by the subjective evaluation, in both LAN and WAN environments. These results validate our approach as a possible solution for multimedia applications which require group synchronization. The solution is valid both for the transmission of stored multimedia information as well as for live content (real-time) transmission.

Our future work will address the issues arising from bandwidth problems and the QoS feedback information provided by RTCP RR EXT packets. In this situation, the source may dynamically modify the transmission parameters (transmission rate, digital coding systems, etc.) to suit the current state of the network and improve the quality of the distributed multimedia system. Other future work will study whether it is necessary to send the extension proposed in all of the RTCP RR RTCP packets sent by the receivers or whether indications from the source can instruct receivers when RTCP RR EXT packets are required. This would further minimize the control load for synchronization, which is already very low without these improvements. Finally, we would like to combine our proposal with the use of agents for the purpose of providing multimedia synchronization, as in [18].

References

1. Akyildiz IF, Yen W (1996) Multimedia group synchronization protocols for integrated services networks. IEEE J Select Areas Commun 14:162–173
2. Boronat F (2004) Specification and evaluation of a Multimedia group synchronization algorithm, PhD Thesis, Polytechnic University of Valencia (UPV), Spain. April
3. Boronat F, Guerri JC (2005) Analysis and comparison of multimedia inter-stream and group synchronisation algorithms. IEEE Lat Am Trans 3(5), Dic.
4. Boukerche A, Owens H (2005) Media synchronization and qos packet scheduling algorithms for wireless systems. Mob Netw Appl 10(1–2):233–249
5. Burmeister C, Hakenberg R, Miyazaki A, Ott J, Sato N, Fukunaga S (2006) Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback: results of the Timing Rule Simulations. RFC 4586, Network Working Group, July
6. Diot C, Gautier L (1999) A distributed architecture for multiplayer interactive applications on the internet. IEEE Netw 13:6–15
7. Gili J, Roser M, Vicente A (1995) MPEG video coding algorithm: image quality subjective assessment. Telef R&D Commun 6(1):20–31
8. Guerri JC (1997) Specification and performance evaluation of an adaptive multimedia stream synchronization protocol based on global time and feedback techniques, with stream control Thesis, U.P.V. June
9. Guerri JC, Esteve M, Palau CE, Casares V (2001) Feedback flow control with hysteresial techniques for multimedia retrievals. Multimedia Tools and Applications 13(3):307–332
10. Ishibashi Y, Tasaka S, Miyamoto H (2002) Joint synchronization between stored media with interactive control and live media in multicast communications. Multimedia Syst E85-B(4):812
11. Ishibashi Y, Hasegawa T, Tasaka S (2004) Group synchronization control for Haptic Media in Networked Virtual Environments, 12th International Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2004, HAPTICS '04 Proceedings, ISBN 0-7695-212-6, pp. 106–113.
12. ITU-R Rec. BT. 500 (2002) Methodology for the subjective assessment of the quality of television pictures. Geneva, June
13. Kaneoka H, Ishibashi Y (2004) Effects of group synchronization control over haptic media in collaborative work. In: Proc. the 14th international conference on artificial reality and telexistence (ICAT'04), pp. 138–145, Nov/Dec
14. Kouvelas I, Hardman V (1997) Overcoming workstation scheduling problems in a real-time audio tool. In: Proc. USENIX, Anaheim, CA, pp. 235–242, Jan
15. Kouvelas Y, Hardman V, Watson A (1996) Lip Synchronization for use over the Internet: analysis and Implementation. Technical report, Department of Computer Science, University College London
16. Kutscher D, Ott J (2000) The Message Bus. A Communication - integration Infrastructure for Component-based Systems, White paper, January 2000. http://www.mbus.org/mbuswp.pdf
17. Laoutaris N, Stavrakakis I (2002) Intrastream synchronization for continuous media streams: a survey of playout schedulers. IEEE Netw Mag 16(3):30–40
18. Manvi SS, Venkataram P (2006) An agent based synchronization scheme for multimedia applications. J Syst Softw (JSS) 79(5):701–713

19. McCanne S, Jacobson V (1995) Vic: A flexible framework for packet video. ACM Multimedia, San Francisco, CA, pp 511–522
20. Mills DL (1991) Internet time synchronization the network time protocol. IEEE Trans Commun 39(10):1482–1493
21. Ott J, Wenger S, Sato N, Burmeister C, Rey J (2006) Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF), RFC 4585, Network Working Group, July
22. Roccetti M, Ghini V, Pau G, Salomoni P, Bonfigli ME (2001) Design and experimental evaluation of an adaptive playout delay control mechanism for packetized audio for use over the internet. Multimedia Tools and Applications 14(1):23–53
23. Ramanathan S, Rangan PV (1993) Feedback techniques for intra-media continuity and inter-media synchronization in distributed multimedia systems. Comput J 36(1):19–31
24. Rangan PV, Ramanathan S (1992) Designing an on-demand multimedia service. IEEE Commun Mag 30(7):56–64
25. Rangan PV, Ramanathan S (1995) Performance of inter-media synchronization in distributed and heterogeneous multimedia systems, 1993. Comput Netw ISDN Syst 27(4):549–565
26. Rangan PV, Ramanathan S, Sampathkumar S (1995) Feedback techniques for continuity and synchronization in multimedia information retrieval. ACM Trans Inf Syst (TOIS) 13(2):145–176
27. Schulzrinne H, Casner S, Frederick R, Jacobson V (2003) RTP: a transport protocol for real-time applications. RFC 3550, July
28. Steinmetz R (1996) Human perception of jitter and media skew. IEEE J Select Areas Commun 14(1): 61–72
29. The Spanish Academic and Research network, Red IRIS. http://www.rediris.es/
30. Yavatkar R, Lakshman K (1994) Communication support for distributed collaborative applications. Multimedia Syst 2(4):74–88

Dr. Fernando Boronat Seguí was born in Gandia, (Spain) and went to the Polytechnic University of Valencia (UPV) in Spain, where he obtained, in 1993, his M.Sc. in Telecommunications Engineering. In 1994 he worked for a couple of years for Telecommunication Companies before moving back to the UPV in 1996 where he is Lecturer in the Communications Department at the Escuela Politécnica Superior de Gandia. He obtained his PhD degree in 2004 and his topics of interest are Communication networks, Multimedia Systems and Multimedia Synchronization Protocols. He is IEEE member since 1993 and is involved in several IPCs of national and international conferences.

Dr. Juan Carlos Guerri Cebollada obtained PhD degree in 1997 and is Lecturer at UPV and he also is the person responsible for the Multimedia Communications Research Group, included in the Instituto de Telecomunicaciones y Aplicaciones Multimedia (iTEAM) at the UPV. He is involved in several IPCs of national and international conferences.

Dr. Jaime Lloret Mauri received his M.Sc. in Physics in 1997, his M.Sc. in Electronic Engineering in 2003 at University of Valencia (Spain) and his Ph.D. in telecommunication engineering from the UPV in 2006. He is a Cisco Certified Network Professional Instructor and he also teaches in the EPSG at the UPV. He has been working as a network administrator in several companies. Nowadays he is researching on P2P Networks and on sensor Networks. He is a member of IASTED, and is involved in several IPCs of national and international conferences.