Multimedia Tools and Applications, 14, 23–53, 2001

Design and Experimental Evaluation of an Adaptive Playout Delay Control Mechanism for Packetized Audio for Use over the Internet

MARCO ROCCETTI roccetti@cs.unibo.it
VITTORIO GHINI
GIOVANNI PAU
PAOLA SALOMONI
MARIA ELENA BONFIGLI
Università di Bologna, Dipartimento di Scienze dell'Informazione Via Mura Anteo Zamboni 7, 40127 Bologna, Italy

Abstract. We describe the design and the experimental evaluation of a playout delay control mechanism we have developed in order to support unicast, voice-based audio communications over the Internet. The proposed mechanism was designed to dynamically adjust the talkspurt playout delays to the traffic conditions of the underlying network without assuming either the existence of an external mechanism for maintaining an accurate clock synchronization between the sender and the receiver during the audio communication, or a specific distribution of the audio packet transmission delays. Performance figures derived from several experiments are reported that illustrate the adequacy of the proposed mechanism in dynamically adjusting the audio packet playout delay to the network traffic conditions while maintaining a small percentage of packet loss.

Keywords: multimedia applications, packetized audio, playout delay control, packet loss, Internet

1. Introduction

Sophisticated applications of Internet multimedia conferencing will become increasingly important only if the quality of the communications will be perceived as sufficiently good by their users. The result of extensive experiments has shown that audio is frequently perceived as one of the most important component of multimedia conferencing [5]. A number of problems have been identified which negatively impacts the quality of audio conversations, but probably the more critical one with audio is the loss of audio packets. Basically, two are the main causes for audio packet loss over wide-area packet-switched networks: 1) traffic congestion at the interconnecting routers that cause audio packets to be discarded, and 2) too large transmission delays that cause audio packets to arrive at the destination past the time instant at which they are scheduled to be played out (the playout point).

With the term *playout delay* we refer to the total amount of time that is experienced by the audio packets of a given talkspurt from the time instant they are generated at the source and the time instant they are played out at the destination. Summarizing, such a playout delay consists of: i) the "collection" time needed for the transmitter to collect audio samples and to prepare them for transmission, ii) the "transmission" time needed for the transmission of audio packets from the source to the destination over the underlying transport network, and finally iii) the “buffering” time, that is the amount of time that a packet spends queued in the destination buffer before it is played out. A crucial tradeoff exists between audio packet playout delay and audio packet loss: the longer the scheduled playout delay, the more likely it is that an audio packet will arrive at the destination before its scheduled playout deadline has expired. However, if on one side a too large percentage of audio packet loss (over 5–10%) may impair the intelligibility of an audio transmission, on the other side, too large playout delays (e.g., more than 200–250 msec) may disrupt the interactivity of an audio conversation [13].

The main purpose of this paper is to describe a playout delay control mechanism that is suitable for adjusting the talkspurt playout delays of unicast, voice-based audio communications across the Internet. The mechanism was designed to dynamically adjust the talkspurt playout delays to the network traffic conditions without assuming either the existence of an external mechanism for maintaining an accurate clock synchronization between the sender and the receiver, or a specific distribution of the end-to-end transmission delays experienced by the audio packets. Succinctly, the technique for dynamically adjusting the talkspurt playout delay is based on obtaining, in periodic intervals, an estimation of the upper bound for the packet transmission delays experienced during an audio communication. Such an upper bound is periodically computed using round trip time values obtained from packet exchanges of a three-way handshake protocol performed between the sender and the receiver of the audio communication. At the end of such protocol exchange, the receiver is provided with the sender’s estimate of an upper bound for the transmission delay that can be used in order to dynamically adjust the talkspurt playout delay. The proposed mechanism guarantees that the talkspurt playout delay may be dynamically set from one talkspurt to the next, without causing *gaps* or *time collisions* (formally defined in the Sections 3.1.1 and 3.1.2) inside the talkspurts themselves, provided that intervening silence periods of sufficiently long duration are exploited for the adjustment.

The need of silent intervals for allowing the mechanism to adjust to the fluctuating network conditions is common to the most part of the existing audio tools (e.g., NeVoT, vat and rat [5, 8, 18]) but renders the proposed scheme particularly relevant for voice-based applications where conversational audio with intervening silence periods between subsequent talkspurts is transmitted.

The design of our mechanism was completed during the Summer of 1997. A prototype version of the mechanism running on workstations equipped with the SunOS 4.3 (BSD Unix) operating system, and based on the datagram based UDP protocol was soon carried out. Based on that prototype implementation, several experiments were conducted over an (IP based) internetworked connection between the University of Bologna (Italy) and the C.E.R.N. Institute in Geneva (Switzerland). The performance figures derived by the experimentations conducted on the field illustrated the adequacy of our mechanism in dynamically adjusting the audio packet playout delay to the traffic conditions of the underlying network while maintaining a small percentage of packet loss.

The paper is structured as follows. In the next section, we discuss some background issues that we have regarded as important for the design of our mechanism. The remaining Sections (3 and 4) describe and discuss: i) the proposed playout delay control mechanism, ii) a prototype implementation we have developed, iii) the results of an experimental assessment we have carried out, and finally iv) a performance comparison between our mechanism and another adaptive playout delay adjustment mechanism recently proposed [13].

We conclude this introduction by summarizing the main features of our playout delay control mechanism. It provides: 1) an embedded and accurate algorithm that maintains tight time synchronization between the sender's system clock and the clock that supports the playout process at the receiving host, during an audio conversation; 2) a method for adaptively estimating the audio packet playout time (on a per-talkspurt basis) with an associated minimal computational overhead for both the source and the destination hosts, and 3) an exact and simple method for dimensioning the playout buffer depending on the network traffic conditions.

2. Packetized audio over the Internet

Since the early experiments with packetized voice in the Arpanet network [3], packetized audio applications have become sophisticated tools that many Internet users try to use with regularity. For example, the audio conversations of many international conferences and workshops are now usually conducted over the Mbone (the multicast backbone), an experimental overlay network of the Internet [11]. The audio tools that are used to transmit packet audio over the Internet (e.g., NeVot [18], vat [8], rat [5], the INRIA audio tool [2]) typically operate by periodically sampling audio streams generated at the sending host, packetizing them, and transmitting the obtained packets to the receiving site by using datagram based connections (e.g., UDP). In addition, at the receiving site, packets are buffered and their playout time is delayed in order to compensate for variable network delays that may be frequently experienced. Such playout mechanisms try to adaptively adjust the playout delay in order to keep this delay as small as possible while minimizing the number of packets that arrive too late (i.e., after their playout point). The next section provides additional information that constitute the background of the algorithm to be presented in this paper. In particular, the main characteristics of the mechanisms that are used to adaptively adjust the playout time for audio packets over the Internet are reviewed.

2.1. Background

A typical audio segment may be considered as constituted of *talkspurt* periods during which the audio activity is carried out, and *silence* periods during which no audio packet is generated. In order for the receiving site to reconstruct the audio conversation, the audio packets constituting a talkspurt must be played out in the order they were emitted at the sending site. If the delay between the arrival of subsequent packets is constant (i.e., the underlying transport network is jitter-free) a receiving site may simply play out the arriving audio packets as soon as they are received. Unfortunately, this is only rarely the case, since jitter-free, ordered, on-time packet delivery almost never occurs in today's packet-switched networks. Those variations in the arrivals of subsequent packets strongly depend on the traffic conditions of the underlying network. Packet loss percentages (due to the effective loss and damage of packets as well as late arrivals) often vary between 15% and 40% [13]. In addition, extensive experiments with wide-area network testbeds have shown that the

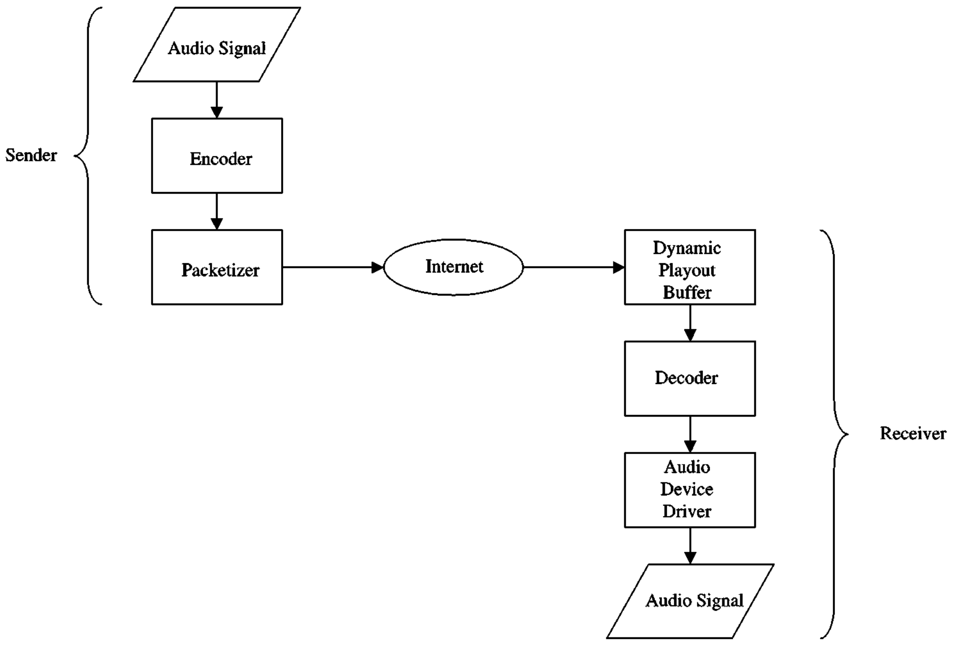

Figure 1. Audio data flow over the Internet.

delays between consecutive packets may also be as much as 1.5 seconds, thus impairing real-time interactive human conversations.

New protocol suites such as the Resource Reservation Protocol (RSVP) [21] might eventually ameliorate the effect of jitter and improve the quality of the audio service over the Internet, but they are not yet widely used. On the other hand, the most used approach is to adapt the applications to the jitter present on the network. Hence, to transport audio over a non-guaranteed packet-switched network, audio samples are encoded (usually with some form of compression), inserted into packets that have creation timestamps and sequence numbers, transported by the network, received in a playout buffer, decoded in sequential order, and finally played out by the audio device, as seen in figure 1. A symmetric scheme is used in the other direction for interactive conversation. The *smoothing* playout buffer is used at the receiver in order to compensate for variable network delays. Received audio packets are queued into the buffer, and the playout of each packet of a given talkspurt is delayed for some quantity of time beyond the reception of the first packet of that talkspurt. In this way, dynamic playout buffers can hide, at the receiver, packet delay variance at the cost of additional delay. A crucial tradeoff exists between the length of the imposed additional quantity of delay and the amount of lost packets due to their late arrival: the longer the additional delay, the more likely it is that a packet will arrive before its scheduled playout deadline. However, too long playout delays may in turn seriously compromise the quality of the conversation over the network.

Typical acceptable values for the end-to-end delay between packet audio generation at the sending site and its playout time at the receiver are below the threshold of 200–250 msec, furthermore a percentage of no more than 5–10% of packet loss is considered quite tolerable in human conversations [2].

Besides adjusting the audio playout delay in order to compensate for the effect of the jitter, modern audio tools typically make also use of error and rate control mechanisms based on a technique known as forward error correction (FEC) to reconstruct many lost audio packets [2]. For example, the INRIA audio tool adjusts the audio packet send rate to the current network conditions, adds redundant information to packets (under the form of highly compressed versions of a number of previous packets) when the loss rate surpasses a certain threshold, and establishes a feedback channel to control the send rate and the redundant information. Simply put, the complete process is controlled by an open feedback loop that selects among different available compression schemes and the amount of redundancy needed, as described in the following. If the network load and the packet loss are high, the amount of compressed redundant information carried in each packet is increased by adding to each packet compressed version of the previous two to four audio packets. In 5-seconds intervals the receiver returns (using the Real Time Protocol suite RTP-RTCP [19]) quality of service reports to the sender in order to regulate and adapt the quantity of redundant information being sent.

As discussed above, efficient playout adjustment mechanisms have been developed to minimize the effect of delay jitter. Typically, a receiving site in an audio application buffers packets and delays their playout time. Such a playout delay may be kept constant for the duration of the audio conversation, or dynamically adjusted from one talkspurt to the next. Due to the fluctuating end-to-end (application-to-application) delays experienced over the Internet, constant, non-adaptive playout delays may result in unsatisfactory quality for audio applications. Hence, two are the approaches widely exploited for adaptively adjusting playout time: the former approach keeps the same playout delay constant throughout a given talkspurt, but permits different playout delays in different talkspurts. In the latter approach, instead, the playout delay is adjusted on a per-packet basis. However, an adaptive adjustment on a per-packet basis may introduce gaps inside talkspurt and thus is considered as of being damaging to the perceived audio quality. On the contrary, the variation of the playout delay from a talkspurt to the next may introduce artificially elongated or reduced silence periods, but this is considered acceptable in the perceived speech if those variations are reasonably limited. Hence, the totality of the above mentioned tools adopt a mechanism for adaptively adjusting the playout delays on a per-talkspurt basis. However, in order to implement such a playout control mechanism, almost all the above cited audio applications make use of the following two strong assumptions.

1. An external mechanism exists that keeps synchronized the two system clocks at both the sending and the receiving site. Usually, the IP-based Network Time Protocol (NTP) is used for this purpose.
2. The delays experienced by audio packets on the network follow a Gaussian distribution.

Based on the above assumptions, the playout control mechanism works as described in the remainder of this section [20]. Let us denote with:

- g_i the time instant in which the audio packet i is generated at the sending site,
- a_i the time instant in which the audio packet i is delivered at the receiving site,

- p_i the time instant in which the audio packet i is played out at the receiving site,
- n_i the end-to-end transmission delay, i.e., $n_i = a_i - g_i$,
- b_i the introduced buffering delay at the receiving site,
- d_i the playout delay, i.e., the time interval between the generation of the audio packet at the sender and the time instant in which the packet is played out at the receiver, i.e., $d_i = n_i + b_i$,
- $\hat{d}_i$ the average playout delay,
- $\hat{v}_i$ the variation of the average playout delay.

If i is the first packet of a given talkspurt, then the playout time p_i for that packet is usually calculated as [14]:

$$p_i = g_i + \hat{d}_i + s \times \hat{v}_i,$$

where the constant s usually ranges in the interval $[0, 4]$, with typical values set equal to $0.5, 2, 4$, since the correspondent multiplications are easily implemented with *shift* operations. From an intuitive standpoint, the reported formula (and the $s \times \hat{v}_i$ term) is used to set the playout time to be far enough beyond the average delay estimate, so that only a small fraction of the arriving packets should be lost due to late arrivals. The playout point for any subsequent packet j of that talkspurt is computed as an offset from the point in time when the first packet i in the talkspurt was played out: $p_j = p_i + t_j - t_i$.

The estimation of both the average delay and the average delay variation are carried out using the well known *stochastic gradient algorithm* [14] by using the following two formulas:

$$\hat{d}_i = a \times \hat{d}_{i-1} + (1 - a) \times n_i, \quad \hat{v}_i = a \times \hat{v}_{i-1} + (1 - a) \times |\hat{d}_i - n_i|,$$

where the constant a (usually equal to $7/8$) is a weight that characterizes the "memory properties" of the estimation.

Extensive experiments have been carried out that have shown that the playout delay control mechanisms based on the formulas above may be adequate to obtain acceptable values for the tradeoff between the average playout delay and the loss due to late packet arrivals. However, in some circumstances, the cited mechanisms may suffer from a number of problems, especially when they are deployed over wide-area networks. In particular, the following problems may be pointed out [13, 20]:

- The "external" software-based mechanisms (e.g., the NTP protocol) used to maintain the system clocks synchronized at both the sending and the receiving sites are not typically widespread all over the Internet. In addition, those mechanisms may turn out to be too much inaccurate to cope with the real-time nature of the audio generation/playout process. For example, even if the NTP protocol may achieve computer clock synchronization within a few tens of milliseconds over most paths in the Internet of today, however, there may be frequent exceptions with synchronization values up to a few hundreds of milliseconds, especially if a client host is not directly connected to a primary server of the NTP hierarchy but achieves synchronization through a stratum-2 (or higher) server via a congested link [12]. The problem with clock synchronization is that if the two different clocks (respectively, at the source and at the destination) do not run at the same rate and the synchronization mechanism is not sufficiently accurate, they will tend to drift further and further apart. Extensive experiments have shown that the above mentioned behavior may have a very negative impact on the provided formulas for the calculation of the playout time, thus resulting in an increased number of lost packets [20].

- The widely adopted assumption that the packet transmission delays over the Internet follow a Gaussian distribution seems to be a plausible conjecture only for those limited time intervals in which the overall load of the underlying network is quite light. Indeed, recent experimental studies carried out over the Internet have indicated the presence of frequent and conspicuously large end-to-end delay spikes for periodically generated packets (as is the case with audio packets) [2, 10].
- Several experiments have been conducted [20] that show that choosing a value equal to 4 for the constant s in the calculation of the formula of the playout delay typically results in a quite small quantity of loss packets (approx. 4%), but also in a quite large average playout delay $\hat{d}_i$ usually equal to 2 times the transmission time n_i.

3. A novel mechanism for packetized audio

The adaptive mechanism for the control of the playout delay proposed in this section ameliorates all the negative effects of the audio tools reported above, while maintaining satisfiable values of both the average playout delay and the packet loss due to late arrivals. The proposed policy assumes neither the existence of an external mechanism for maintaining an accurate synchronization at both the sending and the receiving sites, nor a Gaussian distribution for the end-to-end transmission delays of the audio packets. In particular, it provides:

- an internal and accurate mechanism that maintains tight time synchronization between the system clocks of both the sending and the receiving hosts;
- a method for adaptively estimating the audio packet playout time (on a per-talkspurt basis) with an associated minimal computational overhead;
- an exact and simple technique for dimensioning the playout buffer depending on the traffic conditions of the underlying network.

In the following, a description of the main ideas behind the mechanism is provided. This proposed mechanism has been also subject to a simulated performance study, whose results are described in [1, 16]. For continuous playout of audio packets at the receiving site, it is essential that the audio packets be available at the receiver prior to their respective playout time and that the rate of consumption (i.e., playout) of packets at the receiver meets the rate of transmission at the sender [15]. Hence, when the sender transmits the first packet of an audio talkspurt, it timestamps that packet with the value (say C) of the reading of its own clock. As soon as this first packet arrives at the receiver, it sets the clock that supports the playout process (say C_R) by using the C value, i.e.,

$$C_R = C,$$

and immediately schedules the presentation of that first packet. Subsequent audio packets belonging to the same talkspurt are also timestamped at the sender with the value of the reading of the sender's clock at the time instants when the packets are transmitted. When these subsequent packets arrive at the receiving site, their attached timestamp is compared with the value of the reading of clock that supports the playout process at the receiving host (the receiver's clock, for short). If the timestamp attached to the packet is equal to the value of the receiver's clock, that packet is immediately played out. If the timestamp attached to the packet is larger than the value of the receiver's clock, that packet is buffered and its playout time is scheduled after a time interval equal to the positive difference between the value of the timestamp and the actual value of the receiver's clock. Finally, if the timestamp attached to the packet is smaller than the value of the receiver's clock, the packet is simply discarded since it is too late for presentation.

However, (even if an identical clock rate is assumed) due to the fluctuating delays in real transmissions, the values of the clocks of the sender and of the receiver may differ, at a given time instant, by the following quantity

$$C_S(T) - C_R(T) = \Delta,$$

where $C_S(T)$ and $C_R(T)$ are, respectively, the readings of the local clocks at the sender and at the receiver (at the same time instant T), and Δ is a non negative quantity ranging between 0 (a theoretical lower bound) and Δ_{max} (a theoretical upper bound on the transmission delay introduced by the network between the sender and the receiver). Later, we will show a technique for the estimation of Δ, and how this value impacts the calculation of the playout time for the audio packets.

A crucial issue of the mechanism is an accurate dimensioning of the playout buffer for audio packets. Both buffer underflow and overflow may occur, thus resulting in discontinuities in the playout process. In order to master all the possible problems deriving from both buffer underflow and overflow, a simple technique is proposed that was first used in [15] in order to guarantee the continuity of the playout process controlled by an MPEG codec for video frames. Such a technique accurately distinguishes among the two possible cases of buffer underflow and overflow. The worst case scenario for buffer underflow (corresponding to the case when packets arrive too late for presentation) is clearly when the first packet arrives after a minimum delay (e.g., 0), while subsequent packets arrives with maximum delay (e.g., Δ_{max}). In this case, due to the minimum delay of the first packet, there is a null difference between the clock at the sender and at the receiver

$$C_S(T) - C_R(T) = 0.$$

However, consider now a situation in which a subsequent packet arrives at the receiver that suffers from the maximum delay Δ_{max}. Suppose that this packet has been transmitted by the sender when its clock shows a time value equal to say X. Due to the imposed

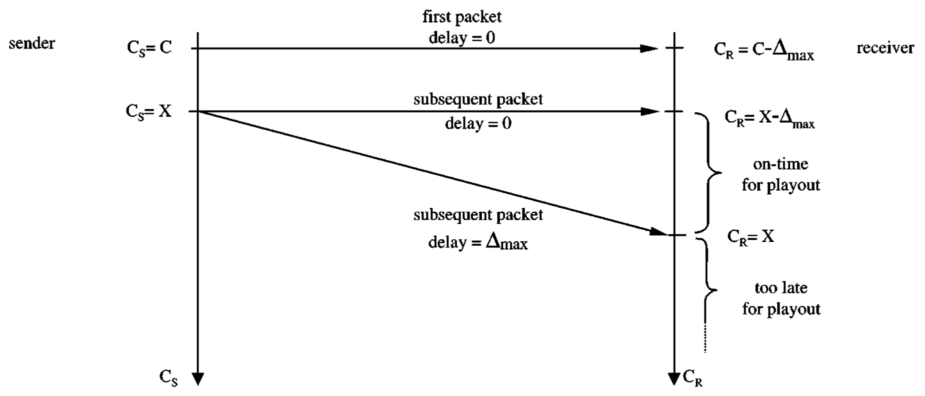

Figure 2. Delayed setting of the receiver’s clock for preventing buffer underflow.

synchronization, at that precise instant also the receiver’s clock would show a value equal to X. Now, adding the transmission delay of Δ_{max}, the arrival time of this subsequent packet occurs when the receiver’s clock shows the value given by $X + \Delta_{max}$. Unfortunately, that packet would be too late for playout and consequently discarded. This example suggests that a practical and secure method for preventing buffer underflow (i.e., packets lost due to their late arrival) is that the receiver delays the setting of its local clock of an additional quantity equal to Δ_{max}, when the first packet of the talkspurt is received. Precisely, when the first packet is received with its timestamp equal to C, the receiver sets its local clock to a value equal to $C - \Delta_{max}$:

$$C_R(T) = C - \Delta_{max}.$$

With this simple modification (see figure 2) the problem of buffer underflow gets solved. Simply put, this policy implicitly guarantees that all the audio packets that will suffer from a transmission delay not greater than Δ_{max} will be on-time for the playout.

However, the above mentioned technique introduces another problem: that of playout buffer overflow. The worst case scenario for buffer overflow occurs in the following circumstance: the first packet of a talkspurt suffers from the maximum delay Δ_{max}, instead a subsequent audio packet experiences the minimum delay 0. At the arrival of the first packet of the talkspurt at the receiving site, the receiver sets its clock equal to the value timestamped in the packet (say C) only after Δ_{max} time units since the packet is arrived. Due to this setting and to the maximum delay experienced by the first packet of the talkspurt the time difference between the two clocks at the sender and at the receiver at a given time instant T is equal to

$$C_S(T) - C_R(T) = 2 \times \Delta_{max}.$$

Now, if a subsequent packet arrives at the receiver that has experienced only a minimum delay equal to 0, then the receiver’s clock, upon the reception of that packet, shows a time

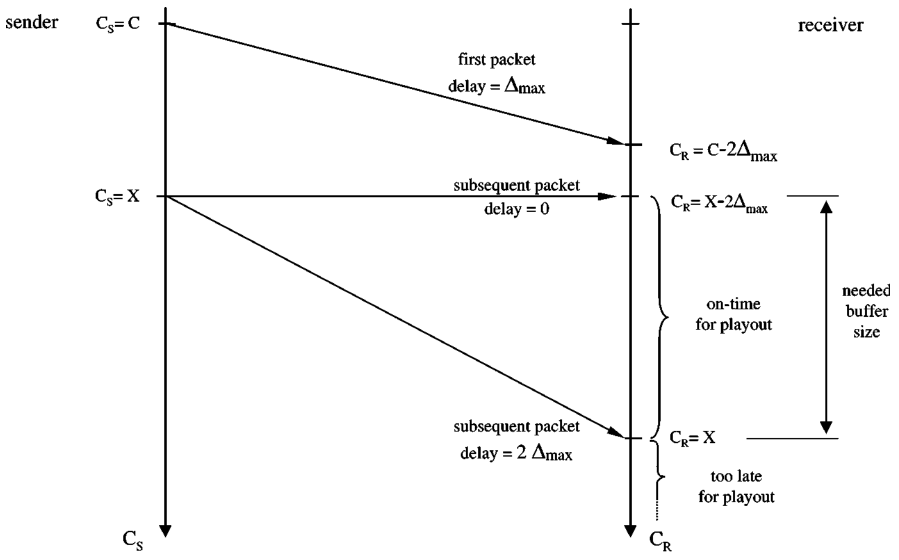

Figure 3. Additional buffering required for preventing buffer overflow.

value equal to

$$C_R(T) = C - 2 \times \Delta_{max},$$

where C is the timestamp attached to the first packet of the talkspurt. From the formula above it is clear that, in order for each packet with an early arrival to have room in the playout buffer, an additional buffering space is required at the receiving site equal to the maximum number of audio packets that might arrive in a time interval of $2 \times \Delta_{max}$ (see figure 3). In conclusion, the example above dictates that the playout buffer dimension may never be less than the maximum number of packets that may arrive in an interval of $2 \times \Delta_{max}$.

Nevertheless, two problems have been left unresolved:

1. The accurate estimation of the value Δ of the difference between the sender's and the receiver's clock at a given time instant T, and
2. The dynamical adaptation of the proposed playout control mechanism in order to compensate for the highly fluctuating end-to-end transmission delays that may be experienced over wide-area packet-switched networks such as the Internet.

The following section is devoted to describe a technique that may be used to evaluate an upper bound on the maximum experienced delay, and also to adapt the proposed playout control mechanism to the highly fluctuating transmission delays of wide-area packet-switched networks.

3.1. Adaptive adjustment of the mechanism

A simple technique may be devised to estimate an upper bound for the maximum transmission delay. This technique exploits the so called Round Trip Time (*RTT*) and is based on a three-way handshake protocol. It works as follows. Prior to the beginning of the first talkspurt in an audio conversation, a *probe* packet is sent from the sender to the receiver timestamped with the clock value of the sender (say C). At the reception of this probe packet, the receiver sets its own clock with the value of the timestamp attached to the probe packet, and sends immediately back to the sender a *response* packet with the same timestamp C. Upon the reception of this response packet, the sender computes the value of the *RTT* by subtracting from the current value of its local clock the value of the timestamp C. At that moment, the difference between the two clocks, respectively at the sender and at the receiver, is equal to an unknown quantity (say t_0) which may range from a theoretical lower bound of 0 (that is, all the *RTT* value has been consumed on the way back from the receiver to the sender), and a theoretical upper bound of *RTT* (that is all the *RTT* has been consumed on the way in during the transmission of the probe packet). Unfortunately, a time difference of only t_0 between the sender's and the eceiver's clocks could not be sufficient to prevent packet loss due to late arrivals, as well as a rough approximation of this value (e.g., $t_0 = RTT/2$) might result in both playout buffer underflow problems and packet loss due to primature arrivals. Based on these considerations, the sender, after having received the response packet from the receiver and having calculated the *RTT* value, sends to the receiver a final *installation* packet, with piggybacked on it the previously calculated *RTT* value. Upon receiving this installation packet, the receiver sets the time of its local clock by subtracting from the value shown at its clock the value of the transmitted RTT. Hence, at that precise moment, the difference between the two clocks at the receiver and at the sender is equal to a value given by

$$\Delta = C_S(T) - C_R(T) = t_0 + RTT,$$

where Δ ranges in the interval $[RTT, 2 \times RTT]$, depending on the unknown value of t_0, that in turn may range in the interval $[0, RTT]$. In essence, with the strategy above, a maximum transmission delay equal to Δ is left to the audio packets to arrive at the receiver in time for playout, and consequently a playout buffering space proportional to Δ is required for packets with early arrivals. In order for the proposed policy to adaptively adjust to the fluctuating network delays experienced over the Internet, the above mentioned *synchronization* technique is first carried out prior to the beginning of the first talkspurt of the audio conversation, and then periodically repeated throughout the entire conversation. The adopted period is about 1 second in order to prevent the two clocks (possibly equipped with different clock rates) from drifting apart. Thus, each time a new *RTT* value is computed by the sender, it may be used by the receiver for dynamically setting both the value of its local clock and the playout buffer dimensions. This method guarantees that both the introduced additional buffering delay and the buffer dimension are always proportioned to the traffic conditions. However, it may be not possible to replace on-the-fly during a talkspurt the current values of the receiver's clock and the dimensions of its playout buffer. In fact, as earlier mentioned, such an instantaneous adaptive adjustment of the *synchronization* parameters might introduce either *gaps* or even *time collisions* inside a talkspurt. (For a formal definition of gaps and time collisions see below in Sections 3.1.1 and 3.1.2). Based on this consideration, the installation at the receiver of the values of a new synchronization (namely, the activity of changing the values of the receiver's playout clock and of the buffer dimension) is carried out only during the periods of audio inactivity, when no audio packets are generated by the sender (i.e., during silence periods between different talkspurts).

The main purpose of the following two sections is to show how the installation of a new synchronization between the sender and the receiver may be conducted during a silence period detected by the sender without introducing either gaps or time collisions inside the talkspurts of the audio conversation.

3.1.1. Installing a new synchronization without introducing gaps. With the term *gaps* inside a talkspurt we refer to those cases when a given sequence of audio packets is artificially contracted (or truncated) by the playout control mechanism thus causing at the receiver an arbitrary skipping of a number of consecutive audio samples. For example, suppose that a given sequence of consecutive audio packets may be numbered with consecutive integers ranging from 1 to 100, a gap would be introduced by the playout control mechanism if that original packet sequence would be perceived at the receiver as composed by the packets numbered from 1 to 36 and then, immediately after, by the packets numbered from 60 to 100, thus excluding all the packets from 37 to 59.

We denote respectively with $\Delta_i = t_i + RTT_i$ and $\Delta_j = t_j + RTT_j$ the values of the differences between the sender's and the receiver's clocks due to two subsequent synchronizations i and j, where the synchronization j occurs after the synchronization i (i.e., $j > i$). We denote with δ_{sync} the value given by $\delta_{sync} = \Delta_j - \Delta_i$. Note that even if the values of t_i and t_j are unknown, instead their difference (i.e., $t_j - t_i$) may be exactly evaluated at the receiver's site upon the reception of the correspondent probe packets, by using, for example, the method that was first proposed in [4].

Possible gaps in the talkspurt (i.e., an arbitrary skipping of audio packets) may occur in the situation when δ_{sync} is smaller than 0. This situation really corresponds to an improvement of the traffic conditions of the underlying network since the value Δ_j turns out to be smaller than the value Δ_i. Because of this improvement in the traffic conditions, the installation of the synchronization j causes the advance of the receiver's clock from its current value (say I) to a larger value $I_* = I - \delta_{sync}$, where $I_* > I$. Hence, in order for the receiver to playout all the audio packets generated by the sending site without skipping any audio sample, it is necessary that the sender does not generate audio packets whose attached timestamps range in the interval given by $[I, I - \delta_{sync} - 1]$. Otherwise, the improvement in traffic conditions could cause possible negative scenarios, like that depicted in figure 4. In figure 4, a situation is represented where an improvement of the traffic conditions is experienced ($\Delta_j = 2 < \Delta_i = 25$, $\delta_{sync} = -23$). Owing to this improvement in the network traffic conditions, a new synchronization j could be installed (rightmost arrow in the figure) that would cause the advance of the receiver's clock from its previous value $I = 37$ to the new value given by $I_* = 60$. Unfortunately, this instantaneous increase in the receiver's clock would make impossible to play out at the receiver all the audio samples in the interval $[37, 59]$.

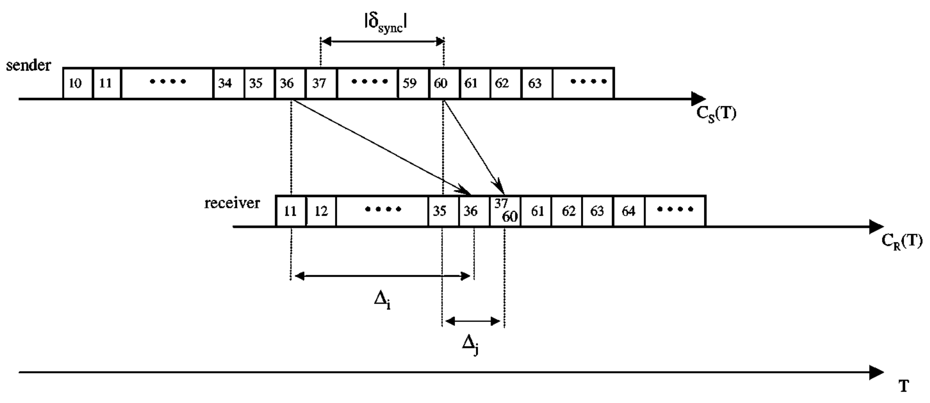

Figure 4. $\delta_{sync} < 0$: gaps at the receiver (from 37 to 59) due to the installation of a new synchronization.

It is possible to avoid the negative scenario described in figure 4 by adopting the following policy. Consider a situation during an audio conversation in which a previous synchronization (denoted with i) has been installed. All the playout activities concerning that audio conversation are carried out using the receiver's clock (say $C_{R,i}$) initialized with the corresponding values of the synchronization i. Suppose that subsequently all the preliminary activities related to a new synchronization (say j) are successfully performed: i.e., the probe and the response packets of the synchronization j are transmitted and received, and at the receiving site a new "spare" clock say $C_{R,j}$ is initialized accordingly. Consequently, at the sender site, the value of $\delta_{sync} = \Delta_j - \Delta_i$ may be computed.

At that precise instant, the sender may exploit a timeout-based mechanism (together with a silence detector) in order to detect the first silence period whose length is not smaller than $|\delta_{sync}|$. As soon as such a silence period is detected whose length is equal to $|\delta_{sync}|$, an installation message is instantaneously transmitted by the sender for installing the synchronization j. Suppose that such a silence period has begun when the value of the sender's clock was $C_S(T) = U$. Hence, the time instant at which that installation packet (timestamped with the value U) is transmitted by the sender is $T = U + |\delta_{sync}|$. In addition, that packet transports (piggybacked in itself) the installation value RTT_j. Such an installation packet will be subsequently received at the receiving site and dealt with according to the normal procedure for audio packets.

In particular, upon the reception of the installation packet, if the attached timestamp U is equal to the value $C_{R,i}(T)$ shown by the receiver's clock, that packet causes the immediate installation of the synchronization j. In essence, the RTT_j value piggybacked in the synchronization packet is used to definitely adjust the value of the new receiver's clock. At that time, the new receiver's clock $C_{R,j}$ may immediately replace the previous clock $C_{R,i}$ in the support of all the audio playout activities. On the contrary, if the installation packet arrives at the receiver when the receiver's clock shows a value $C_{R,i}(T)$ larger than the attached timestamp U, the installation packet is discarded and the installation does not take place. Finally, if the timestamp U attached to the packet is larger than the value of the receiver's clock, that packet is buffered and the synchronization activity will take place when the current receiver's clock $C_{R,i}$ will show a value equal to U.

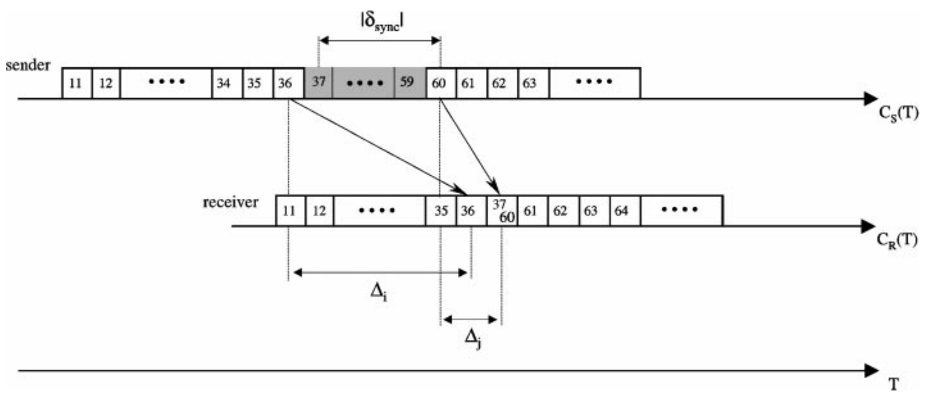

Figure 5. $\delta_{sync} < 0$: installation of a new synchronization during a sufficiently long silence period.

From the discussion above, it is easy to deduce that the proposed policy guarantees that a synchronization j (occurring after the synchronization i), whose corresponding values t_j and RTT_j are such that $t_j + RTT_j < t_i + RTT_i$, may always be installed without producing gaps by using a silence period whose length is at least equal to $|\delta_{sync}|$, since no audio packet is generated during the silence period $[U, U + |\delta_{sync}| - 1]$.

The application of the described policy to the example of figure 4 is depicted in figure 5. In that figure, the installation of the synchronization j is performed by exploiting a silence period detected at the sender (and represented with grey rectangles from 37 to 59) whose length is equal to $|\delta_{sync}| = 23$. As seen from the figure, no audio sample is lost at the receiver due to the installation of the new synchronization.

Finally, it is possible to show that with the described policy all the audio packets that are generated by the sender prior to the time instant U are guaranteed to be played out provided that they experience a transmission delay time not larger than Δ_i, since they arrive before the synchronization j is installed. Furthermore, all the audio packets that are generated after the end of the silence period (i.e., after $U + |\delta_{sync}| - 1$) are guaranteed to be played out provided that they experience a transmission delay not larger than Δ_j, since they arrive at the receiver when the synchronization j has been already installed. (The interested reader may refer to Proposition A1 in the Appendix of the already cited reference [17] for a formal description and a proof of the statements reported above). Summarizing, the improvement of the traffic conditions experienced by the audio packets is accommodated by the proposed policy by installing an up-to-date synchronization j that causes at the receiver an artificial contraction of the silence period chosen for the installation. The reduction of the silence period at the receiver (w.r.t. the original duration of the corresponding silence period at the sender) may be estimated as equal to $|\delta_{sync}|$.

3.1.2. Installing a new synchronization without introducing time collisions. With the term *time collisions* we refer to those circumstances when audio packets that would be too late for playout according to a given synchronization i, may instead be considered in time for playout if their timestamps are processed with the clock values deriving from the installation of a subsequent synchronization j. Following the same notation introduced in the previous section, we denote with $\Delta_i = t_i + RTT_i$ and $\Delta_j = t_j + RTT_j$ the values of the differences between the sender's and the receiver's clocks due to two subsequent synchronizations i and j $(j > i)$. Yet again, the value δ_{sync} may be calculated as given by $\delta_{sync} = \Delta_j - \Delta_i$.

It is straightforward to deduce that time collisions may occur in the circumstances when δ_{sync} is larger than 0. This situation corresponds to a deterioration of the traffic conditions over the underlying network, since the value Δ_j turns out to be larger than the value Δ_i. Because of this, the installation of the synchronization j causes the receiver's clock to be moved back from its current value I to a smaller value $I_* = I - \delta_{sync}$, where $I_* < I$. Thus, in order to avoid collisions, it is necessary that the receiver does not play out, when the synchronization j is active, any audio packet that was generated when the synchronization i was active. Otherwise, it could be the case that, due to the time contraction imposed to the receiver's clock by the synchronization j, some audio packets (arrived at the receiver too late for being played out according to the clock values imposed by the synchronization i) are, instead, in time for being played out according to the clock value imposed by the synchronization j.

The above mentioned problem may be avoided by adopting the following policy. Consider a situation during a transmission of audio packets in which a previous synchronization (say i) has been installed. All the playout activities concerning that audio conversation are carried out using the receiver's clock that has been set with the values of the synchronization i. Then, suppose that, at a subsequent time instant, all the preliminary activities related to a new synchronization j are successfully executed: i.e., both the probe and the response packets of the synchronization j are transmitted and received, and, at the receiving site, a new clock $C_{R,j}$ is initialized accordingly. At that time, at the sender site, the value of $\delta_{sync} = \Delta_j - \Delta_i$ may be computed. Assume that $\delta_{sync} > 0$. At this point, the sender may exploit its silence detector in order to detect the silence period that first occurs. As soon as such a silence period is detected, an installation message may be transmitted by the sender for installing the synchronization j. That installation packet will be timestamped with the time value U (corresponding to the beginning of the silence period), and will also carry the most recently computed installation value RTT_j. Upon the reception of this installation packet at the receiver, an installation procedure may be executed similar to that described in the previous Section 3.3.1.

In particular, if the timestamp U attached to the installation packet is equal to the value $C_{R,i}(T)$ shown by the receiver's clock, that packet causes the installation of the synchronization j. On the contrary, if the installation packet arrives at the receiver when the receiver's clock shows a value $C_{R,i}(T)$ larger than the attached timestamp U, the installation packet is discarded and the installation does not take place. Finally, if the timestamp U attached to the packet is larger than the value of the receiver's clock, that packet is buffered and the synchronization activity will take place when the current receiver's clock $C_{R,i}$ will show a value equal to U. As a result of a successfully installed synchronization, the receiver's clock is moved back from its old value (i.e., U) to the new value equal to $U - \delta_{sync}$, $(U - \delta_{sync} < U)$. Unfortunately, this could be the cause of time collisions.

Nevertheless, in order to avoid time collisions the receiver may now use the time value U that represents the time instant at which the installation packet was transmitted by the sender. In essence, after having installed a new synchronization j, the receiver uses the timestamp U attached to the installation packet as a *playout threshold*. If the timestamps of the received audio packets are smaller than the value U, the corresponding packets are to be discarded, since they are too late w.r.t. the deadline imposed by the the synchronization that was active when they were emitted by the sender. On the contrary, all those audio packets whose attached timestamp is larger than U are considered for playout (and consequently buffered) if they arrive before their playout deadline expires.

Thus, with the policy described above time collisions may be avoided and the decrease of the receiver's clock by δ_{sync} simply causes an artificial elongation of the silence period perceived at the receiver by a quantity equal to δ_{sync}.

From the discussion above, it is easy to deduce that with the proposed policy a synchronization j (occurring after a synchronization i), whose corresponding values t_j and RTT_j are such that $t_j + RTT_j > t_i + RTT_i$, may always be installed without causing *time collisions* of the audio packets in the talkspurt, provided that: i) a silence period is used whose original length is artificially elongated at the receiver by a quantity equal to δ_{sync}, and ii) the installation packet does not experience a transmission delay larger than $t_i + RTT_i$.

Moreover, it is possible to show that, if the installation j is successfully installed: i) all the audio packets that are generated by the sender when the synchronization i is active (i.e., before the installation packet of j is transmitted) are guaranteed to be played out at the receiver provided that their transmission delay over the network does not exceed the value given by $t_i + RTT_i$, and ii) all the audio packets that are emitted by the sender after the installation packet of j is transmitted are guaranteed to be played out at their deadlines provided that their transmission delay does not exceed the value given by $t_j + RTT_j$. (A formal description and a proof of the above mentioned statements may be found in the Proposition A2 contained in the Appendix of [17]). To conclude this section, in figure 6 an example of application of the described policy is depicted. In that figure a situation is represented where a deterioration of the traffic conditions is experienced (i.e., $\Delta_i = 2 < \Delta_j = 6$, $\delta_{sync} = 4$). Following this deterioration, a new synchronization j is installed at the receiver (the rightmost arrow in the figure) by exploiting a silence period which is

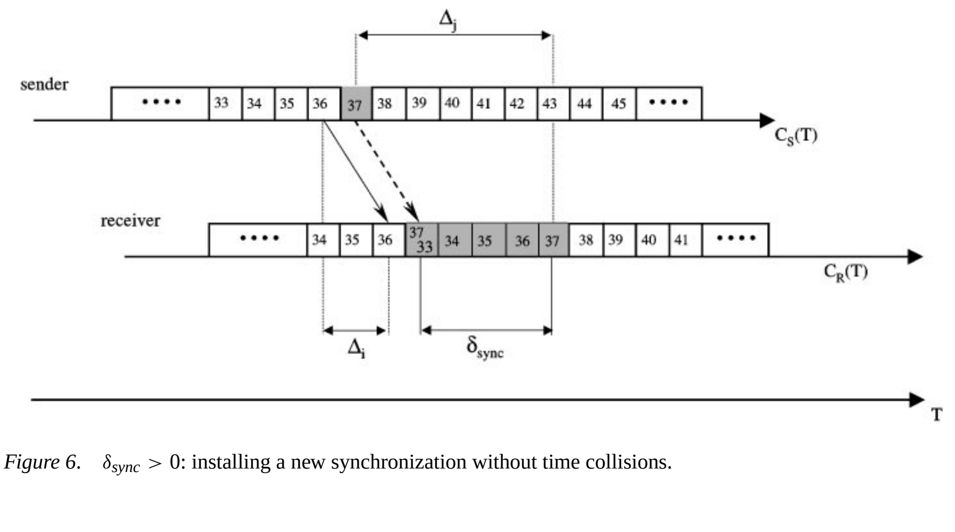

Figure 6. $\delta_{sync} > 0$: installing a new synchronization without time collisions.

detected by the sender when its clock shows a value equal to 37. The installation causes an instantaneous decrease of the receiver's clock from its old value $U = 37$ to the new value $U - \delta_{sync} = 33$. An exact application of the proposed policy guarantees that all the audio packets with timestamps in the interval [33, 36] are regularly played out at the receiver only if they arrive in time w.r.t. their deadlines and before the installation of the synchronization j. After that the synchronization j has been installed, possible late packets with timestamps in the interval [33, 36] are discarded (grey rectangles in the figure). This causes at the receiver the perception of a silence period whose original duration is elongated by $\delta_{sync} = 4$.

3.1.3. Detecting and smoothing out playout delay spikes. A possible problem with the playout delay adjustment mechanism that has been proposed in this paper is related to the possible large value for the obtained *RTT* value, that may be caused by the fact that either the probe or the response packet suffers from a very large transmission delay spike. Due to that, a very large playout delay value (termed *playout delay spike*) may be introduced that impairs the interactivity of the audio conversation. For example, consider a case when, due to a given synchronization i , an RTT_i value of 100 msec has been obtained. Based on this value, the use of the delay playout adjustment mechanism presented in this paper would have the effect to introduce at the receiver a playout delay for the audio packets ranging in the interval [100, 200] msec. Suppose now that a subsequent synchronization j is performed that obtains an RTT_j value equal to 600 msec. This very large value for RTT_j would have the effect of setting the playout delay value to a very large value ranging in the interval [600, 1200] msec. This very large playout delay may be not considered tolerable for an interactive audio conversation. Summarizing, a very large *RTT* value obtained with a given synchronization may cause an untolerable playout delay value that disrupts the interactivity of the conversation. Nevertheless, that problem may get solved by adopting a policy which was inspired by delay spike detection and management mechanism proposed in [13]. Our policy works by using two different modes of operation depending on whether a transmission delay spike has been detected or not. In the *normal* mode (i.e., no transmission delay spike has been detected) the playout delay adjustment mechanism operates by calculating the playout delay to be introduced in the playout process, just as already described. Instead, in the *spike-detected* mode (i.e., a very large transmission delay spike has been detected) the very large *RTT* value obtained from the spike-affected synchronization is smoothed out by multiplying it with a *smoothing factor* k ($k < 1$). Let us denote, in the following, with $\overline{RTT_j}$ such a smoothed value obtained with the synchronization j. Each audio packet that arrives at the receiver after a spike-affected synchronization j has been installed is played out after a playout delay value equal to $t_j + \overline{RTT_j}$.

In the following we give a high level description of the policy we use to detect a playout delay spike. That policy is based on the comparison between the *RTT* values obtained from two consecutive synchronization activities j and i ($j > i$). For ease of understanding, the policy is presented in C-language-like pseudo code in Table 1. For each most recently computed RTT_j value, the algorithm checks the current mode and, if necessary, switches its mode to the other one (lines 1-9 in the leftmost side of the table). More precisely, if the most recently computed RTT_j value is larger than some multiple h ($h > 1$) of the RTT_i

Table 1. Playout delay spike management: delay spike detection (left) and playout delay estimation (right).

```
(1) **For each** newly computed $RTT_j${
(2) **IF**(mode == SPIKE)
(3) **IF** ($RTT_j <= h\times$ old-$RTT_i$)
(4) mode = NORMAL
(5) **ELSE**(mode == NORMAL)
(6) **IF** ($RTT_j > h \times RTT_i$){
(7) mode = SPIKE
(8) old-$RTT_i = RTT_i$}
(9) $RTT_i = RTT_j$}
```

```
(1) **IF** (mode == SPIKE){
(2) $k = f(h)$
(3) $\Delta_j = t_j + \overline{RTT_j} \times k$}
(4) **ELSE** (mode == NORMAL)
(5) $\Delta_j = t_j + RTT_j$
```

value then a delay spike is considered to be detected, and the algorithm switches to the *spike-detected* mode (lines 5–7 in the leftmost side of the table). In such spike-detected mode, the smoothed value $\overline{RTT_j}$ is computed and then used for playout (lines 1–3 in the rightmost side of the table). The end of a spike is detected similarly. If the most recently computed RTT_j value is smaller than some multiple h of the RTT value experienced when the spike-affected period began, the mode is reset to *normal* (lines 2–4 in the leftmost side of the table). Clearly, when the algorithm operates in the normal mode, the normal RTT_j value is used at the receiver for calculating the playout delay (lines 4-5 in the rightmost side of the table).

In the calculation of the smoothed value $\overline{RTT_j}$ the following policy has been adopted. If h is the constant value used to detect the transmission delay spike, then the smoothing factor k is calculated as $k = f(h)$, such that $k < 1$. For instance, take into account the example provided at the beginning of this section. If the value h used to detect the transmission delay spike is equal to 3 ($600 > 3 \times 100$), then a possible value for the smoothing factor k may be given by $k = 1/h = 1/3$. Consequently, the smoothed value $\overline{RTT_j}$ would be 200 msec, and the playout delay imposed by our proposed mechanism would result to range in the interval given by $[200, 400]$ msec.

The motivations behind the mechanisms adopted for both detecting the transmission delay spikes and smoothing out the correspondent playout delay spikes have been derived by taking into account the considerations provided in [13]. Those considerations have been adapted to our proposed playout delay adjustment mechanism as described in the following. In that cited paper (as well as in other ones, e.g., [2]) the results of several experiments concerning audio conversations over the Internet are reported that show that typically the delay values of audio packets during a spike present an initial steep rise and then decrease in a monotonic linear fashion. The delay spikes behavior mentioned above justifies the choice of a multiple of the current *RTT* value in order to detect the occurrence of a very big delay spike. In fact, if the spike is small (that is, its delay is less than an order of magnitude larger than other baseline delays) then probably the normal mode of operation of our proposed mechanism is able to assimilate that spike without causing any disruption at the interactivity of the audio session. Instead, if the spike is large, then the comparison between the *RTT* value obtained with the most recent synchronization and an adequate multiple of the current

RTT may be enough to detect the spike, that can be then assimilated with a smoothed $\overline{RTT}$ value.

With respect to those big delay spikes, it is also worth mentioning that, in principle, our proposed policy may fail either in detecting or in assimilating them, if they either occur in between two subsequent synchronizations or are properly contained in a unique talkspurt. Nevertheless, several experiments (detailed in the following section) have been carried out with our playout delay adjustment mechanism that show that the percentage of packet loss due to either undetected or missed transmission delay spikes is very small, and do not disrupt the audio intelligibility. Probably, this is due to the following three factors: i) the frequency used to periodically repeat the synchronization activity (one synchronization activity per second), ii) the distribution of silence periods in human conversations (it has been experimentally shown that typical human conversations embody almost a 50–60% of silence periods [20]), and finally iii) the typical large duration (about 3–4 seconds) of big delay spikes that usually span through multiple talkspurts. Finally, we wish to mention that an adequate choice for the values of the parameters h and k should be made on a per-connection basis by taking into account the end-to-end transmission delay values that are usually experienced on that connection [17].

4. Experimental evaluation of the mechanism

A working prototype implementation of the playout control mechanism presented in this paper was performed using the C programming language, and the development environment provided by the SunOS 4.3 (BSD Unix) operating system. Using such a prototype implementation, an initial extensive experimentation was carried out aiming at measuring the performance of the mechanism. In the following three sections, the prototype implementation used for the experimentation, the obtained experimental results, and a set of comparative simulation results are presented.

4.1. Prototype implementation and measurement architecture

The Unix socket interface and the datagram based UDP protocol were used to transmit and receive the sampled audio packets. The coding schemes that were used to produce the audio packets use 8-kHz sampled speech with bit rates varying from 5.3-kbit/sec to 8-kbit/sec. In particular, all the audio packets used to perform the measurements were produced using a codec based on the ITU-T G.729 standard [6] that provides coding of speech at 8-kbit/sec while maintaining toll-level audio quality. In addition, in order to reconstruct possibly lost (or corrupted) audio packets, a forward error correction-based mechanism was implemented that was able to add to a given packet redundant information (under the form of a highly compressed version of the previous packet). To this aim, a codec based on the ITU-T G.723.1 standard was exploited that was able to code speech of a low, synthetic quality at 5.3-kbit/sec. Such an alternative coding scheme was adopted based on the consideration that it may provide redundant information by adding only a small amount of byte overhead per packet. This redundant information was piggybacked

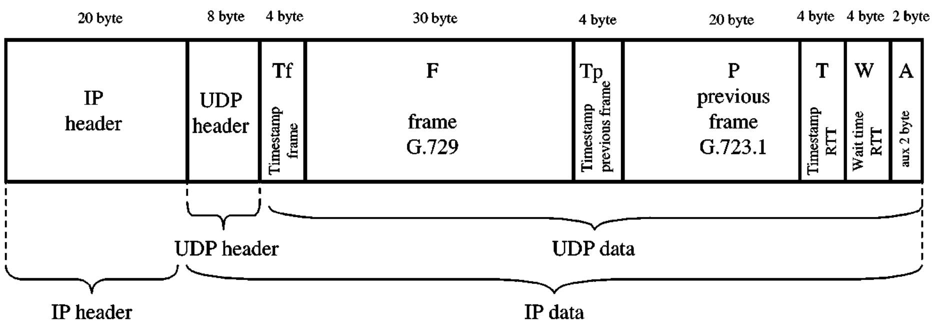

Figure 7. Audio packet used in the experimentation.

in the packet following that containing the primary speech codeword. Thus, the loss of an individual packet can be repaired using the redundant information carried by the following packet [5]. Needless to say, the use of this redundancy technique entails an increase in the overall delay due to both the algorithmic delay of the coding process and the reconstruction delay introduced at the receiver. In substance, the measurements were all done using audio packets that were produced by exploiting both the G.729 and the G.723.1 based codecs.

The resulting audio packets all had the structure represented in figure 7. In particular, besides the IP and UDP headers (respectively of 20 and 8 bytes), each packet includes a 30 msec of speech coded with the G.729 primary coding algorithm, and a 30 msec of previous speech coded with the G.723.1 secondary coding algorithm. The 30 msec of the current speech result in a 30 bytes field (denoted "F" in figure 1), while the 30 msec of previous speech are coded using the 20 bytes field (denoted "P" in figure 1). In addition, to each audio packet two timestamps are associated: the first timestamp (the "Tf" field in figure 1) records using 4 bytes the time of generation of the 30 msec of the current speech. Instead, the second timestamp (the "Tp" field in figure 1) records using 4 bytes the time of generation of the previous 30 msec of speech. The time values recorded by the timestamps use 0.1 msec as time base unit. Hence, a 4 byte timestamp field may be enough to timestamp audio packets for a consecutive 100 hours long period of time. No sequence number was attached to the packet in order to reduce the total amount of bytes needed, but the timestamps were used, at the application level, both to measure end-to-end transmission delays and also to detect packet loss. In addition, the audio packets were also used in the prototype implementation of our protocol for implementing two out of the three phases of the synchronization activity, namely the "probing" and the "installation" phases. Hence, a 4 byte field (denoted "T") was included in each packet to carry the RTT_i value needed for the installation of the synchronization i. Moreover, since each audio packet was emitted by the sender only at regular intervals of 30 msec, the 4 byte field "W" was included to record the total quantity of time elapsed from the time instant when the *RTT* value was available at the sender, and the time instant when that *RTT* value was effectively transmitted. That time value was used at the receiver in order to calculate the average time of completion of the synchronization activity. Finally, an additional 2 byte field (termed "A" in figure 1) was used for numbering subsequent synchronizations. Summarizing, the total number of bytes used for producing the IP-based audio packets amounts to exactly 96 bytes, out of which 28 are used to encode both the IP and the UDP headers, while the remaining 68 bytes are used to encode all the audio data and the correspondent timestamps.

One of the most important performance metrics for packet audio is the percentage of packets lost at the destination host. Such loss results from either the late arrival of a packet (i.e., the arrival of a packet after its scheduled playout point) or a premature arrival of a packet. In the latter case, packet loss derives from the limitation on the finite size of the receiver's playout buffer. In order to manage adequately the receiver's playout buffer, in our prototype implementation a set of communicating processes implements at the application level of the receiving host the buffering/playout policy described in Section 3. In summary, a circular buffering scheme has been adopted according to which only "on-time" audio packets (see below) are first queued in the empty locations of the buffer and then periodically fetched and sent to the audio device for being played out.

As illustrated in Section 3, we have devised a method for an accurate calculation of the playout buffer dimensions depending on the network traffic conditions. In essence, that method dictates that with each synchronization i the receiver's playout buffer dimensions are calculated as equal to $D \times \Delta_i / r$, where D is the total number of bytes needed to encode an audio sample of r milliseconds with a given codec and $\Delta_i = t_i + RTT_i$ is the time value (expressed in milliseconds) of the difference between the sender's and the receiver's clocks according to a given synchronization i. Both buffer overflow and underflow are prevented by managing the buffer according to the strategy summarized below. When an audio packet arrives (i.e., it is delivered to the application level of the receiving host), its timestamp t is compared with the value T of the receiver's clock, and a decision is taken according to the rules depicted in Table 2. If $t < T$, the packet is discarded having arrived too late w.r.t. its playout time to be buffered (first row in Table 2). If $t > T + \Delta_i$, the packet is discarded having arrived too far in advance of its playout time to be buffered (second row in Table 2). Instead, if $T < t <= T + \Delta_i$, the packet is arrived in time for being played out and it is placed in the first empty location in the playout buffer (third row in Table 2). Using the same rate adopted for the sampling of the original audio signal, the playout process fetches audio packets from the buffer and sends them to the audio device for playout as discussed in the following. When the receiver's playout clock shows a value equal to T, the playout process searches in the buffer the audio packet with timestamp T. If such a packet is found, it is fetched from the buffer and sent to the audio device for immediate playout, while the buffer location where the audio packet has been found is marked as empty (fourth row in Table 2). If neither the packet timestamped with T nor the packet (timestamped with $T + r$)

Table 2. Buffering/playout policy implemented at the application level of the receiving host.

Condition	Policy
$t < T$	Packet discarded (late arrival)
$t > T + \Delta_i$	Packet discarded (premature arrival)
$T < t <= T + \Delta_i$	Packet buffered (waiting for playout)
$t = T$	Packet sent to the audio device for playout

that contains the same r milliseconds of previous speech coded with the secondary coding algorithm is present in the buffer, then the playout process replaces the corresponding audio sample with a silence period of length r.

As new synchronization activities are carried out during an audio communication the playout buffer dimensions are recomputed accordingly. However, it is important to notice that with the buffering/playout policy we have adopted, even if the the playout buffer is dynamically manipulated in order to accommodate new synchronization events, those audio packets that are already queued in the playout buffer are neither removed nor invalidated. When a new synchronization j replaces a previous synchronization i at the receiver, in fact, two are the possible cases: either $\delta_{sync} > 0$ or $\delta_{sync} < 0$. If $\delta_{sync} > 0$, this means that a deterioration of the current network traffic conditions has been experienced. In order to take into account this traffic deterioration, the receiver's playout buffer size should be increased. The up-to-date dimensions of the playout buffer may be calculated as equal to $D \times \Delta_j/r$, with an increase of exactly $\lceil \delta_{sync}/r \rceil$ additional locations w.r.t. to the buffer used when the synchronization i was active. In our prototype implementation, those $\lceil \delta_{sync}/r \rceil$ additional empty locations are dynamically inserted in the same circular buffer that was in use with the previous synchronization i. Instead, the other $\lceil \Delta_i/r \rceil$ old locations of the buffer (where some audio packets are possibly queued) are left untouched and are processed by the playout process as already mentioned.

If $\delta_{sync} < 0$, an improvement of the network traffic conditions between two subsequent synchronization events has been measured. We may take advantage of such an improvement in the traffic conditions to decrease the playout buffer dimensions, thus saving memory space. Since each new synchronization j characterized by a $\delta_{sync} < 0$ value may be installed at the receiver only at the end of a silent interval of length equal to $|\delta_{sync}|$ (see Section 3.1.1), we are sure that, with our buffering/playout policy, at least $\lceil |\delta_{sync}|/r \rceil$ empty locations always exist in the buffer when a new synchronization with a $\delta_{sync} < 0$ value is installed. Those $\lceil |\delta_{sync}|/r \rceil$ empty locations may be safely removed from the buffer. The remaining Δ_j/r locations of the buffer (where some audio packets are possibly queued) are left untouched by our mechanism and are processed according to the already mentioned playout policy.

The performance of the prototype implementation of the playout delay control mechanism presented in this paper has been evaluated using an IP-based internetworked infrastructure connecting two SPARCstation 5 workstations running the SunOS 4.3 operating system and situated, respectively, at the Laboratory of Computer Science of Cesena (a remote site of the University of Bologna), and at the C.E.R.N. Institute in Geneva. Each one of the used workstations was locally connected to a 10-Mbit/sec Ethernet LAN. In order to perform measurements while avoiding the issue of sender's and receiver's clock synchronization, the audio packets were sent using the following communication scenario. The source host was the same as the destination host and were located in the same workstation situated at Laboratory of Computer Science of Cesena. Instead, the workstation situated at the C.E.R.N. Institute in Geneva operated as an intermediate host. In essence, each generated audio packet was sent from the source host to the intermediate host. Such an intermediate host, upon the receipt of a packet, simply echoed that packet back to the destination host. This policy has allowed us to take experimental measurements of end-to-end packet delays not affected from the clock synchronization problem, since in the above mentioned scenario

```
poseidon.csr.unibo.it (137.204.72.49):~$ traceroute cms1.cern.ch
 1  romr01.csr.unibo.it (137.204.72.254)
 2  137.204.27.254 (137.204.27.254)
 3  137.204.1.20 (137.204.1.20)
 4  192.12.77.29 (192.12.77.29)
 5  niscn1.bo.infn.it (131.154.151.253)
 6  cnaf-gw1.infn.it (131.154.1.9)
 7  cnaf-int-gw.infn.it (131.154.1.2)
 8  ten-34.garr.net (192.135.34.22)
 9  garr.IT.ten-34.net (193.203.226.25)
10  it.CH-1.ten-34.net (193.203.226.9)
11  ch-1.CH-2.ten-34.net (193.203.226.30)
12  cgate1.cern.ch (192.65.185.1)
13  r513-c-rci47-5-gb0.cern.ch (128.141.211.20)
14  cms1.cern.ch (137.138.129.123)
```

Figure 8. Route between the Laboratory of Computer Science of Cesena and the C.E.R.N. Institute.

the end-to-end delays coincide with round trip delays. In figure 8 the routes taken by the packets sent over the above mentioned connection are shown that were obtained with the *traceroute* routine.

Prior to illustrating the measurements that were taken during the experiments, it is also interesting to notice that at the time the experiments were carried out (September–October 1997) the experimental testbed that was used had almost all interconnecting links with bandwidth ranging from a few to several megabits per second. Indeed, the unique bottleneck link of the above mentioned communication scenario was the regional link interconnecting the router situated at Laboratory for Computer Science of Cesena with the router of the Network Center of the University of Bologna (512 Kbit/sec). It is also interesting to note that the router of the Laboratory for Computer Science of Cesena is typically under a heavy load, since it operates as an interconnecting router for a number of remote university sites of the region.

4.2. Experimental results and data interpretation

Several experiments (about 30) of unicast audio conversations were conducted using the software prototype implementation of our mechanism during daytime (from 7 a.m to 8 p.m) in different days during the period September–October 1997. Each experiment carried out between Cesena and Geneva consisted in transmitting about 15.000 audio packets. Those packets were generated using both the G.729 and the G.723.1 based codecs from prerecorded 10 minutes long audio files.

Two are the most important metrics that influence the users' perception of audio data: 1) the percentage of audio packets that arrive too late at the destination to be played out (and so can be effectively considered lost), and 2) the amount of total delay that an audio packet

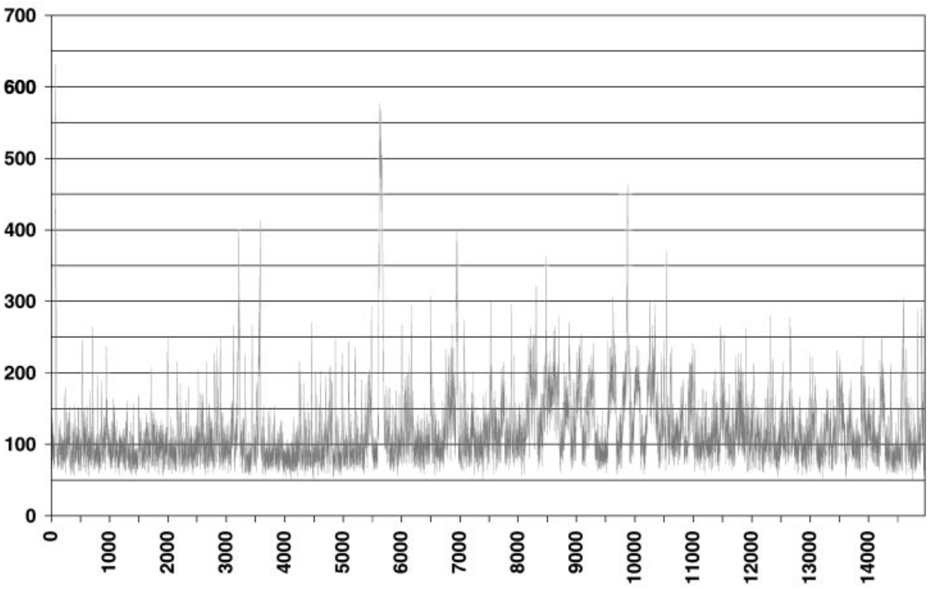

Figure 9. Evolution of the round trip delay (start time: 02.00 pm 10/9/97).

has to experience before it is played out at the destination. Hence, in order to measure the performance of our mechanism, during each experiment we measured: i) the percentage of lost packets, ii) the end-to-end transmission delay experienced by audio packets during transmission, and finally iii) the additional buffering delay imposed by our mechanism.

As an example, in figure 9 the evolution of the (round trip) transmission delay of the nth packet is plotted (as a function of n) that was measured during one of the conducted experiments. The transmission of the 15.000 audio packets of the experiment started at 2.00 pm, on 9th October 1997. Note that on the y-axis of figure 9 the value of the round trip transmission delays are reported in milliseconds. Instead, on the x-axis the value of the timestamps are reported that were re-numbered using consecutive integers (starting from 0), thus eliminating all the gaps in the timestamp-based numbering of the audio packets that were caused by the silence periods between talkspurts. From a statistical analysis of the provided data, it is possible to measure an average value of the round trip transmission delay almost equal to 119 msec, with a maximum delay spike of 627 msec at the beginning of the transmission, and only another spike exceeding 550 msec (after 1/3 of the period of the audio transmission). It is also worth noticing that the total number of audio packets that were completely lost by the network (i.e., never arrived packets) was rather low (about 40). This is probably due to the large bandwidth that is provided by the communication links that interconnect the end hosts of our experiment.

In figure 10, the evolution of the playout delay, i.e., the total amount of delay that each packet n has to experience before it is played out, is reported (as function of n) that was obtained with our playout delay control mechanism, where the synchronization activity was repeated with the frequency of 1 second. Yet again, on the x-axis of figure 10, the value of

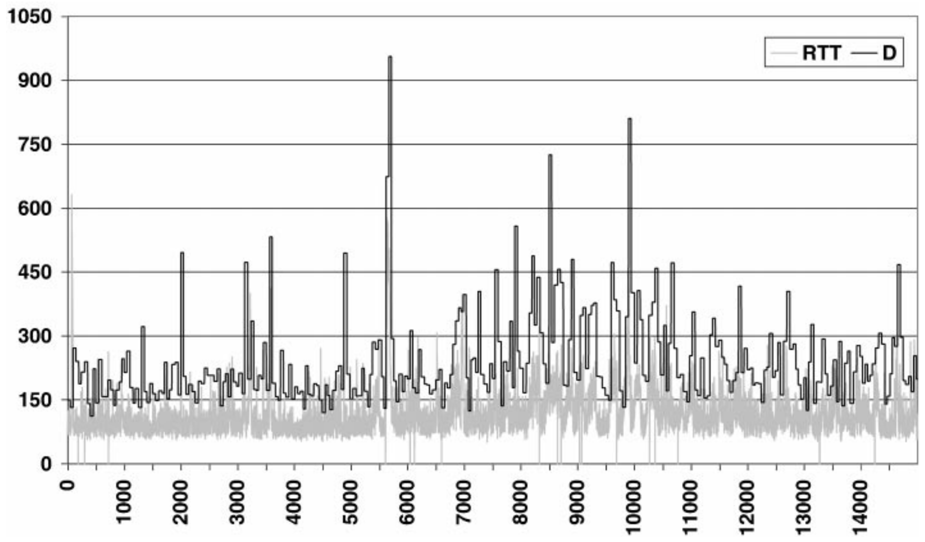

Figure 10. Evolution of the playout delay (start time: 02.00 pm 10/9/97).

the timestamps are reported re-numbered in order to eliminate timestamp gaps. Instead, on the *y*-axis of figure 10 the values of both the (round trip) transmission delay and the playout delay are reported for each packet expressed in milliseconds. In particular, the values of the transmission delays are shown with grey lines (and denoted as "*RTT*" in the caption inside the figure), while the values of the playout delays are plotted with black lines (and denoted as "D" in the caption inside the Figure). More precisely, note that when a grey line exceeds the corresponding black line, this entails that this packet has arrived too late with respect to the playout deadline computed by our playout control mechanism and, consequently, is discarded. On the contrary, if the black line encapsulates the grey line, this means that the corresponding audio packet has arrived in time to be played out at the receiver. From a statistical analysis of the data plotted in figure 10, several interesting considerations may be derived. First, it is important to notice that our playout control mechanism keeps the percentage of lost packets below the threshold of 5%. Furthermore, the average playout delay was calculated as equal to 238 msec. This playout delay value may be considered tolerable for audio conversations and guarantees a good degree of interactivity. In addition, it is worth noticing from figure 10 that a number of playout delay spikes (approx. 15) were produced that exceeded the value of 450 msec. Nevertheless, it is also worth mentioning that our playout control mechanism was used during the experiment with the value of the smoothing factor k equal to 1, that is the mechanism for smoothing out the playout delay spikes was kept deactivated. Finally, in order to fully assess the performance of the proposed playout control mechanism, the percentage of lost packets obtained with our mechanism (i.e., 5%) has been contrasted with the percentage of audio packets that would be lost if a constant playout delay of 150 msec was used throughout all the performed experiment. The percentage of lost audio packets (due to a playout delay of 150 msec) was measured as equal to 23%.

Table 3. Experimental results: average playout delay and packet loss.

Experiment	Start Time	Playout Delay	Packet Loss	Spike Management
# 1	08:20am 10/15/97(Th)	188 msec	7%	Yes ($k = 4/5$)
# 2	02:00pm 10/9/97(Fr)	238 msec	5%	No ($k = 1$)
# 3	11:00am 10/4/97(Su)	202 msec	6%	Yes ($k = 4/5$)
# 4	7:40pm 9/21/97(Mo)	229 msec	5%	No ($k = 1$)
# 5	04:15pm 9/16/97(We)	207 msec	5%	No ($k = 1$)

We conclude this section by reporting in Table 3 the values of the average playout delay and the packet loss percentage of only 5 out of the 30 experiments that were carried out. It is worth mentioning that, besides the results provided in Table 3, also in all the other 25 cited experiments both an acceptable value of the average playout delay (ranging in the interval 180–250 msec) and a tolerable loss percentage of up to 6–7% were experienced, and only rarely playout delay spikes exceeding 600/700 msec were imposed by our mechanism. On the contrary, in all those other experiments that were conducted using a constant playout delay (typically obtained by increasing of a 10% the value of the average transmission delay) an amount of lost audio packets was experienced ranging from about 15% to almost 40%.

4.3. *Comparative simulation results*

This section is devoted to the comparison of our mechanism with another playout delay control mechanism recently proposed [13]. A new adaptive (history-based) delay adjustment algorithm was proposed that tracks the network delays of received audio packets and efficiently maintains delay percentile information [13]. That information, together with an appropriate delay spike detection algorithm, is used to dynamically adjust talkspurt playout delays. In essence, the main idea behind that algorithm is to collect statistics on packets already arrived and then to use them to calculate the playout delay. Instead of using some variation of the stochastic gradient algorithm in order to estimate the playout delay, each packet's delay is recorded and the distribution of packet delays is updated with each new arrival. When a new talkspurt starts, the algorithm proposed in [13] calculates a given percentile point (say q) for the last arrived w packets, and uses it as the playout delay for the new talkspurt. In addition, the algorithm accommodates delay spikes in the following manner. Upon the detection of a delay spike, the algorithm stops collecting packet delays, and follows the spike (until the detection of the spike's end) by using as playout delay the delay experienced by the packet that commenced the spike. Upon detecting the end of the delay spike, the algorithm resumes its normal operation mode. The authors of [13] have experimentally shown that their algorithm outperforms other existing delay adjustment algorithms over a number of measured audio delay traces, and performs close to a theoretical optimum over a range of parameters of interest.

Thus, in order to assess the performance of the mechanism proposed in this paper, we carried out a simulation experiment that compares our playout delay adjustment algorithm with the history-based mechanism proposed in [13]. As already mentioned, the most important metric that influence the users' perception of audio data is represented by the average playout delay vs. the packet loss. Hence, the two algorithms were compared with respect to these values. To this aim, a simulator was designed and developed that reads in the transmission delay of each packet from a given trace, detects if it has arrived before the playout time that is computed by each of the two algorithms, and executes the algorithm [16]. The simulator is also able to calculate the average playout delay and the packet loss for a given trace (for each of the two algorithms). Thus, we run the simulator several times in order to simulate the use of the history-based algorithm over the measured audio delay trace reported in figure 9. The simulator was used by varying (at each run) the percentile point q, in order to reach the following fixed values of loss percentage (approx. 3%, 4%, 5%, 7%, 10%, 12%, 15%, 20%), and then to measure the correspondent average playout delays. The values of the percentile point q that were used to keep the packet loss percentage below the values reported above were the following: .995, .99, .985, .98, .97, .96, .94, .92.

Subsequently, we run repeatedly the simulator to simulate our algorithm over the same measured audio delay trace depicted in the figure 9. The purpose of those simulations was to identify that set of cases where our algorithm reaches approximately the same values of the loss percentage that were obtained with the history-based algorithm, and then to measure the correspondent average playout delays. This allow us to compare the performance of the two algorithms under identical network conditions. To this end, we simulated our algorithm using the buffer size as the control parameter to be varied to achieve different loss percentages, as was done in [13, 14]. Such a variation of the buffer size was achieved by tuning a sensitivity factor in the synchronization formula used to obtain the synchronization values to be installed at the receiver. In essence, instead of using the "regular" synchronization formula (namely $\Delta = t + RTT$), the following formula was used to calculate the playout delay: $\Delta = t + (p \times RTT)$, where p is a non negative constant. Hence, in order for our algorithm to obtain approximately the same packet loss percentages obtained with the history-based algorithm the following values of p were approx. needed: 1.5, 1.3, 1, 4/5, 3/4, 2/3, 1/2, 2/5. The average playout delays obtained using this simulation technique were then compared with the average playout delays obtained with the simulation of the history-based algorithm, at a parity of packet loss percentage.

It is worth mentioning that our algorithm was used twice for simulation. First, it was simulated with the playout delay spike smoothing mechanism deactivated (i.e., $k = 1$), then the experiment was repeated with the playout delay spike smoothing mechanism activated (i.e., $k = 4/5$). In conclusion, in Table 4 the values of the average playout delays are reported, that were obtained with the simulation of respectively: i) the history-based algorithm (first row in Table 4), ii) our algorithm with the playout delay spike smoothing mechanism

Table 4. Simulation results: average playout delays (msec).

Algorithm	3%	4%	5%	7%	10%	12%	15%	20%
History-based	332	249	215	201	178	172	161	152
Our-algorithm I	298	260	202	194	181	170	166	157
Our-algorithm II	352	301	237	223	210	193	187	172

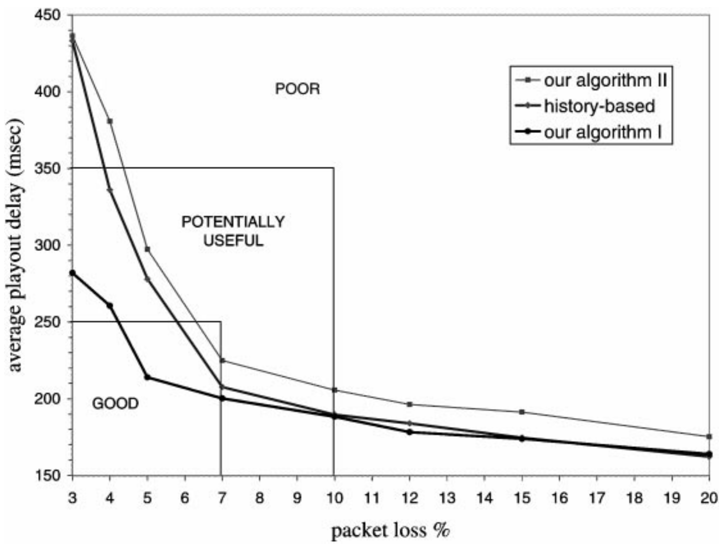

Figure 11. Comparison of the analyzed algorithms: average playout delay vs loss rate.

activated (second row in Table 4), iii) our algorithm with the playout delay smoothing mechanism deactivated (third row in Table 4). From the analysis of Table 4, it is possible to deduce that the history-based algorithm outperforms our algorithm with the playout delay spike smoothing mechanism deactivated. Instead, based on the consideration that audio of acceptable quality may be obtained only if lower delays are achieved while the loss percentage does not exceed the value of 10%, our algorithm (with the delay spike smoothing mechanism activated) shows better performance w.r.t. to the history-based algorithm as long as the loss percentage is kept below the value of 10%.

In order to better illustrate the results of our comparison, in figure 11 the average playout delay is plotted as a function of the loss percentages for each analyzed algorithm. The plot of the playout delay has been obtained by running the simulator over all the 30 experimental traces, and then averaging the results. In order to provide the reader with an understanding of the effect that various delay and loss rates (as well as buffer dynamics) have on the quality of the perceived audio, we have reported in figure 11 an approximate and intuitive representation of three different ranges for the quality of the perceived audio. The three following audio quality ranges have been adopted from [6]: *good*, for delays of less than 200/250 msec and low loss rate, *potentially useful*, for delays of about 300–350 msec and higher loss rates, and *poor*, for delays larger than 350 msec and very high loss rates. As seen from the Figure, our algorithm (with the playout delay smoothing mechanism activated) shows better *average* performance w.r.t. the history-based algorithm in both the *good* and the *potentially useful* audio quality regions.

To conclude this comparison, it is worth noticing that the synchronization policy embedded in our mechanism imposes little computational overhead (both at the source and the destination hosts) w.r.t. the history-based algorithm, where delay statistics have to be collected, and percentile points of the delay distribution have to be calculated on-the-fly.

5. Concluding remarks

An adaptive mechanism for the control of the playout delay of audio packets over the Internet has been proposed. This mechanism is suitable for dynamically adjusting the talkspurt playout delays for unicast, voice-based communications where conversational audio with silence periods between subsequent talkspurts is transmitted. We commenced the design and the experimental evaluation of our playout delay control mechanism during the Summer of 1997, and completed them in October 1997. During that period, several experiments were carried out that showed that our design was successful in maintaining satisfiable values of the average playout delay, while minimizing the number of audio packets that were lost during an audio transmission performed over a interconnected (IP based) link. The designed mechanism assumes neither the existence of an external algorithm for maintaining an accurate synchronization at both the sending and the receiving sites of the audio connection, nor a Gaussian distribution for the end-to-end transmission delays experienced by the audio packets. In addition, our mechanism provides: i) an internal and accurate algorithm that maintains tight time synchronization between the sending and the receiving hosts, ii) a method for adaptively estimating the audio packet playout time (on a per-talkspurt basis) with an associated minimal computational overhead, iii) an exact and simple technique for dimensioning the playout buffer depending on the traffic conditions of the underlying network.

An extended version of this paper, referenced as [17] in the bibliography, is available via anonymous FTP from *ftp.cs.unibo.it:/pub/TR/UBLCS.*

Acknowledgments

This research has been funded by the Italian MURST and CNR under a grant. We are indebted to Prof. Paolo Capiluppi (University of Bologna and C.E.R.N.) for letting us to experiment our software prototype implementation using a workstation of the C.E.R.N. Institute in Geneva. We are grateful to the anonymous referees of *Multimedia Tools and Applications* for their exhaustive and helpful reviews of this paper.

References

1. M. Bernardo, R. Gorrieri, and M. Roccetti, “Formal performance modeling and evaluation of an adaptive mechanism for packetized audio over the internet,” Formal Aspects of Computing, to appear.
2. J. Bolot, H. Crepin, and A. Vega Garcia, “Analysis of audio packet loss on the internet,” in Proc. of Network and Operating System Support for Digital Audio and Video, Durham (NC), 1995, pp. 163–174.
3. D. Cohen, “Issues in transnet packetized voice communications,” in Proc. of Fifth Data Communication Symposium, Snowbird (UT), 1977, pp. 6.10–6.13.
4. F. Cristian, “Probabilistic clock synchronization,” Distributed Computing’ Vol. 3, pp. 146–158, 1989.
5. V. Hardman, M.A. Sasse, and I. Kouvelas, “Successful multi-party audio communication over the Internet,” in Communications of the ACM Vol. 41, pp. 74–80, 1998.
6. ITU-T Recommendation G.729, “Coding of speech at 8-kb/s using coniugate-structure algebraic-code-excited linear-prediction,” 1996.
7. ITU-T Recommendation G.723.1, “Dual rate speech coder for multimedia communications transmitting at 5.3/6.3-kb/s,” 1996.
8. V. Jacobson and S. McCanne, *vat*, ftp://ftp.ee.lbl.gov/conferencing/vat/.

9. T.J. Kostas, M.S. Borella, I. Sidhu, G.M. Schuster, J. Grabiec, and J. Mahler, “Real-time voice over packet-switched networks,” IEEE Network, Vol. 12, pp. 18–27, 1998.
10. W.E. Leland, M.S. Taqqu, W. Willinger, and D.V. Wilson, “On the Self-Similar Nature of Ethernet Traffic,” in IEEE/ACM Trans. on Networking Vol. 2, pp. 1–15, 1994.
11. M. Macedonia and D. Brutzmann, “mbone provides audio and video across the Internet,” in IEEE Computer Magazine, Vol. 21, pp. 30–35, 1994.
12. D.L. Mills, “Improved algorithms for synchronizing computer network clocks,” in Proc. of *ACM SIG-COMM’94*, London (UK), 1994, pp. 317–327.
13. S.B. Moon, J. Kurose, and D. Towsley, “Packet audio playout delay adjustment: Performance bounds and algorithms,” in ACM Multimedia Systems Vol. 6, pp. 17–28, 1998.
14. R. Ramjee, J. Kurose, D. Towsley, and H. Schulzrinne, “Adaptive playout mechanisms for packetized audio applications in wide-area networks,” in Proc. of *IEEE INFOCOM’94*, Montreal (CA), 1994.
15. P.V. Rangan, S.S. Kumar, and S. Rajan, “Continuity and synchronization in MPEG,” in IEEE Journal on Selected Areas in Communications, Vol. 14, pp. 52–60, 1996.
16. M. Roccetti, M. Bernardo, and R. Gorrieri, “Packetized audio for industrial applications: A simulation study,” in Proc. of 10th European Simulation Symposium, Nottingham (UK), 1988, pp. 495–500.
17. M. Roccetti, V. Ghini, G. Pau, P. Salomoni, and M.E. Bonfigli, “Design and experimental evaluation of an Adaptive playout delay control mechanism for packetized audio for use over the Internet,” UBLCS Technical Report n. 98-4 , Laboratory for Computer Science, University of Bologna, May 1998.
18. H. Schulzrinne, “Voice communication across the Internet: A network voice terminal,” Tech. Rep., Dept. of ECE and CS, Univ. of Massachusetts, Amherst (MA), 1992.
19. H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, “RTP: A transport protocol for real-time applications,” Request for Comments 1889, IETF, Audio-Video WG, 1995.
20. A. Vega Garcia, “Mecanismes de controle pour la transmission de l’audio sur l’Internet,” Doctoral Thesis in Computer Science, University of Nice-Sophia Antipolis, Ecole Doctoral SPI, 1996.
21. L. Zhang, “RSVP: A new resource reservation protocol,” in IEEE Network Magazine Vol. 7, pp. 8–18, 1993.

Marco Roccetti is a Professor of Computer Science at the Department of Computer Science of the University of Bologna (Italy). He received the Laurea degree in Electronic Engineering from the University of Bologna, in the academic year 1987/88. From 1992 to 1998, he was with the Department of Computer Science of the University of Bologna as a research associate. He has worked in distributed computing systems, communication protocols and performance analysis. His current research interests include architectural design of distributed real time computing systems and protocol architectures for Internet based multimedia communications.

Vittorio Ghini is a Ph.D. student in Computer Science at the Computer Science Department of the University of Bologna (Italy). He received the Laurea degree (with honors) in Computer Science from the University of

Bologna in 1997. He has worked in digital radio and computer networks. His current research interests include interconnection networks, digital radio and QoS management over IP.

Giovanni Pau received the Laurea degree in Computer Science from the University of Bologna (Italy) in 1998. In the same year, he started his Ph.D. studies in Computer Engeneering and Telecommunications at the Computer Engeneering Department of the University of Bologna. His current research interests include network architecture and protocols, distributed systems and QoS in the IP-based differenziated services.

Paola Salomoni received the Laurea degree in Computer Science from the University of Bologna (Italy) in 1992. Since 1995, she is an Assistant Professor of Computer Science at the Department of Computer Science of the University of Bologna. Her current research interests include Internet based multimedia systems and artificial intelligence.

Maria Elena Bonfigli received the Laurea degree (with honors) in Computer Science from the University of Bologna (Italy) in 1995. In 1997 she started her Ph.D. studies at Department of Computer Science of the University of Bologna, Italy. Since 1998 she is collaborating with CINECA—Interuniversity SuperComputing Centre—within the Nu.M.E. project. Her research interests include design and development of distributed multimedia systems and 3-D Web interfaces.