Comparing the QoS of Internet Audio Mechanisms via Formal Methods

Alessandro Aldini and Roberto Gorrieri and Marco Roccetti
Università di Bologna
and
Marco Bernardo
Università di Torino

We compute and compare the quality of service (QoS) of three soft real time applications for audio transmission over the Internet. The main metric we consider to capture the user perception of audio is the average packet audio playout delay vs. the packet loss rate. Other metrics we take into account are the packet loss rate vs. the receiving buffer capacity, the lateness of discarded packets vs. the average packet audio playout delay, and the waiting time in the receiver buffer for the played packets vs. the average packet audio playout delay. The study is conducted on formal descriptions of the three audio mechanisms expressed in the process algebraic language EMPA, which are analyzed via simulation through the software tool TwoTowers under various (experimentally obtained or randomly generated) traffic conditions. The stochastic process algebra EMPA has been used because of its compositional support to system modeling, its adequacy to allow functional properties of systems to be formally verified (unlike conventional simulators), and its capability of representing generally distributed durations (which come into play in the three audio mechanisms). The comparison reveals that neither of the three mechanisms outperforms the other two in general, as their performance depends on the traffic conditions.

Categories and Subject Descriptors: C.3 [**Computer Systems Organization**]: Special-Purpose and Application-Based Systems; I.6.0 [**Computing Methodologies**]: Simulation and Modeling—*general*

General Terms: Languages, Performance

Additional Key Words and Phrases: Internet audio mechanisms, quality of service, stochastic process algebras, discrete event simulation, software tools, case studies

This paper is a full version of: A. Aldini, M. Bernardo, R. Gorrieri, M. Roccetti, *"A Simulative Analysis of Internet Audio Mechanisms Using Formal Methods"*, in Proc. of the *11th European Simulation Symp. (ESS 1999)*, SCS International, pp. 281-288, Erlangen (Germany), 1999.
Name: Alessandro Aldini, Roberto Gorrieri, Marco Roccetti.
Affiliation: Dipartimento di Scienze dell'Informazione.
Address: Mura Anteo Zamboni 7, 40127 Bologna, Italy.
Name: Marco Bernardo.
Affiliation: Dipartimento di Informatica.
Address: Corso Svizzera 185, 10149 Torino, Italy.

1. INTRODUCTION

The value of integrating voice and data networks onto a common platform is well known [Kostas et al. 1998]. Traditional telecommunication companies have proposed ATM as a standard for upgrading the Internet to provide both real time and data services. In contrast, it has been empirically demonstrated that voice services may be profitably added to traditional IP networks that were originally designed for data transmission alone [Kostas et al. 1998]. Since the IP community has not considered the provision of QoS guarantees with the same intensity as the ATM community – the current Internet service model offers a flat, classless, best-effort service to its users – the feasibility and the expected QoS of audio applications over IP networks have to be carefully examined, if we wish those applications be successful. Many are the factors that affect the QoS for real time audio applications over the Internet such as, e.g., codec, access, operating system, and sound card delays, and none of them can be ignored. Conversely, among the many possible metrics that influence the user perception of audio, probably the most important one is represented by the average packet audio playout delay vs. the packet loss rate. Other relevant metrics are the receiving buffer capacity vs. the packet loss rate, the average packet audio playout delay vs. the lateness of discarded packets, and the waiting time in the receiver buffer for the played packets vs. the average packet audio playout delay.

In this paper we consider such metrics in order to compare the QoS of three audio protocols [Ramjee et al. 1994; Moon et al. 1998; Roccetti et al. 1999b] via simulation using both experimentally obtained delay measurements of several 5 min long audio conversations between two different Internet sites (trace driven simulation) and two 1 min long audio conversations randomly generated according to Gaussian distributed delays and exponentially distributed delays, respectively. The simulation conditions above have been chosen because they provide a reasonable large spectrum of distributions, hence offering a standard, cheap alternative to network traffic models (e.g., statistic self-similarity [Leland et al. 1994]) that more precisely reveal the fractal like behavior of network traffic, as motivated in Sect. 5.

Originally, each mechanism has been devised and experimented for complementary network conditions (e.g. different sets of experimental traces), so that a precise comparison among them was not possible. In this paper, we have simulated each mechanism under the same network conditions, obtaining full agreement with the experimental results when available [Ramjee et al. 1994; Moon et al. 1998; Roccetti et al. 1999b], but also allowing us to compare precisely the three mechanisms.

The novelty of our approach to the study of the QoS of audio mechanisms for the Internet is the adoption of formal methods, in particular an executable specification language extended with performance modeling capabilities, called EMPA [Bernardo 1999]. The semantics for EMPA is precisely defined in such a way that the resulting performance model is a (continuous time or discrete time) Markov chain, that can be analyzed also via simulation. The pure Markovian calculus has been recently extended to allow data driven computations and generally distributed durations to be modeled, which is badly needed for the specification of the three audio mechanisms at hand. For this case study we use the EMPA sublanguage generating discrete time Markov chains, where arcs are annotated with expressions of general distributions to be evaluated during the simulation analysis. Hence, the underlying simulation model for each of the three audio mechanisms is a generalized semi-Markov process like model derived from the EMPA specification. Since the theory developed in [Bernardo 1999] has been mechanized, resulting in the tool TwoTowers [Bernardo et al. 1998a] that is being profitably used in the functional and performance analyses of several case studies – ranging from distributed algorithms (e.g., mutual exclusion protocols [Bernardo 1999]) to communication systems (e.g., ATM switches [Aldini et al. 1999a]) – EMPA specifications can be automatically analyzed.

An approach based on such a formal technique may be advantageous w.r.t. conventional simulators routinely used by network engineers for the following reasons:

—*Simplicity*: complex protocols are described with a high level specification notation, which simplifies the modeling process.

—*Modularity*: a specification is developed in a modular, top-down way, which makes the updating and reuse of single components easier.

—*Scalability*: adding new components or upgrading the parameters of existing components can be done smoothly.

—*Verification support*: well known formal verification techniques can be applied to such specifications to investigate functional properties, e.g. deadlock freedom.

This paper is organized as follows. Sect. 2 recalls the main features of the three audio mechanisms. Sect. 3 introduces the basics of EMPA and an overview of the tool TwoTowers, whereas the EMPA specifications of the audio mechanisms are presented in Sect. 4. The reader not interested in understanding the details of the specifications can safely skip Sect. 3 and 4. The results of the (functional and) simulative analysis of the three mechanisms are reported in Sect. 5, where the effects that playout delays, loss rates, buffer dimensions, and others metrics may have on the QoS are precisely shown. Finally, in Sect. 6 some conclusions are drawn.

2. PACKET AUDIO OVER THE INTERNET

A typical packetized audio segment may be considered as being constituted of talkspurt periods, during which the audio activity is carried out, and silence periods, during which no audio packet is generated. During a packet audio connection over the Internet, 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. Thus, to transport audio over a non-guaranteed, packet switched network like the Internet, usually audio samples are encoded (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. A symmetric scheme is used in the other direction for interactive conversation.

Internet audio services must operate in a bandwidth-, delay-, and packet loss-constrained environment. This environment has been passed down to the codec development efforts of the ITU (International Telecommunication Union). Recently, several codecs have been designed that work well in the presence of the scarce network bandwidth constraint. For example, the ITU codecs G.729 and G.723.1 [ITU

1996] have been designed for transmitting audio data at bit rates ranging from 8 Kbps to 5.3 Kbps. However, available network bandwidth is not the only requirement to meet for quality audio. Indeed, each step in the audio data flow pipeline mentioned above (from coding to transport, to reception, to decoding) adds delay to the overall transmission. While some delays are relatively fixed (such as coding and decoding), others depend on network conditions. For example, very high average packet delay and packet delay variance (known as jitter) on the order of 500/1000 ms may be experienced over many congestioned Internet links. As a consequence, packet loss percentages, due to the effective loss and damage of packets as well as belated arrivals, may vary between 15% and 40% [Mauth 1998]. User studies indicate that telephony users find round trip delays greater than about 300 ms more like a half duplex connection than a real time conversation. However, user tolerance of delays may vary significantly from application to application. The most critical users may require delay of less than 200 ms, but more tolerant users may be satisfied with delays of 300-500 ms. Finally, many experimental tests have revealed that random independent packet loss rates of up to 10% have little impact on speech recognition [Kostas et al. 1998].

The most used approach in order to ameliorate the effect of jitter and to improve the quality of the audio service over the Internet is to adapt the applications to the jitter present on the network. Hence, a 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; however, packets delayed past the point at which they are supposed to be played out (the playout point) are effectively lost. As a consequence, a crucial tradeoff exists between the length of the imposed additional quantity of delay (and the depth of the receiver buffer) 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.

In the remainder of this section, we present three different playout delay control mechanisms that have been designed to dynamically adjust the audio packets playout delay (and the receiver buffer depth) of packet audio applications, for compensating the highly variable Internet packet delays. For the sake of brevity, only the relevant features of these three nontrivial mechanisms are reviewed.

2.1 Internet Audio Mechanism #1

In this section we briefly describe the adaptive playout delay adjustment algorithm proposed in [Ramjee et al. 1994] on which several Internet audio tools (such as NeVoT [Schulzrinne 1992], FreePhone [Ramjee et al. 1994], and rat [Hardman et al. 1998]) are based. As mentioned above, a receiving site in an audio application buffers packets and delays their playout time. Such a playout delay is usually adaptively adjusted from one talkspurt to the next one.

In order to implement this playout control policy, the playout adjustment mechanism proposed in [Ramjee et al. 1994] (denoted in the following as mechanism #1) makes use of the two following assumptions:

i) there exists an external mechanism that keeps synchronized the two system clocks at both the sending and the receiving sites (for example, the Internet based Network Time Protocol NTP);

ii) the delays experienced by audio packets on the network follow a Gaussian distribution.

Based on these assumptions, the playout control mechanism #1 works by calculating the playout time p_i for the first packet i of a given talkspurt as

$$p_i = t_i + \hat{d}_i + k \cdot \hat{v}_i$$

where t_i is the time at which the audio packet i is generated at the sending site, $\hat{d}_i$ is the average value of the playout delay (i.e. the average value of the time interval between the generation of previous audio packets at the sender and the time instants in which those packets have been played out at the receiver), $k \in \{1, 2, 4\}$ is a variation coefficient (whose effect can be enforced through shift operations) providing some slack in playout delay for arriving packets (the larger the coefficient, the more packets that are played out at the expense of longer playout delays), and finally $\hat{v}_i$ is the average variation of the playout delay. From an intuitive standpoint, the reported formula 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 [Ramjee et al. 1994]:

$$\begin{aligned} \hat{d}_i &= \alpha \cdot \hat{d}_{i-1} + (1 - \alpha) \cdot n_i \\ \hat{v}_i &= \alpha \cdot \hat{v}_{i-1} + (1 - \alpha) \cdot |\hat{d}_i - n_i| \end{aligned}$$

where constant α (usually set to 0.998) is a weight that characterizes the memory properties of the estimation, while n_i is the total delay introduced by the network (difference between the time at which packet i is received at the receiving site and t_i).

In essence, the playout adjustment mechanism #1 is a linear filter that has revealed to be slow in catching up with change in delays, but is quite good at maintaining a steady state value when the control value $(1 - \alpha)$ is set to be very low. Hence, in order to circumvent its inability to adapt to very large change in transmission delays, the playout adjustment mechanism #1 has been subsequently equipped with a delay spike detection and management mechanism. Several studies, in fact, have indicated the presence of spikes in end-to-end Internet delays. A spike is a sudden, large increase in the end-to-end network delay, followed by a series of audio packets arriving almost simultaneously. In cases where a delay spike spans multiple talkspurts, it is important to quickly react to the delay spike. The method proposed to react to spikes is reported in Table 1 (see [Ramjee et al. 1994] for more details).

Two different modes of operation are used depending on whether a transmission delay spike has been detected or not (the spike detected mode and the normal mode, respectively). For every packet i that arrives at the receiver, the mechanism

```
(1)  n_i = Receiver_Timestamp − Sender_Timestamp;
(2)  IF (mode == NORMAL)
         IF (|(n_i − n_{i−1}| > |v̂| · 2 + 800)
            var = 0; /* detected beginning of spike */
            mode = IMPULSE;
     ELSE
         var = var/2 + |(2 · n_i − n_{i−1} − n_{i−2})/8|;
         IF (var ≤ 63)
            mode = NORMAL; /* end of spike */
            n_{i−2} = n_{i−1};
            n_{i−1} = n_i;
            return;
(3)  IF (mode == NORMAL)
         d̂_i = α · d̂_{i−1} + (1 − α) · n_i;
     ELSE
         d̂_i = d̂_{i−1} + n_i − n_{i−1};
     v̂_i = α · v̂_{i−1} + (1 − α) · |n_i − d̂_i|;
(4)  n_{i−2} = n_{i−1};
     n_{i−1} = n_i;
     return;
```

Table 1. The pseudo-code of the Internet Audio Mechanism #1 with spike detection.

computes the end-to-end network delay n_i, then it checks the current mode (part (2) of the algorithm) and, if necessary, switches it. In particular, if the delay variation between two consecutive packets is larger than some multiple of the current average delay variation $|\hat{v}_i|$ then the algorithm switches to the spike detected mode. The end of a spike is detected in a much more tricky way. The consideration that has been exploited to detect the end of a spike is based on experimental observations of the spike phenomenon and it is as follows. Typically, at the end of a spike, a series of packets arrive one after another, almost simultaneously, at the receiver. The motivation for this fact is that, as packets of a talkspurt are transmitted at regular intervals by the sender, at the end of a spike the last belated packets of that spike arrive simultaneously with subsequent packets that are experiencing progressively smaller delays. In order to take into account this phenomenon, the variable var has been used in the algorithm with an exponentially decaying value that adjusts to the slope of the spike. When this variable has a small value, revealing that there is no longer a significant slope, the normal mode of the algorithm is resumed. During the spike detected mode, the mechanism computes the playout delay to be applied to each packet in the talkspurt on the basis of the most recently observed delay values. Instead, in the normal mode the mechanism operates by calculating the playout delay as described at the beginning of this section (see part (3) of the algorithm).

2.2 Internet Audio Mechanism #2

Another adaptive on-line delay adjustment algorithm based on the same two assumptions as mechanism #1 has been presented in [Moon et al. 1998]. This mechanism, which we denote as mechanism #2, is based on the history experienced during the transmission of the audio samples in a given audio connection. The algorithm tracks the network delays of received audio packets and efficiently maintains delay percentile information. 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 delay is recorded and the distribution of packet delays is updated with each new arrival. When a new talkspurt starts, the algorithm proposed in [Moon et al. 1998] 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, this algorithm detects and 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 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. Internet audio mechanism #2 is fully detailed in [Moon et al. 1998], but, for the sake of completeness, we report it here in Table 2. As already mentioned, also this mechanism operates in two modes. For each packet arrived at the destination, the current operation mode is checked, and one mode is switched into the other, if necessary (lines 1-7). When in normal mode, the delay distribution is updated according to the statements in lines 9-22. The beginning of a spike is detected when a packet arrives with a delay larger than some multiple of the current playout delay. In such a case, the algorithm switches to the spike detected mode. The end of a spike is detected, instead, when the most recently received packet (during a spike) has a delay which is less than some multiple of the playout delay before the current spike. In this case, the operation mode is set back to normal. In the algorithm reported in Table 2 two parameters *head* and *tail* are used (in lines 5 and 2) for detecting the beginning and the end of a spike.

To conclude this section, it is worth mentioning that the authors of [Moon et al. 1998] 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.

2.3 Internet Audio Mechanism #3

Recently, a new mechanism [Roccetti et al. 1999b] (denoted as mechanism #3 in the remainder of the paper) has been designed to dynamically adapt the talkspurt playout delays to the network traffic conditions assuming neither the existence of an external mechanism for maintaining an accurate clock synchronization between the sender and the receiver, nor a specific distribution of the end-to-end transmission delays experienced by the audio packets. This mechanism is at the basis of an Internet audio tool, called BoAT, developed at the University of Bologna [Roccetti et al. 1999a]. 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

```
IF (mode == SPIKE)
    IF ($\hat{d}^i_k$ ≤ tail · old_d) /* the end of a spike */
        mode = NORMAL;
ELSE
    IF ($\hat{d}^i_k$ > head · $\hat{p}_k$) /* the beginning of a spike */
        mode = SPIKE;
        old_d = $\hat{p}_k$; /* save $\hat{p}_k$ to detect the end of a spike later */
    ELSE
        IF (delays[curr_pos] ≤ curr_delay)
            count −= 1;
        distr_fnc[delays[curr_pos]] −= 1;
        delays[curr_pos] = $\hat{d}^i_k$;
        curr_pos = (curr_pos + 1) % w;
        distr_fnc[$\hat{d}^i_k$] += 1;
        IF (delays[curr_pos − 1] ≤ curr_delay)
            count += 1;
        WHILE (count < w · q)
            curr_delay += unit;
            count += distr_fnc[curr_delay];
        WHILE (count > w · q)
            curr_delay −= unit;
            count −= distr_fnc[curr_delay];
```

Table 2. The pseudo-code of the Internet Audio Mechanism #2.

a protocol, 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 (defined in [Roccetti et al. 1999b]) inside the talkspurt themselves, provided that intervening silence periods of sufficiently long duration are exploited for the adjustment.

A simplified description of the mechanism #3 is as follows. When the sender transmits the first packet of an audio talkspurt, the sender timestamps that packet with the value (say C) of the reading of its own clock. When this first packet arrives at the receiver, the receiver also sets its clock to 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 instant when the packet is transmitted. When those subsequent packets arrive at the receiving site, their attached timestamp is compared with the value of the reading of the receiver's clock. If the timestamp attached to the packet is equal to the value of the clock of the receiver, that packet is immediately played out. If the timestamp attached to the packet is greater than the value of the clock of the receiver, 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 value of the receiver's clock. Finally, if the timestamp attached to the packet is less than the value of the clock of the receiver, the packet is simply discarded since it is too late for presentation. However, due to the fluctuant delays in real transmissions, the value of the clocks of the sender and the receiver at a given time instant may differ of

$$T_S - T_R = \Delta$$

where T_S and T_R are, respectively, the readings of the local clocks at the sender and at the receiver, and Δ is a nonnegative quantity ranging between 0, a theoretical lower bound, and Δ_{max}, a theoretical upper bound on the transmission delays introduced by the network between the sender and the receiver.

Hence, a crucial issue of the mechanism is an accurate dimensioning of the playout buffer. Both buffer underflow and overflow may result in discontinuity in the playout process. 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 a subsequent packet arrives with maximum delay (e.g. Δ_{max}). It is possible to show that in this case the subsequent packet (transmitted by the sender when its clock shows a time value equal to X, and consequently timestamped with the value X) may arrive at the receiver too late for playout, precisely when the receiver's clock shows the value given by $X+\Delta_{max}$. This consideration suggests that a practical and secure method for preventing from buffer underflow 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. With this simple modification, the policy guarantees that all the audio packets of the talkspurts that will suffer from a transmission delay not greater than Δ_{max} will be on time for 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 (i.e. Δ_{max}), instead a subsequent audio packet experiences minimum delay (e.g. 0). It is possible to show that in such a case the subsequent audio packet (transmitted at time X) may arrive at the receiving site when the receiver's clock is equal to $X - 2 \cdot \Delta_{max}$. In conclusion, this example dictates that the playout buffer dimension may never be less than the maximum number of bytes that may arrive in an interval of $2 \cdot \Delta_{max}$.

In the mechanism #3, a technique has been devised that is used to estimate an upper bound for the maximum delay transmission. 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 to C and sends immediately back to the sender a response packet with attached the same timestamp C. Upon the receiving 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 is equal to an unknown quantity (say t_0) which may range from a theoretical lower bound of 0 to a theoretical upper bound of RTT. Unfortunately, t_0 is unknown and a rough approximation of this value might result in both playout buffer underflow problems and packet loss due to late arrivals. Based on these considerations, the sender, after having received the response packet from the receiver and calculated the RTT value, sends to the receiver a final installation packet, with attached 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 by its clock at the arrival of the installation packet 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 Δ given by

$$\Delta = T_S - T_R = t_0 + RTT$$

where Δ ranges in the interval $[RTT, 2 \cdot RTT]$ depending on the unknown value of t_0, which, in turn, may range in the interval $[0, RTT]$. In essence, 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 highly fluctuant end-to-end delays experienced over wide area, packet switched networks (like 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.

Hence, each time a new value for RTT is computed by the sender, it may be used by the receiver for adaptively setting the value of its local clock and the playout buffer dimension. This method guarantees that both the introduced additional playout time and the buffer dimension are always proportioned to the traffic conditions. However, it may be not possible to replace on-the-fly at the receiver the current values of its own clock and the dimension of its playout buffer, during a talkspurt. In fact, such a per-packet adaptive adjustment of the synchronization parameters might introduce either gaps inside the talkspurt or even timing collisions between the audio packets. Consequently, the installation of the new synchronization values at the receiver is carried out only during the periods of audio inactivity, when no audio packets are generated by the sender. A formal theorem (with a corresponding algorithm) is provided in [Roccetti et al. 1999b] that shows that the installation of a synchronization may be conducted during a silence period (detected by the sender) without introducing either gaps or packet collisions inside the talkspurts of the audio conversation, only if the silence period is equal to or greater than a given known value.

Another problem with the mentioned policy is related to the possible high value for the obtained RTT that may be caused by the fact that either the probe packet or the response packet (during the first two phases of the synchronization operation) has suffered from a very high delay spike. Due to that, a very high value for the playout delay would be introduced, thus impairing the interactivity of the audio conversation. This problem has been solved in [Roccetti et al. 1999b] by adopting a policy which was inspired by the delay spike detection and management mechanism proposed in [Ramjee et al. 1994].

The proposed policy works by using two different modes of operation. 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 $(k < 1)$, as detailed in [Roccetti et al. 1999b]. Let us denote by $\overline{RTT}$ such a smoothed value obtained with the a spike-affected synchronization. Each audio packet that arrives at the receiver after a spike-affected synchronization has been installed is played out after a playout delay value calculated on the basis of one such smoothed $\overline{RTT}$ value.

A high level description of the policy used to detect a playout delay spike is as follows. That policy is based on the comparison between the RTT values obtained from two consecutive synchronization activities. For each most recently computed RTT value, the algorithm checks the current mode and, if necessary, switches its mode to the other one. More precisely, if the most recently computed RTT value is larger than some multiple of the previously measured RTT value then a delay spike is considered to be detected, and the algorithm switches to the spike-detected mode. In such spike-detected mode, the smoothed value $\overline{RTT}$ is computed and then used for playout. The end of a spike is detected similarly. If the most recently computed RTT value is smaller than some multiple of the RTT value experienced when the spike-affected period began, the mode is reset to normal. Clearly, when the algorithm operates in the normal mode, the normal RTT value is used at the receiver for calculating the playout delay.

3. EMPA/TWOTOWERS TECHNOLOGY

To investigate the QoS of the three audio mechanisms introduced in the previous section, we have employed the EMPA/TwoTowers technology [Bernardo 1999; Bernardo et al. 1998a]. TwoTowers is a software tool implementing the integrated approach of [Bernardo et al. 1998b] that combines the capabilities of verifying functional properties of systems and predicting their performance since the early stages of their design. This is achieved in TwoTowers by analyzing systems formally described with the process algebraic language EMPA, which allows for the high level, compositional modeling of functional and performance aspects of complex systems. Once the EMPA specification of the system under study is provided, TwoTowers translates it into a formally defined, low level, semantic model which can be a state transition graph or a Markov chain, depending on the kind of analysis that must be performed. To analyze such low level models, TwoTowers invokes other tools tailored for that purpose.

In this section, besides a presentation of the syntax of the specification language and the architecture of the tool, we concentrate on features such as general distributions and simulation that we have exploited in our case study. The reader interested in the EMPA specifications of the three mechanisms (Sect. 4) should not skip Sect. 3.1 and 3.2.

3.1 The Specification Language EMPA

To the aim of this case study, we consider a subcalculus of EMPA where the specifications generate discrete time Markov chains only. In order to make the reader familiar with the language, we present as an example an abstraction of the scenario in which the audio mechanisms of Sect. 2 operate. Consider a producer/consumer system with a buffer of capacity two. The overall system, called $PCSystem$, can be described as the interaction of a producer, a buffer, and a consumer:

$$Producer \,\|_{\{produce\}}\, Buffer \,\|_{\{consume\}}\, Consumer$$

EMPA supports compositional modeling, in order to develop the algebraic representation of a system in a modular way; subsystems can be modeled separately, then combined using the appropriate operators. In particular, as far as our example is concerned, the overall model *PCSystem* consists of three subterms composed in parallel, called *Producer*, *Buffer*, and *Consumer*. The parallel composition operator "$\_ \|_S \_$" is used to express a set of concurrent system components and their communication interfaces. In our example, *Producer* interacts with *Buffer* via action *produce*, whereas *Consumer* interacts with *Buffer* via action *consume*. The exact meaning will be clarified later.

The *Consumer* repeatedly consumes items (if available):

$$Consumer \stackrel{\Delta}{=} {<}consume_{l,w_3}{>}.Consumer$$

As we can see, EMPA specifications are given as sets of equations of the form $A \stackrel{\Delta}{=} E$, where A is the process name and E is an algebraic term built out of several different operators. Terms of EMPA specifications are formed by actions (describing system activities). Every action ${<}\alpha_{l,w}{>}$, called active action, is composed of the type α of the action (e.g. message consumption), a priority level l, and a weight w. Term *Consumer* executes action *consume* and afterwards it behaves as term *Consumer* itself. In EMPA recursive equations of this form are typically employed to describe repetitive behaviors.

The *Producer* repeatedly produces new items:

$$Producer \stackrel{\Delta}{=} {<}produce_{l,w_1}{>}.Producer + {<}\tau_{l,w_2}{>}.Producer$$

Among active actions, we designate one of them as internal, i.e. not visible by an external observer, and we denote its type by τ. Invisible actions are useful to abstract from unnecessary internal details of a system.

In the definition of the producer we have used the basic operator "$\_+\_$", summation. Process $E_1 + E_2$ behaves either like E_1 or like E_2; as soon as one performs its first action the other is discarded. If both alternatives are permitted, then the choice is resolved according to the preselection policy: the active actions with lower priority are discarded, then each of the remaining actions is given an execution probability proportional to its weight. In our example, term *Producer* executes action *produce* with probability $w_1/(w_1 + w_2)$ and the internal action τ (in this case we mean that the producer is not ready to produce a message, so from an external observer viewpoint, it stays idle) with probability $w_2/(w_1 + w_2)$. We adopt the same policy for the choice between active actions of different terms composed in parallel. Moreover, process $E_1 \|_S E_2$ asynchronously executes actions of E_1 or E_2 whose type does not belong to S and synchronously executes actions of E_1 and E_2 of the same type belonging to S.

The *Buffer*, instead, is at any time ready to accept new incoming items (if not full) and to deliver previously produced items (if not empty):

$$\begin{aligned}
Buffer &\stackrel{\Delta}{=} {<}produce_*{>}.Buffer_1 \\
Buffer_1 &\stackrel{\Delta}{=} {<}produce_*{>}.Buffer_2 + {<}consume_*{>}.Buffer \\
Buffer_2 &\stackrel{\Delta}{=} {<}consume_*{>}.Buffer_1
\end{aligned}$$

Term *Buffer* represents the empty buffer (it permits to accept messages only); term $Buffer_1$ represents the buffer with a message and an empty place (it can accept a new produced message or deliver the present message); finally, term $Buffer_2$ represents the full buffer (it permits to deliver messages only).

Note that we have used a new kind of action, called passive action, whose subscript is $*$ (instead of a priority level and a weight). Passive actions are underspecified in the sense that the choice among two passive actions is purely nondeterministic. This reflects the fact that the interactions of term *Buffer* are guided by the other two processes. Indeed, a passive action is underspecified until it is synchronized with an active action. In particular, in a binary synchronization at least one of the two actions must be passive and the subscript of the resulting action is determined by the other action. As a consequence, note that in case of multi-way synchronization, at most one active action can be involved and it determines the resulting action; all the other involved actions must be passive. This reflects a master-slaves discipline, where a master decides the action to execute and the slaves react to its decision.

In Fig. 1 we report the interleaving semantic model of *PCSystem*, which is automatically generated by applying the formal semantic rules in [Bernardo 1999] to the specification. Such a model is essentially a state transition graph (with one state denoted as initial), where states are in correspondence with terms and transitions are labeled with actions. In the figure, the initial state s_0 corresponds to *PCSystem*, state s_1 corresponds to

$$Producer \,\|_{\{produce\}}\, Buffer_1 \,\|_{\{consume\}}\, Consumer$$

and state s_2 corresponds to

$$Producer \,\|_{\{produce\}}\, Buffer_2 \,\|_{\{consume\}}\, Consumer$$

According to the weights associated with the active actions, we point out that in state s_1 the related term executes action *produce* with probability $w_1/(w_1+w_2+w_3)$, action *consume* with probability $w_3/(w_1+w_2+w_3)$, and the internal action τ with probability $w_2/(w_1+w_2+w_3)$. Note that the choice is purely probabilistic and the probabilities of the executable actions sum up to 1. Finally, in state s_2, the related term executes action *consume* with probability $w_3/(w_2+w_3)$ and the internal action τ with probability $w_2/(w_2+w_3)$. From graphs like this, a functional semantic model (or a performance semantic model in the form of a discrete time Markov chain) can be automatically derived by dropping action probabilities and priorities (or action types).

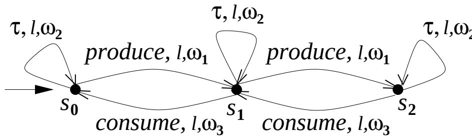


Fig. 1. Interleaving semantic model of *PCSystem*.

The language characteristics we have presented are insufficient for modeling the three audio mechanisms of Sect. 2. Indeed, processes of complex systems frequently need to exchange information during a synchronization (e.g. message packets and timestamps), and then they need to decide their behavior depending on the value of such information. Hence, we now present the capabilities of EMPA allowing for value passing among processes.

As far as types are concerned, an action can be unstructured (e.g. all the actions of the above example), carry values (if active), or accept values (if passive). In the second case, we talk about an output action whose type is denoted $a!(\underline{e})$ where $\underline{e}$ is a vector of expressions, while in the third case we talk about an input action whose type is denoted $a?(\underline{x})$ where $\underline{x}$ is a vector of variables. Input/output actions are used to model information exchange among system components which synchronize when performing certain activities. As an example, consider *PCSystem* in the case the producer specifies the type of the message sent to the buffer. The synchronization between the unstructured actions $produce_{l,w_1}$ and $produce_*$ is not sufficient to express this additional information exchange, so we use the input/output actions $produce_{l,w_1}!(class)$ and $produce_*?(x)$, where *class* is a parameter denoting the kind of message and x is a variable used to get values from the corresponding output action. During the synchronization between the producer and the buffer, the latter process reads in x the value *class* transmitted by the former process. In particular, process $<produce_{l,w_1}!(class)>.Producer \,\|_{\{produce\}} <produce_*?(x)>.Buffer_1$ executes the synchronizing action of type *produce* and afterwards behaves as *Producer* $\|_{\{produce\}}$ $Buffer_1\{class/x\}$, where parameter *class* substitutes for the occurrences of x in term $Buffer_1$.

A new choice operator is introduced to manage alternative actions depending on the value of a certain expression: process "if (β) then E_1 else E_2" behaves as either E_1 or E_2 depending on whether boolean expression β evaluates to true or not. Unlike the previous choice operator, this operator expresses a data driven choice as the behavior is determined by the value of the variables involved in expression β.

As an example, consider the value passing version of *PCSystem* as described above, and suppose that the buffer discards the produced messages whose class is equal to 1. The body of *Buffer* becomes as follows:

$$<produce_*?(x)>.(\text{if } (x = 1) \text{ then } Buffer \text{ else } Buffer_1)$$

Finally, to store values EMPA processes are defined through equations of the form

$$A(type_1\, f\_par_1, \ldots, type_n\, f\_par_n; type_{n+1}\, l\_var_1, \ldots, type_{n+m}\, l\_var_m) \stackrel{\Delta}{=} E$$

where formal parameters f_par_i and local variables l_par_i are declared. Formal parameters are bound to actual parameters whenever an invocation of the form $A(a\_par_1, \ldots, a\_par_n)$ occurs. Local variables, instead, are used to get values from other system components via synchronization. Both formal parameters and local variables are typed. The supported types are integer, real, boolean, list, and array.

3.2 General Distributions via Value Passing

The three audio mechanisms we shall model with EMPA in the next section are part of a scenario where generally distributed durations come into play: think e.g. of packet transmission delays. As we said, with the subcalculus of EMPA we consider, only active and passive actions can be modeled, which implies that only discrete time Markov chains can be generated. However, value passing based expressions can be used to syntactically specify arbitrary probability distributions, and this can be exploited in order to overcome the limitation above. To this aim, we proceed as follows from the modeling viewpoint. Clock variables are employed in the specification in order to mark the discrete time steps. Since only active (and passive) actions are present in the specification, timed events are treated by means of suitable list typed variables which store the occurrence times (denoted by appropriate expressions) of the timed events. Clock variables are used in the specification to detect when the occurrence time of a timed event has come, in order to enable the particular action representing the occurrence of the event itself.

As an example, consider the following two terms representing a simplification of the description of the channel between the sender and the receiver for any of the audio mechanisms described in Sect. 2 which involves Gaussian delays:

$$
\begin{array}{l}
\mathit{Channel}(\mathrm{int}\ \mathit{clock}, \mathrm{list}(\mathrm{int})\ \mathit{packet\_l};\,) \triangleq \\
\quad <\mathit{prepare\_packet}_*>.\mathit{DeliveryCheck}(\mathit{clock}, \mathrm{insert}([\mathit{clock}+\mathrm{gauss}(100{,}7)], \mathit{packet\_l})) \\
\quad + <\mathit{elapse\_tick}_{1,1}>.\mathit{DeliveryCheck}(\mathit{clock}, \mathit{packet\_l}) \\
\mathit{DeliveryCheck}(\mathrm{int}\ \mathit{clock}, \mathrm{list}(\mathrm{int})\ \mathit{packet\_l};\,) \triangleq \\
\quad \mathrm{if}\ (\mathit{packet\_l} \neq [\,] \wedge \mathrm{first}(\mathit{packet\_l}) = \mathit{clock})\ \mathrm{then} \\
\quad\quad <\mathit{trans\_packet}_{1,1}>.\mathit{DeliveryCheck}(\mathit{clock}, \mathrm{tail}(\mathit{packet\_l})) \\
\quad \mathrm{else} \\
\quad\quad \mathit{Channel}(\mathit{clock}+1, \mathit{packet\_l})
\end{array}
$$

Both constants *Channel* and *DeliveryCheck* have two formal parameters: *clock*, which marks the passage of time, and *packet_l*, the list of packets that are flowing along the channel. Every packet is described through the only piece of information which is relevant for the channel, i.e. the time at which the packet will reach the receiver. If we consider *Channel*, we see that at each time instant, either the channel receives a packet from the sender (action $<\mathit{prepare\_packet}_*>$) or nothing happens (action $<\mathit{elapse\_tick}_{1,1}>$). In the former case, assuming network delays following a Gaussian distribution with expected value 100 ms and standard deviation 7 ms, the delivery time scheduled for the arrived packet is inserted into *packet_l* by preserving the order of its elements. Afterwards, the control is passed to *DeliveryCheck* which delivers to the receiver all the packets in *packet_l* whose delivery time equals *clock* (action $<\mathit{trans\_packet}_{1,1}>$), then increments *clock* and passes the control back to *Channel*.

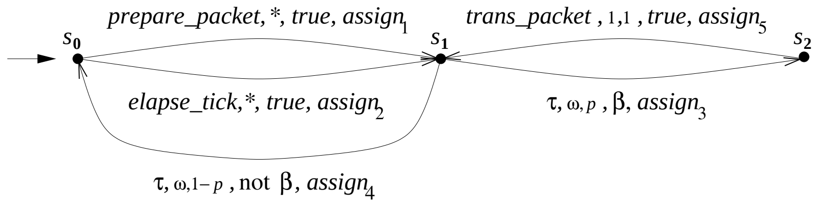


Fig. 2. Interleaving semantic model of *Channel*.

From the semantic viewpoint, value passing is managed by resorting to symbolic state transition graphs. Simply put, the model underlying an EMPA specification with value passing based general distributions is a generalized semi-Markov process like model. In the former model (i.e., the model underlying an EMPA specification with value passing based general distributions) the passage of time is described through both state variables and the related assignments labeling the transitions, whereas in the latter model (i.e., a generalized semi-Markov process) the time progress is kept by explicit clocks. Every state of our model is thought of as having a set of free variables (corresponding to formal parameters and local variables) set by transition labeling assignments which come from input/output synchronizations and parameterized constant invocations, e.g. $\mathit{Channel}(\mathit{clock}+1, \mathit{packet\_l})$.

Moreover, each transition is labeled with a boolean guard enabling/disabling the transition according to the boolean expressions of possible conditional operators. As an example, we report in Fig. 2 the interleaving semantic model of *Channel*. The initial state s_0 corresponds to *Channel*, state s_1 corresponds to *DeliveryCheck*, and state s_2 corresponds to term $<\mathit{trans\_packet}_{1,1}>.$
$\mathit{DeliveryCheck}(\mathit{clock}, \mathrm{tail}(\mathit{packet\_l}))$. The transition labeling assignments are as follows:

$$\begin{aligned}
\mathit{assign}_1 &= \{\mathit{DeliveryCheck.clock} := \mathit{Channel.clock}, \\
&\quad\ \mathit{DeliveryCheck.packet_l} := \mathrm{insert}([\mathit{Channel.clock} + \mathrm{gauss}(100, 7)], \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{Channel.packet_l})\} \\
\mathit{assign}_2 &= \{\mathit{DeliveryCheck.clock} := \mathit{Channel.clock}, \\
&\quad\ \mathit{DeliveryCheck.packet_l} := \mathit{Channel.packet_l}\} \\
\mathit{assign}_3 &= \emptyset \\
\mathit{assign}_4 &= \{\mathit{Channel.clock} := \mathit{DeliveryCheck.clock} + 1, \\
&\quad\ \mathit{Channel.packet_l} := \mathit{DeliveryCheck.packet_l}\} \\
\mathit{assign}_5 &= \{\mathit{DeliveryCheck.clock} := \mathit{DeliveryCheck.clock}, \\
&\quad\ \mathit{DeliveryCheck.packet_l} := \mathrm{tail}(\mathit{DeliveryCheck.packet_l})\}
\end{aligned}$$

Note that the internal transitions leaving state s_1 translate the conditional operator in the specification. If we denote by β the boolean expression

$$\mathit{DeliveryCheck.packet\_l} \neq [\,] \wedge \mathrm{first}(\mathit{DeliveryCheck.packet\_l}) = \mathit{DeliveryCheck.clock}$$

the transition to s_2 (then-branch) has guard β, while the transition to s_0 (else-branch) has guard $\neg\beta$. Both transitions are internal, and have the highest priority level of the system (denoted by ω) and a weight depending on p, representing the probability that β holds (necessary only for the computation of the underlying discrete time Markov chain).

It is worth noting that, in the case of EMPA specifications where generally distributed durations come into play, the discrete time Markov chains that would be obtained (from models like that of Fig. 2) by keeping only weight based probabilities lack information about the value of the durations. Therefore, an exact performance analysis carried out on such Markov chains would be meaningless and we have to resort to a simulative analysis of performance conducted on the semantic model, so that durations encoded within assignments are taken into account.

As an example, consider at simulation time the execution of the transition from s_1 to s_0. In s_1, the state variables *DeliveryCheck.clock* and *DeliveryCheck.packet_l* have some specific values. According to such values, the boolean expression β above is evaluated. If β holds false, which is the precondition in order for the considered transition to be executed, either *DeliveryCheck.packet_l* is empty or the first packet in such a list has a delivery time greater than the current clock value. This means that no packet currently flowing along the channel reaches destination in this time unit, so we have to go back to state s_0. In doing so, the state variables of s_0 must be suitably updated according to assign_4. More precisely, *Channel.clock* is given by *DeliveryCheck.clock* incremented by one (one time unit has elapsed), while *Channel.packet_l* inherits the same contents as *DeliveryCheck.packet_l*.

3.3 TwoTowers: Architecture and Functionalities

EMPA has a companion software tool called TwoTowers [Bernardo et al. 1998a], which is composed of 20,000 lines of code divided into a graphical user interface, a tool driver, an integrated kernel, a functional kernel, and a performance kernel.

The graphical user interface allows the user to edit EMPA specifications, compile them, and run the various analysis routines we are going to present. The tool driver is essentially a parser for the EMPA specifications. If a specification is correct the control is passed to one of the three kernels for analysis.

The integrated kernel contains the routine to generate the integrated interleaving semantic model of correct, finite state EMPA specifications according to the formal semantics. As said above, the integrated interleaving semantic model is a state transition graph where states are in correspondence with EMPA terms and transitions are labeled with action types, action priorities and weights, boolean guards, and assignments, as in Fig. 2. The integrated kernel contains the routines for analysing integrated interleaving semantic models. More precisely, it is possible to verify whether two such models possess the same functional and performance properties (integrated equivalence checking), as well as to conduct a simulative analysis.

The functional kernel contains the routine to generate the functional semantic model (a state transition graph where transitions are labeled only by action types) from the integrated interleaving semantic model. The functional kernel is interfaced with the Concurrency Workbench of North Carolina (CWB-NC) [Cleaveland and Sims 1996], thereby inheriting its analysis capabilities, e.g. model checking in the μ-calculus or CTL [Clarke et al. 1986].

The performance kernel contains the routine to generate the performance semantic model, a discrete time Markov chain which can be thought of as a state transition graph where transitions are labeled only by action weights. The performance kernel is interfaced with the Markov Chain Analyzer (MarCA) [Stewart 1994] to derive reward based performance measures.

3.4 Simulation with TwoTowers

In TwoTowers the simulation of EMPA specifications is carried out according to the method of independent replications [Welch 1983]. At each step of each simulation run, the transitions for the current state are computed according to the formal semantics and one of them (together with the related derivative state) is chosen according to the preselection policy. Pseudo random number generators [Shedler 1983] are employed to solve the choice. In particular, after partitioning $\mathbb{R}_{[0,1]}$ in as many subintervals as there are enabled active transitions in the current state, where the length of each subinterval is proportional to the weight of the corresponding enabled active transition, a pseudo random number generator is used to sample a number uniformly distributed in $\mathbb{R}_{[0,1]}$ which determines the active transition to execute depending on the subinterval to which it belongs. Since at any time during the simulation only the current state of the integrated interleaving semantic model is needed, we are not required to build the whole state space a priori. Providing the outgoing transitions for the current state without having the whole state space generated is naturally supported as the semantics for EMPA is defined in an operational style. Therefore, with TwoTowers simulation can in general be employed to study the performance of specifications with a huge or even infinite state space. The simulative analysis of an EMPA specification requires the provision of additional information consisting of the indication of the termination condition, the specification of the performance measures of interest, and possible trace files to be used in case of trace driven simulation (in such a case a trace file has to be specified from which values must be taken during simulation instead of being produced by pseudo random number generators). The expected values of the measures of interest are estimated together with the related 90% confidence interval. As an example, the termination condition is of the form

$$a\, n_1\, n_2$$

where a is an action type occurring in the EMPA specification, $n_1 \geq 1$ is the number of times a must be executed before terminating each simulation run, and $1 \leq n_2 \leq 30$ is the number of simulation runs.

4. SPECIFYING AN INTERNET AUDIO MECHANISM WITH EMPA

The three adaptive playout delay adjustment mechanisms informally described in Sect. 2 have been formally modeled with EMPA in order to compare their performance with TwoTowers. Resorting to value passing has been necessary to cope with each data driven computation (think, e.g., of the different activities to undertake depending on whether the timestamp of a message is greater than the value of the clock at the receiving site) and delay distributions (such as Gaussian network delays). For the sake of conciseness, we now present a detailed description of the algebraic specification of mechanism #1, and then we will briefly describe the characteristics of the other models. See [Aldini et al. 1999b] for a full description.

Let us now show the overall model of the audio mechanism #1 in the case of spike detection deactivated. We model with separate terms the various components of the playout adjustment mechanism: a sender site, the channel, and a receiver site. All components must periodically synchronize with a clock representing the time progress.

—$\mathit{Spec\#1} \stackrel{\Delta}{=} \mathit{Clock}_1 \|_{T_1} (\mathit{Sender}_1(0) \|_{SC_1} \mathit{Channel}_1(0,[\,]) \|_{RC_1} \mathit{Receiver}_1(0,[\,],d_0,v_0,0))$

—$T_1 = \{\mathit{elapse\_tick}\}$

—$SC_1 = \{\mathit{elapse\_tick},\ \mathit{prepare\_packet}\}$

—$RC_1 = \{\mathit{elapse\_tick},\ \mathit{trans\_packet}\}$

It is worth observing the synchronization actions *prepare_packet*, between the sender site and the channel, denoting the fact that a packet is put on the channel, and *trans_packet*, between the channel and the receiver site, denoting the reception of a packet at the receiver site. The clocks for the sender and the receiver sites must have the same initial value (which we have chosen to be 0). The other parameters for the receiver are the list of the received packets (ordered on the playout instant), the parameters d and v of the stochastic gradient algorithm, and the playout delay for packets of the current talkspurt. The second parameter in $\mathit{Channel}_1$ is the list of packets currently flowing along the channel. In each term but Clock_1 occurs a passive action of type *elapse_tick*, allowing for a multiway synchronization with term $Clock_1$. It endlessly performs action of type *elapse_tick* on which every component must synchronize whenever no other action logically belonging to the same time unit can occur.

—$Clock_1 \triangleq \, <\!elapse\_tick_{1,1}\!>.Clock_1$

The priority level of action $<\!elapse\_tick_{1,1}\!>$ is the lowest in the model, so such an action is executed only whenever no other action can.

Let us now specify the model for the channel, similar to the one seen in Sect. 3.2. The channel accepts packets from the sender site, via synchronization on action of type *prepare_packet*. Packets are queued in a buffer according to their delivering time; during a time unit, if the first packet of the buffer is ready, the channel transmits it to the receiver site, via synchronization on action of type *trans_packet*.

—$Channel_1(\text{int } clock\_c, \text{list(list(int))}\ packet\_l; \text{int } timestamp, \text{int } fp) \triangleq$
$\quad <\!prepare\_packet?(timestamp, fp)_*\!>.$
$\quad\ DeliveryCheck_1(clock\_c, \text{insert}([clock\_c + \text{gauss}(100,7), timestamp, fp], packet\_l))$
$\quad + <\!\tau_{2,1}\!>.DeliveryCheck_1(clock\_c, packet\_l)$

—$DeliveryCheck_1(\text{int } clock\_c,\ \text{list(list(int))}\ packet\_l;) \triangleq$
$\quad \text{if } (packet\_l \neq [\,] \wedge \text{first}(\text{first}(packet\_l)) \leq clock\_c) \text{ then}$
$\quad (<\!trans\_packet!(\text{tail}(\text{first}(packet\_l)))_{2,1}\!>.DeliveryCheck_1(clock\_c, \text{tail}(packet\_l))$
$\quad\ + <\!elapse\_tick_*\!>.Channel_1(clock\_c + 1, packet\_l))$
$\quad \text{else}$
$\quad\ <\!elapse\_tick_*\!>.Channel_1(clock\_c + 1, packet\_l)$

It is worth observing the transmission delay introduced following the particular general distribution specified in term $Channel_1$; in case of exponential traffic, expression gauss(100, 7) is replaced by expression exp(0.01). In case of trace driven traffic, the same expression represents the reading of a (transmission delay) value from the trace file. Following the priority level of the actions, the activity of the channel consists of accepting audio packets from the sender (input action *prepare_packet*) if any is ready (otherwise the internal action allows for calling term $DeliveryCheck_1$), sampling their delivering time (computed as $clock\_c + \text{gauss}(100, 7)$), queueing them in an unbounded buffer (called *packet_l*) according to their delivering time, and finally delivering them at the scheduled time (output action *trans_packet*) in term $DeliveryCheck_1$.

As far as the sender site is concerned, the transmission alternates talkspurt periods, varying from 1 to 4 seconds, and silence periods whose duration is 30% of the last talkspurt. The choice of the duration of the next talkspurt is probabilistic (in particular, the four possible durations have the same probability to be chosen) and internal (specified by means of an alternative choice among τ actions), because no other term has to be involved. The model is as follows:

—$Sender_1(\text{int } clock\_s;) \triangleq$
$\quad <\!\tau_{3,1}\!>.FirstPacketGen_1(clock\_s, clock\_s + 20, clock\_s + 1000, 300)\ +$
$\quad <\!\tau_{3,1}\!>.FirstPacketGen_1(clock\_s, clock\_s + 20, clock\_s + 2000, 600)\ +$
$\quad <\!\tau_{3,1}\!>.FirstPacketGen_1(clock\_s, clock\_s + 20, clock\_s + 3000, 900)\ +$
$\quad <\!\tau_{3,1}\!>.FirstPacketGen_1(clock\_s, clock\_s + 20, clock\_s + 4000, 1200)$

The priority level of the four internal actions in term $Sender_1$ is the highest of the model (the choice of the talkspurt length has to precede other possible actions). The second parameter of $FirstPacketGen_1$, which is the term called after the τ action, represents the instant of the next transmission (a packet every 20 ms.); the third and fourth parameters compute the end of the new talkspurt (in ms.) and the duration of the following silence period (in ms.), respectively.

The first packet of a given talkspurt has a different mark allowing for a correct computation of the playout point at the receiver site. So this event is separately treated in a suitable term called $FirstPacketGen_1$. The transmission of the subsequent packets is managed in term $PacketGen_1$, where we have to consider also the possible end of the current talkspurt.

—$FirstPacketGen_1(\text{int } clock\_s, \text{int } trans\_time, \text{int } end\_talk, \text{int } idle\_time;) \triangleq$
  if $(clock\_s \geq trans\_time)$ then
   $<prepare\_packet!(clock\_s,1)_{3,1}>.<elapse\_tick_*>.$
    $PacketGen_1(clock\_s + 1, trans\_time + 20, end\_talk, idle\_time)$
  else
   $<elapse\_tick_*>.FirstPacketGen_1(clock\_s + 1, trans\_time, end\_talk, idle\_time)$

Note the output action $<prepare\_packet!(clock\_s,1)_{3,1}>$, where the first parameter is the timestamp associated with the packet and the second parameter is set to 1, in order to inform the receiver site that a new talkspurt is going to start.

—$PacketGen_1(\text{int } clock\_s, \text{int } trans\_time, \text{int } end\_talk, \text{int } idle\_time;) \triangleq$
  if $(clock\_s \geq end\_talk)$ then
   $Idle_1(clock\_s, clock\_s + idle\_time)$
  else
   if $(clock\_s \geq trans\_time)$ then
    $<prepare\_packet!(clock\_s,0)_{3,1}>.<elapse\_tick_*>.$
     $PacketGen_1(clock\_s + 1, trans\_time + 20, end\_talk, idle\_time)$
   else
    $<elapse\_tick_*>.PacketGen_1(clock\_s + 1, trans\_time, end\_talk, idle\_time)$

The four formal parameters of $PacketGen_1$ ($FirstPacketGen_1$) represent the sender clock, the instant of the next transmission, the end of the current talkspurt, and the length of the next idle period, respectively. The output action *prepare_packet* in term $PacketGen_1$ marks each packet with the same value, set to 0, denoting that the packet belongs to the current talkspurt.

Term $Idle_1$ represents the silence periods between talkspurts, as follows:

—$Idle_1(\text{int } clock\_s, \text{int } end\_idle;) \triangleq$
  if $(clock\_s \geq end\_idle)$ then
   $Sender_1(clock\_s)$
  else
   $<elapse\_tick_*>.Idle_1(clock\_s + 1,\ end\_idle)$

Until the end of the silence period the unique action that can occur is *elapse_tick*. According to the priority level of each active action, we can observe the following events in the specified order: the sender chooses the length of the new talkspurt if a new talkspurt is going to start, the sender puts (during a talkspurt) a new packet on the channel, and finally the clock is enabled to begin a new time unit.

As far as the receiver site is concerned, we now show the basic playout adjustment algorithm in absence of the delay spike detection mechanism. When we consider the one equipped with the delay spike detection, terms $Receiver_1$ and the following ones represent the exact counterpart of the pseudo-algorithm of Table 1. Term $Receiver_1$ models the reception of new packets (action of type *trans_packet*) and the following ones model the modification of the parameters d and v (following the formulas of the stochastic gradient algorithm). A new value for the playout delay is computed if the arrived packet is the first of a new talkspurt, then the packet is discarded if the value of its timestamp is smaller than the value of the receiver clock. The accepted packets are buffered according to their playout time, and then played in the suitable instant, as we can see in the following terms.

—$Receiver_1(\mathrm{int}\ clock\_r, \mathrm{list(int)}\ packet\_l, \mathrm{real}\ d, \mathrm{real}\ v, \mathrm{int}\ pd; \mathrm{list(int)}\ arrive\_l) \triangleq$
$<trans\_packet?(arrive\_l), *>.$
$Receiver'_1(clock\_r,\ packet\_l, \alpha \cdot \mathrm{d} + (1-\alpha)\cdot(clock\_r - \mathrm{first}(arrive\_l)),$
$\alpha \cdot \mathrm{v} + (1-\alpha)\cdot|(\alpha\cdot \mathrm{d} + (1-\alpha)\cdot(clock\_r - \mathrm{first}(arrive\_l))) -$
$(clock\_r - \mathrm{first}(arrive\_l))|,\ pd, \mathrm{first}(arrive\_l), \mathrm{first}(\mathrm{tail}(arrive\_l)))$
$+\ <\tau_{1,1}>.PlayoutCheck_1(clock\_r,\ packet\_l,\ d,\ v,\ pd)$

Besides the receiver clock *clock_r* and buffer *packet_l*, the invocation of term $Receiver'_1$ contains the new values of the parameters d and v, the current playout delay (called pd), and the information obtained at the receipt of the packet (i.e. its timestamp and mark) during the synchronization with the channel. The internal action of term $Receiver_1$ occurs only if the channel is not ready to deliver a new packet (its priority level is the lowest of the model), and in this case the receiver directly calls the buffer manager (term $PlayoutCheck_1$).

—$Receiver'_1(\mathrm{int}\ clock\_r, \mathrm{list(int)}\ packet\_l, \mathrm{real}\ d, \mathrm{real}\ v, \mathrm{int}\ pd, \mathrm{int}\ timestamp, \mathrm{int}\ fp;) \triangleq$
if $(fp = 1)$ then
if $(timestamp + d + k\cdot v \geq clock\_r)$ then
$PlayoutCheck_1(clock\_r, \mathrm{insert}(timestamp + d + k\cdot v,\ packet\_l),\ d,\ v,\ d + k\cdot \mathrm{v})$
else
$<discard\_packet_{2,1}>.PlayoutCheck_1(clock\_r,\ packet\_l,\ d,\ v,\ d + k\cdot v)$
else
if $(timestamp + pd \geq clock\_r)$ then
$PlayoutCheck_1(clock\_r, \mathrm{insert}(timestamp + pd,\ packet\_l),\ d,\ v,\ pd)$
else
$<discard\_packet_{2,1}>.PlayoutCheck_1(clock\_r,\ packet\_l,\ d,\ v,\ pd)$

Note that a new value of the playout delay is considered if the current packet is the first of the talkspurt ($fp = 1$). In both cases, the action of type *discard_packet* occurs whenever the playout instant of the received packet precedes the current receiver clock, otherwise the packet is queued in the suitable position of the buffer. The expression $timestamp + pd$ represents the playout instant of the current packet.

For the sake of simplicity, the buffer of the receiver site specified in the model is unbounded; when we consider the buffer capacity as a metric for the simulative analysis, we simply add a control on the length of the buffer before the insert operation in term $Receiver'_1$; if the buffer is already full, exceeding packets are discarded. Term $PlayoutCheck_1$ simply playouts packets, if any is ready, before calling again term $Receiver_1$ and starting a new time unit.

$$
\begin{aligned}
&\text{—}PlayoutCheck_1(\text{int } clock\_r, \text{list(int) } packet\_l, \text{real } d, \text{real } v, \text{int } pd;) \triangleq\\
&\quad \text{if } (packet\_l \neq [\,] \wedge \text{first}(packet\_l) = clock\_r) \text{ then}\\
&\quad\quad <playout\_packet_{2,1}>.PlayoutCheck_1(clock\_r, \text{tail}(packet\_l),\ d,\ v,\ pd)\\
&\quad \text{else}\\
&\quad\quad <elapse\_tick_*>.Receiver_1(clock\_r + 1,\ packet\_l,\ d,\ v,\ pd)
\end{aligned}
$$

Both actions *discard_packet* and *playout_packet* have priority level 2, in order to precede the begin of a new time unit.

As far as audio mechanism #2 is concerned, the sender and channel models are the same as for audio mechanism #1. Terms modeling the receiving site have to manage the temporal window containing the transmission delay of the last w packets and the array representing the number of occurrences of such delays (according to Table 2). The structure of these terms is similar to the one seen for mechanism #1. We only point out that terms in the EMPA specification are the exact counterpart of the pseudo-algorithm of Table 2 (see [Aldini et al. 1999b] for a full description).

As far as audio mechanism #3 is concerned, the following term shows the overall model of the mechanism:

$$\text{—}Spec\#3 \triangleq Clock_1 \,\|_{T_1}\, (ISSender_3(C_s) \,\|_{SC_3}\, ISChannel_3 \,\|_{RC_3}\, ISReceiver_3(C_r))$$

$$
\begin{aligned}
\text{—}SC_3 = \{&elapse\_tick,\ prepare\_packet,\ prepare\_probe,\ trans\_response,\\
&prepare\_install,\ trans\_ack\}
\end{aligned}
$$

$$
\begin{aligned}
\text{—}RC_3 = \{&elapse\_tick,\ trans\_packet,\ trans\_probe,\ prepare\_response,\\
&trans\_install,\ prepare\_ack\}
\end{aligned}
$$

The clocks at the sender and the receiver sites can have different initial values C_s and C_r, because the synchronization is embedded in the algorithm. Talkspurt and silence periods are as in the former mechanism. An initial synchronization phase precedes the first talkspurt. The initial interaction between the two sites is composed of a *prepare_probe* action, denoting the fact that a probe message (i.e. the first message of the three way handshake) is put on the channel, a *trans_probe* action, denoting the reception of a probe message at the receiver, a *prepare_response* action, denoting the fact that a synchronization response is put on the channel, a *trans_response* action, denoting the reception of a synchronization response at the sender, a *prepare_install* action, denoting the fact that an install message is put on the channel, a *trans_install* action, denoting the reception of an install message at the receiver, a *prepare_ack* action, denoting the fact that an acknowledgement is put on the channel, and finally a *trans_ack* action, denoting the reception of an acknowledgement at the sender. After the initial synchronization phase, actions concerning transmission of the packets are interleaved with actions concerning the periodic synchronization (see [Aldini et al. 1999b] for a full description of the EMPA specification).

	$Spec\#1$	$Spec\#1_s$	$Spec\#2$	$Spec\#2_s$	$Spec\#3$	$Spec\#3_s$
States	146	191	192	252	4005	4692
Transitions	281	401	423	583	7727	9020
Trace based traffic	345	379	546	552	1030	1002
Gaussian traffic	176	191	293	292	409	401
Exponential traffic	177	194	290	296	418	418

Table 3. State space size and simulation time (sec) of the algebraic models.

5. FUNCTIONAL AND PERFORMANCE ANALYSIS

5.1 Functional Analysis

Before presenting the outcome of the comparison of the QoS of the three mechanisms, we would like to point out that developing formal descriptions for the three audio mechanisms has allowed us to formally verify properties concerned with the correctness of the mechanisms themselves.

As a simple example, we have proved that the mechanisms #1, #2, and #3 are deadlock free. To achieve this, we have generated with TwoTowers the state space of the EMPA specifications of the three mechanisms, whose size is reported in the upper part of Table 3, where $Spec\#i$ and $Spec\#i_s$ are the EMPA specifications for mechanism $\#i$ without or with the spike detection mechanism, respectively. Note that the state spaces are very compact because the semantic models are symbolic, as we have seen in Sect. 3.2. Since TwoTowers has shown that neither of the three models has absorbing states, i.e. states without outgoing transitions, the three mechanisms are deadlock free.

As another example, we have verified via model checking an aspect of the behavior of the three way handshaking of mechanisms #3. More precisely, we have proved that no new synchronization is started before an installation packet is acknowledged. This property has been formalized through the following CTL [Clarke et al. 1986] formula:

$$\mathrm{AG}([\mathit{prepare\_install}]\,\mathrm{ff} \vee \\ \langle \mathit{prepare\_install}\rangle \mathrm{A}(([\mathit{prepare\_probe}]\,\mathrm{ff} \wedge [\mathit{prepare\_install}]\,\mathrm{ff})\mathrm{W}(\langle \mathit{trans\_ack}\rangle\,\mathrm{tt})))$$

which means that for any state (operator G) of any computation (operator A) starting at the initial state, action *prepare_install* cannot be executed (formula $[\mathit{prepare\_install}]\,\mathrm{ff}$) or it can be executed ($\langle \mathit{prepare\_install}\rangle$) and for any computation (operator A) of the derivative state the CTL formula $(([\mathit{prepare\_probe}]\,\mathrm{ff} \wedge [\mathit{prepare\_install}]\,\mathrm{ff})\ \mathrm{W}(\langle \mathit{trans\_ack}\rangle\,\mathrm{tt}))$ is satisfied. This formula means that neither action *prepare_probe* nor action *prepare_install* is executed until (operator W) action *trans_ack* can be executed.

5.2 How to Conduct the Performance Analysis

As far as the QoS comparison is concerned, because of the presence of generally distributed delays typically experienced by audio samples over the Internet, the analysis has been conducted via simulation. We have evaluated the three mechanisms using both experimentally obtained delay measurements of three 5 min long audio conversations between two different Internet sites (trace driven simulation) and two 1 min long audio conversations randomly generated according to Gaussian distributed delays and exponentially distributed delays, respectively. The two randomly generated traffics are intended as limiting scenarios while the traces represent intermediate, more realistic scenarios. The considered metrics are:

—the user perception of audio (which is represented by the packet loss rate vs. the average packet audio playout delay, where the packet loss rate is obtained by dividing the total number of discarded packets by the total number of transmitted packets);

—the packet loss rate vs. the receiving buffer capacity;

—the lateness of discarded packets vs. the average packet audio playout delay (where the lateness is computed as the difference between the value of the receiver clock at the arriving instant and the playout instant of the discarded packet);

—the average packet audio playout delay vs. the waiting time in the receiver buffer for the played packets (for mechanism #3).

To resort to a simulative analysis of the EMPA specifications of the three mechanisms, as explained in Sect. 3.4, we have provided an auxiliary specification containing the termination condition and the number of simulation runs, the list of performance measures we are interested in (and how to compute them), and (in the case of trace driven simulation) the file containing the experimentally obtained traces (and how to read them).

As an example, in the case of randomly generated traffic, the termination condition in the auxiliary specification concerning mechanism #1, is as follows:

$$\mathit{elapse\_tick}\ 60000\ 10$$

The simulation run stops after action *elapse_tick*, representing the passage of time, has been executed 60000 times (one time unit is one ms); the experiment consists of 10 simulation runs.

As far as the trace driven simulation is concerned, since the trace files are composed of 15000 transmission delays, which corresponds to a period of 5 min assuming a transmission rate of one packet every 20 ms, it has to be specified that the trace driven simulation terminates after 15000 executions of action *prepare_packet*:

$$\mathit{prepare\_packet}\ 15000\ 1$$

The duration of the experiments is reported in the lower part of Table 3. Experiments have been conducted on a personal computer with a 200 MHz Pentium II processor, 64 MB RAM, and Linux operating system.

5.3 Performance Results and Data Interpretation

In this section we present the results we have derived for the performance measures of interest by using both experimentally-obtained and randomly-generated traffic traces. It is worth mentioning that all the performance figures obtained under randomly generated traffic patterns are shown with 90% confidence intervals.

It is well known that two are the most important metrics that influence the user's perception of audio data: 1) the percentage of audio packets that arrive at the destination too late to be played out (and can be so considered lost), and 2) the amount of total delay that audio packets have to experience before they are played out at the destination. Hence, in order to evaluate effectively the performance of the three considered audio mechanisms, in Fig. 3 and 4 we report the average playout delay vs. the loss rate for Gaussian and exponential traffic, respectively, where Gaussian delays have expected value 100 ms and standard deviation 7 ms and exponential delays have expected value 100 ms (and standard deviation 10 ms). The difference between Fig. 3 and 4 is concerned with the fact that the performance results have been obtained with the spike detection mechanism deactivated and activated, respectively. Fig. 5, instead, shows the same metrics averaged over a set of traffic traces experimentally gathered between two interconnected sites, situated, respectively, at the Laboratory of Computer Science of Cesena (a remote site of the University of Bologna) and the CERN Institute of Geneva.

BoAT, an Internet audio conferencing tool, was used to collect those experimental traces. As detailed in [Roccetti et al. 1999b; Roccetti et al. 1999a], BoAT transmits approx. 100 bytes of audio data every 30 ms. The traces were obtained on a Sun Sparcstation platform, using 8-KHz sampled speech with bit rate of 8Kbit/sec (ITU-T G.729) as coding scheme, and unicast UDP as transport protocol. All the traces were collected by repeatedly playing an audio file at the sending host, with the receiving host listening to the received packets. However, 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 sending process was the same as the receiving process and were located in the same workstation situated at Laboratory of Computer Science of Cesena. Instead, the workstation situated at the CERN 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 the end-to-end delays coincide with round trip delays. As a final consideration, it is interesting to note that the above mentioned IP-based interconnection had 16 hops and was typically quite lossy.

With regard to the adopted simulation parameters, it is worth noticing that in all the considered simulation experiments, mechanism #1 has been simulated with $k = 4$ and $\alpha = 0.998002$. Mechanism #2, instead, has been simulated by varying the percentile point q from 0.995 to 0.92, in order to reach the fixed values of loss percentage (approx. from 3% to 15%), and then to measure the corresponding average playout delay. To this aim, and considering the length of the audio conversations for the trace driven simulation (about 15000 packets), we have chosen for w a value equal to 1500. As far as the parameters *head* and *tail* (see lines 2 and 5 of Table 2) are concerned we have chosen the values 4 and 2, respectively.

In order to provide the reader with an understanding of the effect that various playout delays and loss rates (as well as buffer dimensions) have on the quality of the perceived audio, we have reported in all the above mentioned Fig. 3 - 5 an approximate and intuitive representation of three different ranges of the quality of the perceived audio adopted from [Kostas et al. 1998]: toll quality for delays of less than 200-250 ms and low loss rates, potentially useful for delays of about 300-350 ms and higher loss rates, and poor for delays larger than 350 ms and very high loss rates. Furthermore, it is important to mention that all the Fig. 3 - 5 have been plotted with the x-axis (representing the loss percentage) starting with the 3% value. This decision has been taken based on the consideration that, during the experimental collection of the audio traces with the BoAT audio tool, a loss rate lower than 3% was never experienced.

Our figures illustrate several interesting points. First, we can observe that the mechanism #2 outperforms the others in all cases except for Fig. 5(*b*). This confirms the results of the analytical study reported in [Moon et al. 1998] where it was demonstrated that mechanism #2 performs close to a theoretical optimum. Second, by a comparison of the behavior of the audio mechanisms under different traffic patterns we can notice that mechanisms #1 and #2 perform better under Gaussian traffic than under exponential traffic. This result was expected since the basic design ideas behind both those mechanisms rely upon the assumption that audio packets follow a Gaussian distribution. Mechanism #3, instead, performs better under exponential traffic rather than under Gaussian traffic. The reason for this may be understood based on the consideration that mechanism #3 has been designed to be particularly reactive to sudden and large variations of the network delay (as it is the case with exponential traffic).

In addition, from a comparative analysis of the upper graphs of Fig. 3 and 4, it comes to evidence that under Gaussian traffic none of the three audio mechanisms benefits by the activation of the spike detection mechanism. This was an expected result since it is known that Gaussian traffic has no delay spikes. As far as the exponential traffic is concerned, instead, the following considerations may be derived from a comparison of the lower graphs reported in Fig. 3 and 4. With the delay spike mechanism activated, mechanism #3 benefits by an appreciable reduction of the average playout delay in the range of the loss rates of interest (i.e., less than 10% of losses). This reduction may be quantified as ranging in the interval of $[20, 30]$ ms. The activation of the spike detection/adaptation algorithm produces, instead, a reduction of the playout delay for both mechanisms #1 and #2 quantified as ranging in the interval of $[5, 15]$ ms. One such improvement may be considered almost negligible since a difference in the average playout delay less than 5/10 ms is not that significant. The minimum reduction under exponential traffic of the playout delay obtained by mechanisms #1 and #2 with the spike mechanism activated may be probably justified as follows. As indicated previously, the two randomly generated traffic patterns (Gaussian and exponential) are to be intended as limiting traffic scenarios. On the one hand, under Gaussian traffic, transmission delays which are significantly larger than the average delay value (and are consequently recognized as spikes) represent quite rare events. In this situation the spike algorithm of all the three audio mechanisms comes into play very seldom with no appreciable reduction in the playout delay. On the other hand, under exponential traffic large values for the transmission delays may be frequently generated. The high frequency of those large transmission delays impacts directly on the estimation of the average value of the network delay, as well as on the calculation of the average network delay variation. This fact has a strong influence on the detection of occurring spikes since the spike algorithms of mechanisms #1 and #2 are designed to capture spikes' occurrence based on the estimation of the average network delay and the average network delay variation, respectively. The surprising consequence of this fact is that a considerable amount of large transmission delays are not recognized as spikes, as they occur too much frequently within the exponentially-generated traffic pattern. As a consequence of this situation, the spike recognition/adaptation algorithm of mecha-

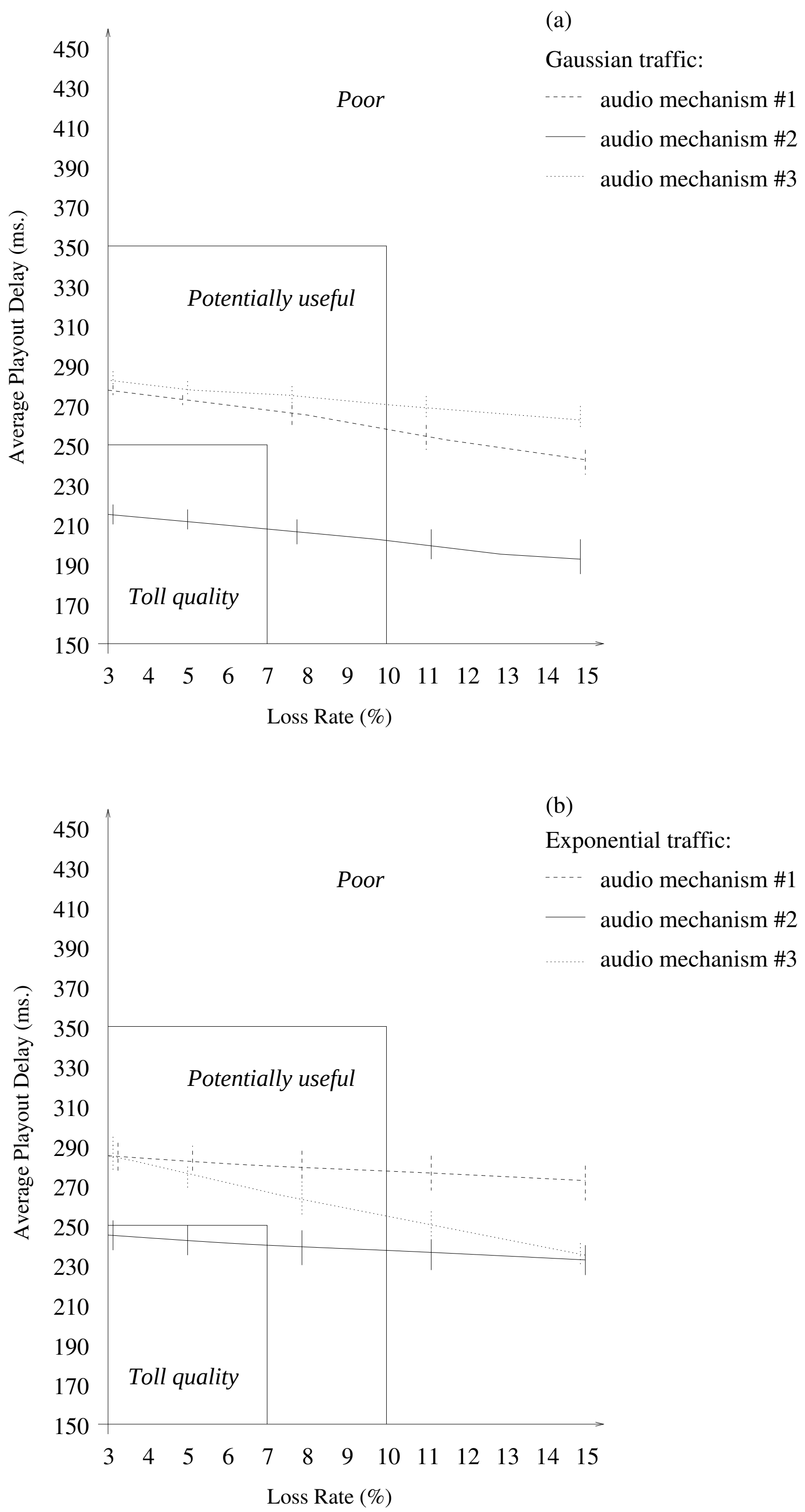
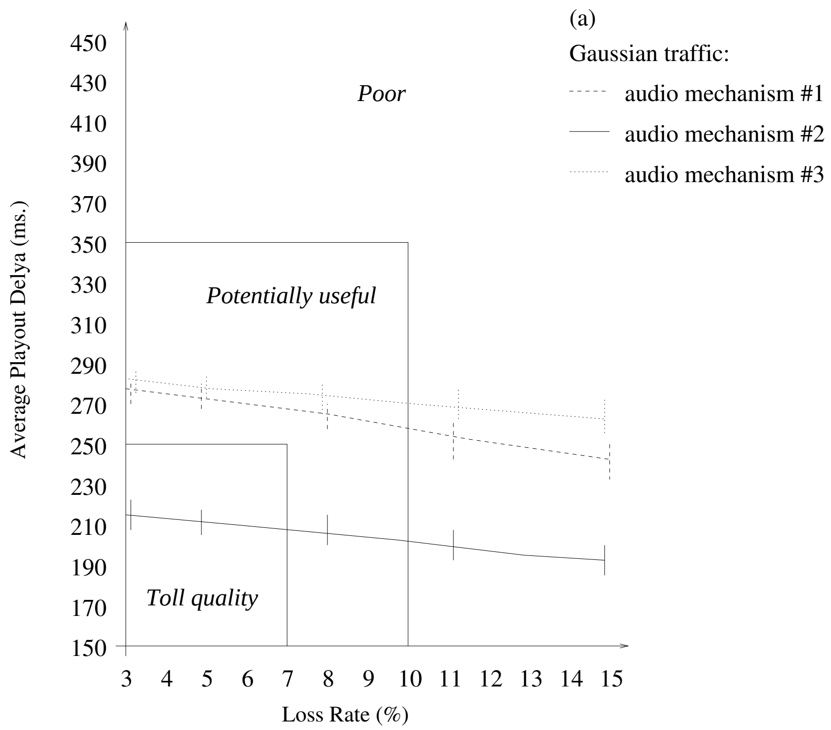


Fig. 3. Average playout delay (ms.) vs. loss rate (%): randomly generated traffic with playout delay spike mechanism deactivated. Gaussian traffic (a) and Exponential traffic (b).



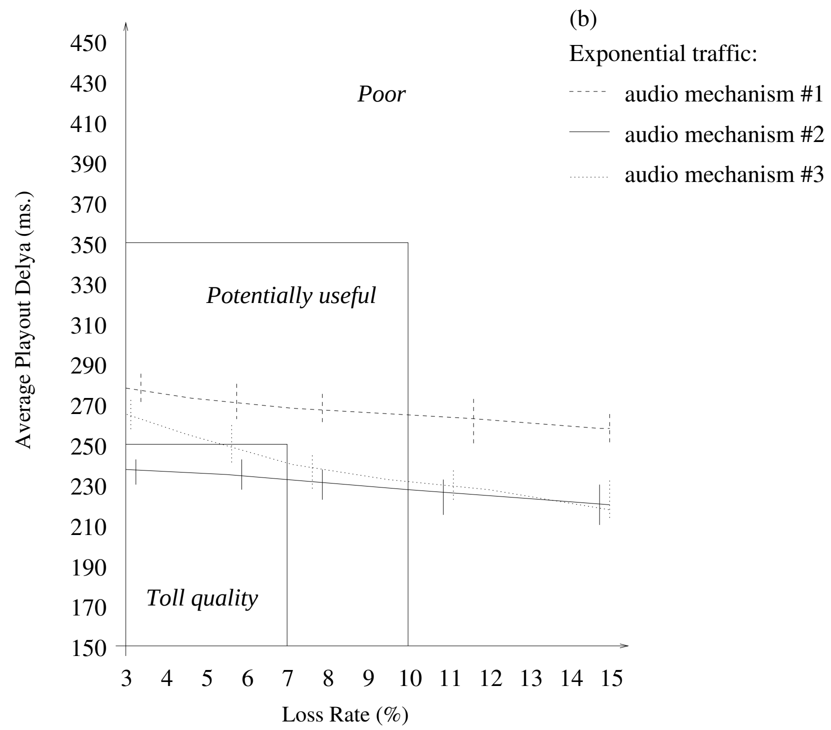


Fig. 4. Average playout delay (ms.) vs. loss rate (%): randomly generated traffic with playout delay spike mechanism activated. Gaussian traffic (a) and Exponential traffic (b).

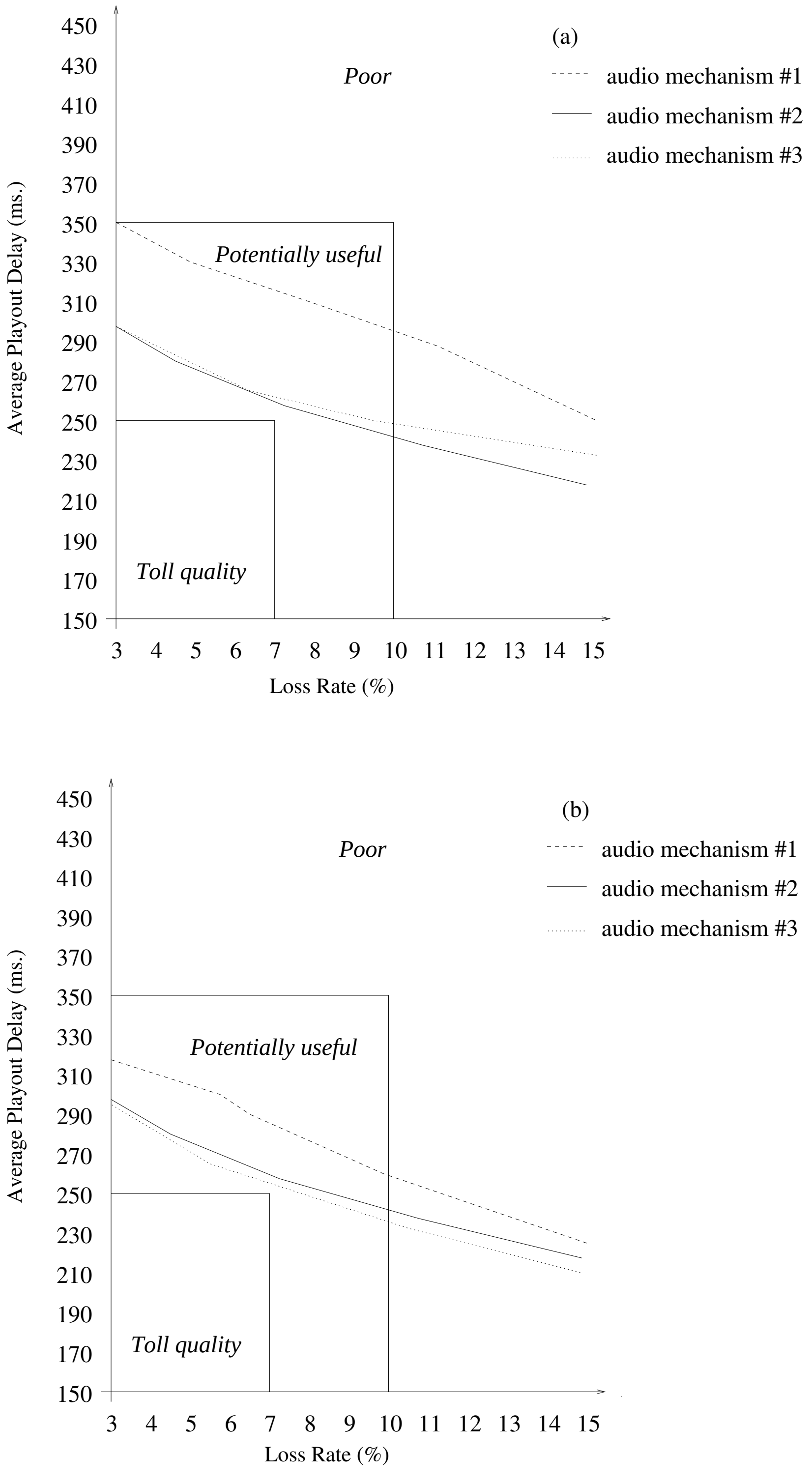


Fig. 5. Average playout delay (ms.) vs. loss rate (%): trace driven simulation with playout delay spike mechanism deactivated (a) or activated (b).

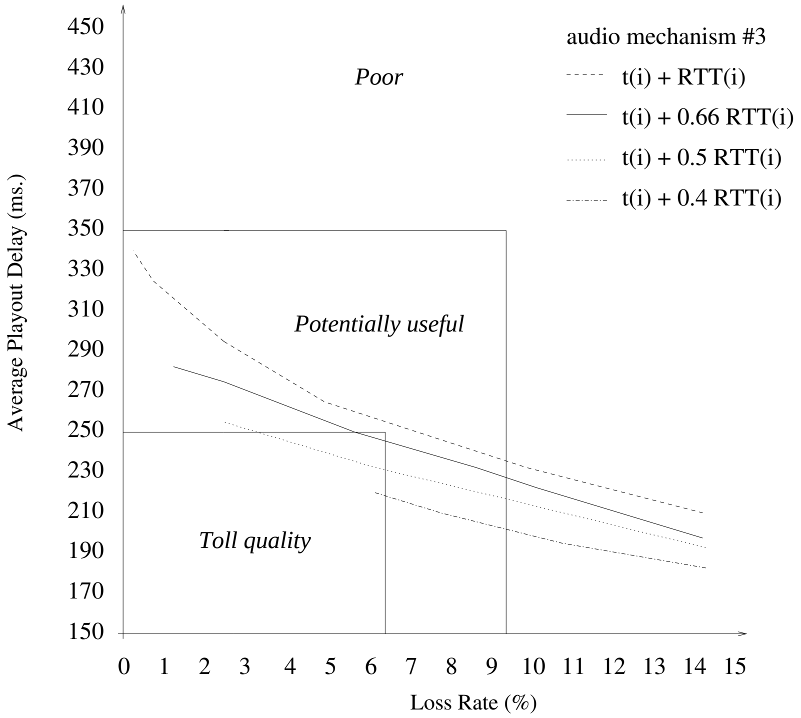


Fig. 6. Average playout delay (ms.) vs. loss rate (%) for different versions of Mechanism #3: trace driven simulation with playout delay spike mechanism activated.

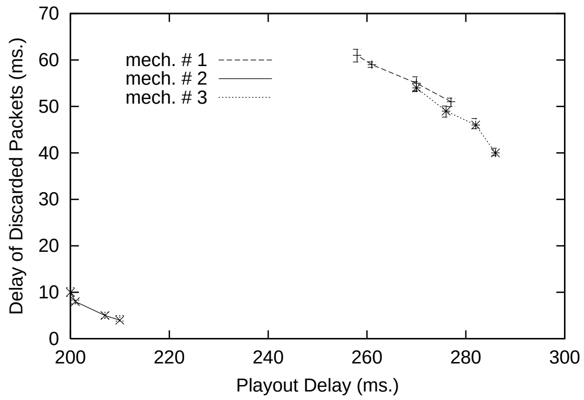


Fig. 7. Lateness of discarded packets (ms.) vs. average playout delay (ms.): Gaussian traffic with playout delay spike mechanism activated.

nisms #1 and #2 is activated rarely and no appreciable reduction of the parameters of interests may be experienced. Contrariwise, the spike detection/adaptation algorithm embodied in mechanism #3 performs better under exponential traffic due to the following motivation. Such a spike related algorithm has been designed to capture the occurrence of network delay spikes by simply comparing the values of subsequent RTT measurements. Hence, when a RTT value is measured which is considerably larger than the RTT value measured previously this event is recognized as the occurrence of a delay spike, independently of the frequency of the large transmission delays generated in the traffic pattern.

Summarizing, all the considerations above provide further evidence that probably none of the commonly used traffic models (i.e. Gaussian and exponential) is able to capture the characteristic aspects of the interaction between network delay and packet audio playout in cases where the jitter is very high. With this in view, more significant are the results shown in Fig. 5 since they stem from experimentally-obtained delay measurements of audio traffic during field trials. In particular, those results show that mechanism #3 is as much effective as mechanism #2, and that the activation of the spike detection mechanism is beneficial for all the three mechanisms.

As far as mechanism #3 is concerned, we have conducted an additional experiment, shown in Fig. 6, to measure the influence of the RTT parameter on the QoS; the experiment concerns the above mentioned traces experimentally gathered. On the basis of this experiment, we can conclude that mechanism #3 has the potential to outperform the others if a reduction by a factor of up to 50% of the RTT value is carried out. However, with respect to this point the following additional consideration is in order. Fig. 6 illustrates the different curves that have been obtained by performing different reductions of the RTT parameter. As a comparative analysis of the plotted curves shows, the larger the reduction of the RTT value the smaller the obtained average audio playout delay. However, as seen from the figure, this desirable result may not be obtained for any given packet loss rate. Simply put, if our intent is to keep the desirable packet loss range restricted to $2 - 5\%$, our analysis shows that this would not be possible with a reduction of the RTT value by a factor of up to 60%. For example, if we wish to keep the packet loss percentage as close to 0% as possible, no reduction of the RTT value is possible, independently of how large the playout delay is chosen. Based on the consideration above, the mechanism #3 has been designed for use in a real system equipped with the possibility of having the RTT parameter reduced by a factor of up to 50% [Roccetti et al. 1999a; Roccetti et al. 1999b]. The decision on how much to reduce the RTT parameter is left to the discretion of the system user. If he/she wishes to obtain very low packet loss rates, no reduction of the RTT parameter is recommended. On the contrary, if the user prefers lower playout delays at the cost of larger loss percentages an adequate reduction of the RTT parameter is suggested.

With the term lateness we refer to the difference between the value of the receiver clock upon the arrival of the discarded packet and its timestamp. Fig. 7 to 9 aim at measuring the lateness of discarded packets vs. the average playout delay. Therefore, those figures show for each considered audio mechanism the average increment of the playout delay we must pay in order to reduce lateness and, consequently, avoid discarded audio packets. In simple words, the plotted curves quantify the cost of a reduction of the loss percentage in terms of an increase of the playout delay. For the sake of clarity it is relevant to note that all the graphs representing the lateness value of discarded packets are plotted for average playout delay ranges corresponding to those reported in Fig. 4(a), 4(b), and Fig. 5(b), respectively. This entails that the ranges of the average delay values reported in the x-axis of all the above mentioned Fig. 7 to 9 are plotted restricted from the minimum value of 200 ms to the maximum value of 300/330 ms. With respect to the data reported in those figures, the following comments are in order. First, we may note that under Gaussian traffic (Fig. 7) all the plotted curves are characterized by a short dimension and a relatively flat slope. This kind of curves accounts for a situation where a small increase of the playout delay drastically reduces the packet loss percentage (from 15% to 3%) even if it does not determine a variation of the lateness of discarded packets by a significant margin. Yet again, as under Gaussian traffic the transmission delays of audio packets are not affected by the jitter phenomenon, it was an expected result to obtain one such tolerable cost for reducing the packet loss indipendently of the considered audio mechanism.

Under all the other traffic scenarios (i.e. exponential and trace driven), instead, the plotted curves become quite long. This entails that in order for a small increment of the playout delay to determine a significant reduction of the packet loss rate, the curve has to be very steep. More precisely, the steeper the lateness curve, the more a small increase of the playout delay may cause a reduction of both the lateness parameter and the loss percentage. On the contrary, a flat shape of a quite long curve means that a large cost must be paid (in terms of increased playout delay) in order to obtain an appreciable reduction of the packet loss rate. Based on these considerations we may deduce that, under exponential traffic, the cost to be paid in order to reduce the loss percentage may be affordable only for mechanisms #1 and #2. Contrariwise, under trace driven traffic, this cost is no longer affordable for any considered audio mechanism.

As noted previously, one of the primary performance metrics is the percentage of packets lost experienced at the destination. Such loss may be due to either packets that arrive too late for presentation or packets that arrive too far in advance of their playout time. In the latter case, packet loss results from the limitation on the finite size of the buffer (i.e. buffer overflow due to packets' premature arrivals). With this in view, Fig. 10 to 13 quantify for each mechanism the tradeoff between the loss rate and the maximum buffer capacity w.r.t. the three different traffic scenarios (Gaussian, exponential, trace driven). The figure obtained with field trials shows that, in order for the audio mechanisms to have the expected performance, the dimensioning of the receiving buffer must be dynamically adjusted according to the playout delay value computed by the mechanism. Instead, with Gaussian and exponential traffic, a threshold value seems to exist beyond which the increase of the buffer capacity provides no benefit. Moreover, it is worth mentioning that, while under the randomly generated traffic patterns mechanisms #1 and #2 perform a little better than mechanism #3 the plotted curves indicate that mechanism #3 performs much better than the other mechanisms under the trace driven traffic. In particular, for any given loss rate (e.g. 5%), mechanism #3 uses as much as the half of the buffer size requested by mechanisms #1 and #2. Yet again, mechanism #3 shows a significant performance advantage in cases where a large delay jitter

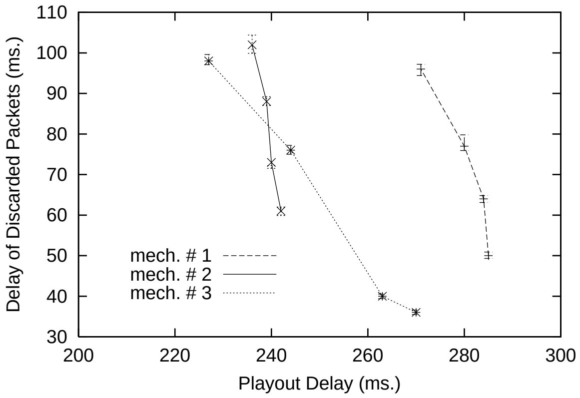


Fig. 8. Lateness of discarded packets (ms.) vs. average playout delay (ms.): Exponential traffic with playout delay spike mechanism activated.

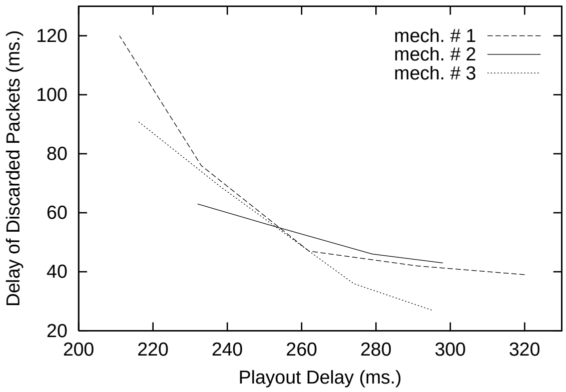


Fig. 9. Lateness of discarded packets (ms.) vs. average playout delay (ms.): trace driven traffic with playout delay spike mechanism activated.

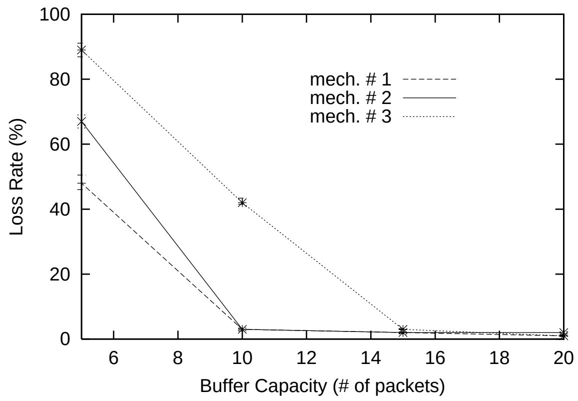


Fig. 10. Loss rate (%) vs. buffer capacity (packets): Gaussian traffic with playout delay spike mechanism activated.

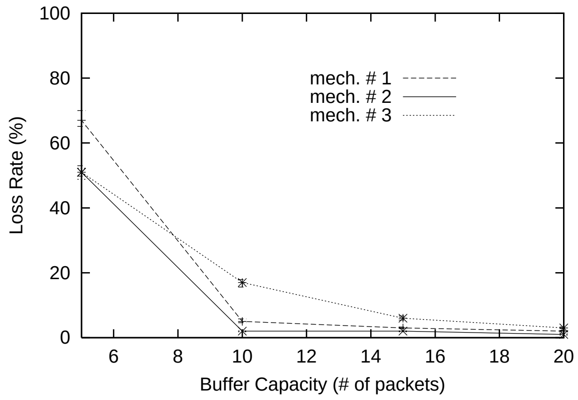


Fig. 11. Loss rate (%) vs. buffer capacity (packets): Exponential traffic with playout delay spike mechanism activated.

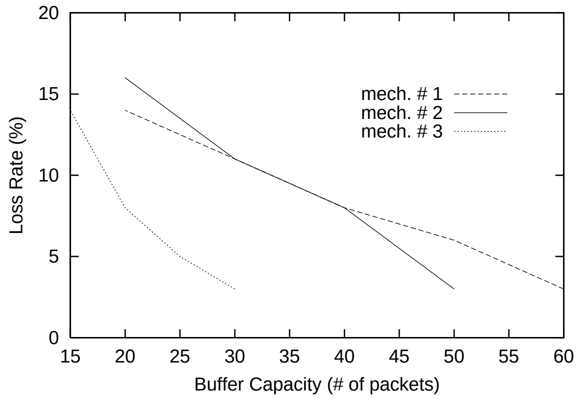


Fig. 12. Loss rate (%) vs. buffer capacity (packets): trace driven traffic with playout delay spike mechanism activated.

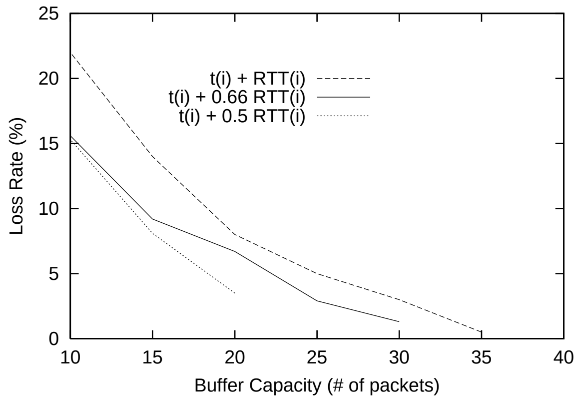


Fig. 13. Loss rate (%) vs. buffer capacity (packets) for different versions of Mechanism #3: trace driven traffic with playout delay spike mechanism activated

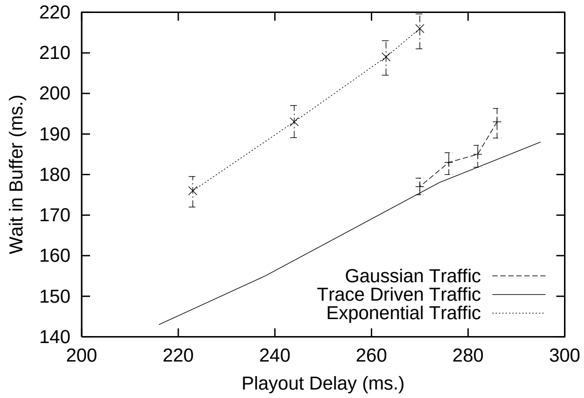


Fig. 14. Wait in buffer (ms.) vs. average playout delay (ms.) for Mechanism #3 with playout delay spike mechanism activated.

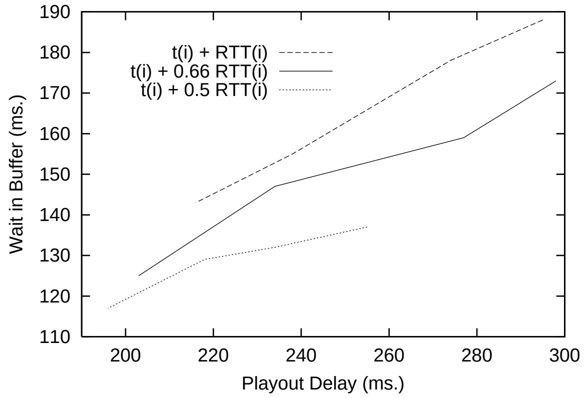


Fig. 15. Wait in buffer (ms.) vs. average playout delay (ms.) for different versions of Mechanism #3: trace driven traffic with playout delay spike mechanism activated.

is experienced. For the sake of completeness, we have also reported in Fig. 13 the same metrics for mechanism #3 in the case of trace driven traffic for different values of the *RTT* parameter. The data plotted in this figure confirm the indication that if a certain reduction of the *RTT* parameter is performed, then packet loss rates below a given threshold cannot be achieved.

Finally, the curves in Fig. 14 and 15 show for mechanism #3 the total amount of time that nondiscarded packets wait in the receiver buffer before their playout time, for the different values of the average playout delay shown in Fig. 4(a), 4(b), 5(b), and 6. From an analysis of those curves it is important to mention the following consideration. It is evident that the shorter the measured range of the variation of the playout delay, the shorter the curves plotted in the figures. Consequently, the larger the average playout delay, the larger the wait time in the buffer. In particular, Fig. 15 shows how a reduction of the *RTT* parameter may determine the reduction of both the playout delay and the waiting time in the receiver's buffer by a significant margin.

To conclude this section, we can summarize the obtained results by observing that the three audio mechanisms, although based on quite different design principles, are all effective since they basically provide comparable QoS. As expected, the audio mechanisms #1 and #2 outperform the audio mechanism #3 in the case of Gaussian traffic, while the reverse is true in the realistic case of field trials. Furthermore, the sensitivity analysis conducted for mechanism #3 (see Fig. 6, 13, and 15) has revealed that the QoS provided by the mechanism may greatly benefit by the reduction of *RTT* value.

6. CONCLUSION

In this paper we have formally modeled and analyzed, by means of the stochastic process algebra EMPA and the related tool TwoTowers, three soft real time applications concerning audio transmission over the Internet. This case study has shown the adequacy of the stochastic process algebra approach to model complex systems and to analyze them not only from the performance but also from the functional point of view. As an example, it would not have been possible to verify the deadlock freedom of the three mechanisms nor the correct behavior of the three way handshaking of mechanisms #3 with a conventional simulation program.

We would like to point out that this paper is, in some sense, the natural continuation of [Bernardo et al. 1999], where we reported on our experience of using EMPA/TwoTowers during the design of the synchronization phase for the audio mechanism of [Roccetti et al. 1999b] in order to predict the performance of different implementation schemes of the phase. In this paper, instead, we have analyzed a complete EMPA specification of this mechanism, while in [Bernardo et al. 1999] only a simplified description of the same mechanism (developed during the design of the mechanism itself) is shown, as well as an analysis of complete EMPA specifications of the other two audio mechanisms in order to compare their QoS.

From the performance standpoint, we have compared the three mechanisms under various traffic conditions (both experimentally obtained on the field and randomly generated), w.r.t. different metrics. Using TwoTowers has been very helpful to detect that neither of the three mechanisms outperforms the other two in general, and considering several metrics has permitted to analyze advantages and weaknesses of the three mechanisms. Furthermore, the experiments have revealed the relationship between the particular mechanism and the traffic conditions in order to establish where we can profitably use one mechanism rather than another one. The sensitivity analysis conducted for the three mechanisms (see Fig. 7 and the following ones) has revealed that it is not possible to improve the offered QoS by simply playing on the value of the parameters, and probably the presented results are close to the maximum performance these mechanisms can guarantee.

In the case of mechanism #3, however, the sensitivity analysis concerning the RTT value has turned out to be quite interesting and its usefulness will be explored in a future research. Indeed, we observe that the actual objective of formal description techniques is not that of being used to assess system properties a posteriori. More specifically, they should be profitably used when designing new protocols to predict their performance and verify their functional properties before they are implemented, like in [Bernardo et al. 1999].

ACKNOWLEDGMENTS

We would like to thank the anonymous referees for their helpful comments. This research has been funded by Progetto MURST Cofinanziato TOSCA and SALADIN and by a grant from Microsoft Research Europe.

REFERENCES

Aldini, A., Bernardo, M., and Gorrieri, R. 1999a. An algebraic model for evaluating the performance of an atm switch with explicit rate marking. In *Proc. of the 7th Int. Workshop on Process Algebras and Performance Modeling (PAPM '99), Zaragoza (Spain)* (1999), pp. 119–138. Prensas Universitarias de Zaragoza.

Aldini, A., Bernardo, M., Gorrieri, R., and Roccetti, M. 1999b. Comparing the qos of internet audio mechanisms via formal methods. Tech. Rep. UBLCS 99-04, University of Bologna (Italy).

Bernardo, M. 1999. *Theory and Application of Extended Markovian Process Algebra*. Ph. D. thesis, University of Bologna (Italy). http://www.di.unito.it/~bernardo.

Bernardo, M., Cleaveland, W., Sims, S., and Stewart, W. 1998a. Twotowers: A tool integrating functional and performance analysis of concurrent systems. In *Proc. of the IFIP Joint Int. Conf. on Formal Description Techniques for Distributed Systems and Communication Protocols and Protocol Specification, Testing and Verification (FORTE/PSTV '98), Paris (France)* (1998), pp. 457–467. Kluwer.

Bernardo, M., Donatiello, L., and Gorrieri, R. 1998b. A formal approach to the integration of performance aspects in the modeling and analysis of concurrent systems. *Information and Computation 144*, 83–154.

Bernardo, M., Gorrieri, R., and Roccetti, M. 1999. Formal performance modelling and evaluation of an adaptive mechanism for packetised audio over the internet. *Formal Aspects of Computing 10*, 313–337.

Clarke, E., Emerson, E., and Sistla, A. 1986. Automatic verification of finite state concurrent systems using temporal logic specifications. *ACM Trans. on Programming Languages and Systems 8*, 244–263.

Cleaveland, W. and Sims, S. 1996. The ncsu concurrency workbench. In *Proc. of the 8th Int. Conf. on Computer Aided Verification (CAV '96), New Brunswick (NJ)* (1996), pp. 394–397. Lecture Notes in Computer Science, vol. 1102.

Hardman, V., Sasse, M., and Kouvelas, I. 1998. Successful multi-party audio communication over the internet. *Comm. of the ACM 41*, 74–80.

ITU. 1996. *ITU-T Recommendation G.729 - G.723.1*. ITU.

KOSTAS, T., BORELLA, M., SIDHU, I., SCHUSTER, G., GRABIEC, J., AND MAHLER, J. 1998. Real-time voice over packet-switched networks. *IEEE Network 12*, 18–27.

LELAND, W., TAQQU, M., WILLINGER, W., AND WILSON, D. 1994. On the self-similar nature of ethernet traffic. *IEEE/ACM Transactions on Networking 2*, 1–15.

MAUTH, R. 1998. Adjust to the jitter. *BYTE*, 3–5.

MOON, S., KUROSE, J., AND TOWSLEY, D. 1998. Packet audio playout delay adjustment: Performance bounds and algorithms. *ACM Multimedia Systems 6*, 17–28.

RAMJEE, R., KUROSE, J., TOWSLEY, D., AND SCHULZRINNE, H. 1994. Adaptive playout mechanisms for packetized audio applications in wide-area networks. In *Proc. of INFOCOM '94, Montreal (Canada)* (1994).

ROCCETTI, M., GHINI, V., BALZI, D., AND QUIETI, M. 1999a. *BoAT: The Bologna Optimal Audio Tool*. University of Bologna (Italy). `http://radiolab.csr.unibo.it/BoAT/src`.

ROCCETTI, M., GHINI, V., PAU, G., SALOMONI, P., AND BONFIGLI, M. 1999b. Design and experimental evaluation of an adaptive playout delay control mechanism for packetized audio for use over the internet. *Multimedia Tools and Applications*. To appear.

SCHULZRINNE, H. 1992. Voice communication across the internet: A network voice terminal. Tech. rep., University of Massachusetts, Amherst (MA).

SHEDLER, G. 1983. Generation methods for discrete event simulation. In S. LAVENBERG Ed., *Computer Performance Modeling Handbook*, pp. 223–266. Academic Press.

STEWART, W. 1994. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press.

WELCH, P. 1983. The statistical analysis of simulation results. In S. LAVENBERG Ed., *Computer Performance Modeling Handbook*, pp. 267–329. Academic Press.