Formal Aspects of Computing (2006) 3: 1–25

Formal Performance Modelling and Evaluation of an Adaptive Mechanism for Packetised Audio over the Internet

Marco Bernardo, Roberto Gorrieri, Marco Roccetti

Università di Bologna, Dipartimento di Scienze dell'Informazione
Mura Anteo Zamboni 7, 40127 Bologna, Italy
E-mail: {bernardo, gorrieri, roccetti}@cs.unibo.it

Keywords: Formal support to system design; Packetized audio; Process algebras; Performance evaluation; Simulation

Abstract. A case study is presented which concerns the design of an adaptive mechanism for packetized audio for use over the Internet. During the design process, the audio mechanism was modeled with the stochastically timed process algebra EMPA and analyzed via simulation by the EMPA based software tool TwoTowers in order to predict the percentage of packets that are received in time for being played out. The predicted performance figures obtained from the algebraic model illustrated in advance the adequacy of the approach adopted in the design of the audio playout delay control mechanism. Based on these performance figures, it was possible to implement and develop the complete mechanism without incurring in additional costs due to the late discovery of unexpected errors or inefficiency. Performance results obtained from experiments conducted on the field confirmed the predictive simulative results.

1. Introduction

The desirability of taking into account the performance aspects of a system in the early stages of its design has been widely recognized [YK82, Fer86, Har86, BV88]. Nevertheless, it often happens that a system is tested for efficiency only after it has been fully designed and tested for functionality. As a consequence, the detection of poor performance causes the system to be designed again, so the cost of the project increases. In order to cope with such a problem, several formal

Correspondence and offprint requests to: Marco Bernardo.

description techniques have been and are being extended by adding the capability of modeling time. An emerging field in the area of the integration of formal methods and performance evaluation is the field of stochastically timed process algebras (see e.g. [GHR93, Hil96, BDG98]), algebraic languages endowed with a small set of powerful operators as well as a family of actions which permit to express not only the type but also the duration of the activities executed by the systems being modeled. A major advantage of stochastically timed process algebras is that they provide a linguistic framework for the compositional modeling and analysis of systems.

In this paper we shall report on the use of the stochastically timed process algebra EMPA [BG98] to formally describe and analyze the adaptive mechanism for packetized audio over the Internet proposed in [RGSPB98]. The reason why such a mechanism has been chosen for our case study is that in the two decades that have passed since the early experiments with packetized voice (e.g. Arpanet [Coh77]) packetized audio applications have turned out into tools that many Internet users now try to use with regularity. For example, the audio conversations of many international conferences and workshops are now usually transmitted over the Mbone (the multicast backbone), an experimental overlay network of the Internet [MB94], and also smaller audio group meetings are frequently conducted over the Internet using quite well known multimedia tools. The purpose of the case study, besides showing the adequacy of EMPA to model real world algorithms, is to predict the performance of the audio mechanism. To accomplish this, the EMPA model of the audio mechanism is studied by the EMPA based software tool TwoTowers [BCSS98] in order to assess the percentage of packets that are received in time for being played out.

The general motivation behind the use of EMPA for this case study is its recognized expressive power [BG98]. An additional reason for adopting EMPA is that the execution of the audio mechanism strongly depends on the use of clock values and packet timestamps. Needless to say, such features can be adequately modeled only if the adopted algebraic formalism supports value passing. Hence the use of EMPA, since it is equipped with the capability of exchanging values among processes [Ber98]. Finally, the possibility of modeling general distributions, in particular Gaussian distributions, is a necessary requirement in order to describe the audio mechanism of interest. Such a requirement is met by the combined use of EMPA immediate actions with EMPA value passing based expressions. It is worth noting that, since general distributions come into play, a Markovian analysis is no longer possible. Hence the need of resorting to a simulative analysis of performance, which is supported by the EMPA based software tool TwoTowers.

The remainder of the paper is structured as follows. In the next section a novel adaptive mechanism for controling the playout delay of packet audio applications [RGSPB98] is sketched. Then in Sect. 3 we recall EMPA and TwoTowers, i.e. the stochastically timed process algebra and the related software tool we have used to model and analyze the audio mechanism mentioned above. Finally, in Sect. 4 simulative performance figures are provided which have been derived from the experiments conducted on the algebraic models of the audio mechanism. Such figures are also compared with results obtained from experiments conducted on the field with a prototype implementation of the mechanism itself.

We conclude this introduction by emphasizing the main contribution of the paper. In contrast with other case studies where either toy examples or already existing mechanisms (such as, e.g., ATM switches and multiprocessor main-

frames) are discussed, the construction of the formal algebraic model of the audio mechanism has been part of the design process of the mechanism itself. The predicted performance figures obtained from the algebraic model have illustrated in advance the adequacy of the approach adopted in the design of the audio playout delay control mechanism. Based on these performance figures, it has been possible to implement and develop the complete mechanism without incurring in additional costs due to the late discovery of unexpected errors or inefficiency.

2. An Adaptive Mechanism for Packetized Audio

With the growth in communication bandwidth provided by high speed networks and the improvement in microprocessor technology and compression techniques, multimedia conferencing over the Internet is emerging as a low cost alternative to more traditional videoconferencing services offered by telecommunications companies through either satellite or ISDN based solutions. In essence, the use of the IP multicast technology has made packet switched networks able to support multiparty real time conferences between a large number of participants geographically dispersed all over the world [HSK98]. Internet based multimedia conferencing systems may benefit many users in several application areas, such as telemedicine, remote computer supported collaboration, electronic commerce, and distance learning [Roc99].

Even if the provision of video and shared textual data represents an important benefit offered by networked multimedia systems, it is experimentally demonstrated that the transmission of audio of satisfactory quality is a necessary condition for successfully supporting critical distributed multimedia applications. Hence, providing satisfactory audio quality must be considered fundamental for realizing adequate networked multimedia conferencing systems. In recent years, many multicast audio conferencing tools were designed and developed that provide an audio quality which is comparable to that of the telephone system, at a typical sampling rate of 8 kHz [BV96, HSK98, JM, Sch92]. Typically, in those packet audio applications, packets are buffered at the receiving site and their playout time is delayed in order to compensate for variable network delays.

In particular, packet audio tools, such as NeVot [Sch92], vat [JM], and the INRIA audio tool [BV96], operate by periodically sampling audio streams generated at the sending host, packetizing them, and transmitting the obtained packets to the receiving site using datagram based connections (e.g. UDP). A typical 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 packets are generated. In order for the receiving site to reconstruct the audio conversation, the audio packets constituting a talkspurt must be played out in the order they were emitted at the sending site. If the delay between the arrival of subsequent packets (known as *jitter*) is constant (i.e. the underlying transport network is jitter free), a receiving site may simply play out the arriving audio packets as soon as they are received. Unfortunately, this is only rarely the case, since jitter free, ordered, on-time packet delivery almost never occurs in today's packet switched networks. Those variations in the arrivals of subsequent packets strongly depend on the traffic conditions of the underlying network. Packet loss rates (due to the effective loss and damage of packets as well as late arrivals) often vary between 15% and 40% [MKT98]. In addition, extensive experiments

with wide area network testbeds have shown that the delays between consecutive packets may also be as much as 1.5 seconds, thus impairing real time interactive human conversations.

New protocol suites, such as Resource Reservation Protocol (RSVP) [ZH93] and Real Time Transport Protocol (RTP) [SCFJ95], might eventually ameliorate the effect of jitter and improve the quality of the audio service over the Internet, but they are not yet widely used.

On the other hand, the most used approach (adopted by all the already mentioned audio tools) is to adapt the applications to the jitter present on the network. In order to compensate for these variable delays, a smoothing playout buffer is used at the receiver. Received audio packets are first queued into the buffer, then the playout of a packet of a given talkspurt is delayed for some quantity of time beyond the reception of the first packet of that talkspurt. Needless to say, a crucial tradeoff exists between the length of the imposed playout delay and the amount of lost packets due to their late arrival: the longer the additional playout delay, the more likely it is that a packet will arrive before its scheduled playout deadline. However, too long playout delays may in turn seriously compromise the quality of the conversation over the network.

Typical acceptable values for the end-to-end delay between packet audio generation at the sending site and its playout time at the receiver are in the range of 300-400 msec, instead a percentage of no more than 5-10% of packet loss is considered quite tolerable in human conversations [BCV95].

In addition to adjusting the audio playout delay in order to compensate for the effect of the jitter, modern audio tools typically also make use of error and rate control mechanisms based on a technique known as forward error correction (FEC) to reconstruct many lost audio packets [BV98, PR97]. For example, the INRIA audio tool adjusts the audio packet send rate to the current network conditions, adds redundant information to packets (under the form of highly compressed versions of a number of previous packets) when the loss rate surpasses a certain threshold, and establishes a feedback channel to control the send rate and the redundant information. Simply put, the complete process is controled by an open feedback loop that selects among different available compression schemes and the amount of redundancy needed, as described in the following.

If the network load and the packet loss are high, the amount of compressed redundant information carried in each packet is increased by adding to each packet the compressed version of the previous two to four audio packets. In 5 second intervals the receiver returns quality of service reports to the sender, in order to regulate and adapt the quantity of redundant information being sent.

As discussed above, efficient playout adjustment mechanisms have been developed to minimize the effect of delay jitter. Typically, as already mentioned, a receiving site in an audio application buffers packets and delays their playout time. Such a playout delay may be kept constant for the duration of the audio session or adaptively adjusted from one talkspurt to the next. Due to the fluctuant end-to-end (application-to-application) delays experienced over the Internet, constant nonadaptive playout delays may result in unsatisfactory quality for audio applications.

Hence, there are two approaches widely exploited for adaptively adjusting playout time. The former uses the same playout delay throughout a given talkspurt, but permits different playout delays in different talkspurts. In the latter, instead, the playout delay varies from packet to packet. However, an adaptive adjustment on a per-packet basis may introduce gaps inside talkspurts and thus

is considered as of being damaging to the perceived audio quality. On the contrary, the variation of the playout delay may introduce artificially elongated or reduced silence periods, but this is considered acceptable in the perceived speech if those variations are reasonably limited. Hence, the totality of the above mentioned tools adopt a mechanism for adaptively adjusting the playout delays on a per-talkspurt basis. In order to design this playout control mechanism, all the above cited audio applications make use of the two following strong assumptions.

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

Based on the above assumptions, the playout control mechanism (which is in essence common to all the cited audio tools) works as described in the following. Let:

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

If i is the first packet of a given talkspurt, then the playout time p_i for that packet is calculated as

$$p_i = t_i + \hat{d}_i + 4 \cdot \hat{v}_i$$

Instead, any other packet j of a talkspurt (different from the first one i) is played out using the following playout time

$$p_j = p_i + t_j - t_i$$

where the estimation of both the average delay and the average delay variation are carried out using the well known stochastic gradient algorithm [RKTS94] by means of the following formulas

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

in which the constant a (usually equal to 7/8) is a weigth that characterizes the memory properties of the estimation. Extensive experiments have been carried out that have shown that the above playout delay control mechanism may be adequate to obtain acceptable values for the tradeoff between the average playout delay and the loss due to late packet arrivals. However, in some circumstances, the above mechanism may suffer from many problems, especially when deployed over wide area networks. In particular, the following problems have been pointed out [MKT98]:

- The external software based mechanisms (e.g. the Internet based NTP protocol) that are typically used to maintain synchronized the clocks of both the sender and the receiver may turn out to be too much inaccurate to cope with the real time nature of the audio generation/playout process.
- The widely adopted assumption that the packet transmission delays follow a Gaussian distribution seems to be a plausible conjecture only for those limited time intervals in which the overall load of the underlying network is quite light. Indeed, recent experimental studies carried out over the Internet have indicated the presence of frequent and conspicuously large end-to-end delay spikes for periodically generated packets (as is the case with audio packets) [BCV95].
- Even in the cases when the delays (due to the traffic conditions over the network) may be adequately modeled with a Gaussian distribution, the computational effort and complexity which are necessary for elaborating the equations provided above may be too large, thus negatively impacting on the real time playout process (especially when the receiving host is under heavy load).

Recently, a new on-line, adaptive mechanism for the control of the playout delay has been proposed that ameliorates all the negative effects of the audio tools described above, while maintaining satisfiable values of the tradeoff between the average playout delay and the packet loss due to late arrivals [RGSPB98]. This policy assumes neither the existence of an external mechanism for maintaining an accurate synchronization at both the sending and the receiving sites, nor a Gaussian distribution for the end-to-end transmission delays of the audio packets. In particular, it provides:

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

In the following, a succinct and simplified (even if exhaustive) description of the main ideas underlying the mechanism is provided. For additional technical details and a deeper discussion of its performance, see [RGSPB98].

For continuous playout of audio packets at the receiving site, it is essential that the audio packets be available at the receiver prior to their respective playout time and that the rate of consumption (i.e. playout) of packets at the receiver meets the rate of transmission at the sender [RKR96]. Hence, 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 actual 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 the following quantity

$$T_S - T_R = \Delta$$

where T_S and T_R are, respectively, the readings of the local clocks at the sender and at the receiver, and Δ (assuming identical drifts for both clocks) is a non negative 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. Later, we will show how the above mentioned upper bound Δ_{max} may be estimated and how this value impacts on the playout time instant of the audio packets.

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 (0), while subsequent packets arrive with maximum delay (Δ_{max}). In this case, due to the minimum delay of the first packet, there is a null difference between the clock at the sender and at the receiver

$$T_S - T_R = 0$$

However, consider now a situation in which a subsequent packet arrives at the receiver that suffers from maximum delay (Δ_{max}). Suppose that this packet has been transmitted by the sender when its clock shows a time value equal to, say, X. Due to the imposed synchronization, at that precise instant also the receiver's clock would show a value equal to X. Now, adding the transmission delay of Δ_{max}, the arrival time of this subsequent packet occurs when the receiver's clock shows the value given by $X + \Delta_{max}$. Unfortunately, that packet would be too late for playout and consequently discarded.

This example suggests that a practical and secure method for preventing buffer underflow (i.e. packets lost due to their late arrival) is that the receiver delays the setting of its local clock of an additional quantity equal to Δ_{max} when the first packet of the talkspurt is received. Precisely, when the first packet is received with its timestamp equal to C, the receiver sets its local clock to a value equal to $C - \Delta_{max}$. With this simple modification the problem of buffer underflow gets solved. Simply put, this policy implicitly guarantees that all the audio packets that will suffer from a transmission delay not greater than Δ_{max} will be on-time for the playout.

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

to the maximum delay experienced by the first packet of the talkspurt, the time difference between the two clocks at the sender and at the receiver at a given time instant is equal to

$$T_S - T_R = \Delta_{max} + \Delta_{max} = 2 \cdot \Delta_{max}$$

Now, if a subsequent packet arrives that has experienced a minimum delay (in comparison with the maximum delay suffered from the first packet of the talkspurt), the receiver's clock shows a time value equal to

$$T_R = C - 2 \cdot \Delta_{max}$$

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

However, from the discussion above, two problems have been left unresolved:

- How is it possible to estimate an upper bound on the maximum delay experienced in real transmissions over the Internet?
- How is it possible to adapt the playout control mechanism sketched above in order to compensate for the highly fluctuant end-to-end delays experienced over the Internet?

In the mechanism proposed in [RGSPB98], 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 with the value of the the timestamp attached to the probe packet 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, respectively at the sender and at the receiver, is equal to an unknown quantity (say t_0) which may range from a theoretical lower bound of 0 (that is all the RTT value has been consumed on the way back from the receiver to the sender) and a theoretical upper bound of RTT (that is all the RTT has been consumed on the way in during the transmission of the probe packet).

Unfortunately, as already mentioned, t_0 is unknown and a scarce approximation of this value (e.g. $t_0 = RTT/2$ [AY96]) might result in both playout buffer underflow problems and packet loss due to late arrivals thus impairing the audio conversation. Based on these considerations, in the mechanism proposed in [RGSPB98] the sender, after getting the response packet from the receiver and calculating 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 at 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 given 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 order for the proposed policy to adaptively adjust to the highly fluctuant end-to-end delays experienced over the Internet, the above mentioned synchronization technique is first carried out prior to the beginning of the first talkspurt of the audio conversation, then periodically repeated throughout the entire conversation (the adopted period is about 1 second). 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, as mentioned above.

This method guarantees that both the introduced additional playout time and the buffer dimension are always proportional to the traffic conditions. However, it may not be possible, in general, to replace on-the-fly at the receiver the current value of its own clock and the dimension of its playout buffer during a talkspurt. In fact, as sketched at the beginning of this section, such a per-packet adaptive adjustment of the synchronization parameters might introduce either gaps inside the talkspurt or even time collisions between the audio packets. Consequently, the installation of the new synchronization values at the receiver (i.e. the activity of changing the setting of the value of the receiver's playout clock and of the buffer dimension) is carried out only during the periods of audio inactivity, when no audio packets are generated by the sender (i.e. during silence periods between different talkspurts).

A formal theorem (with a corresponding algorithm) is provided in [RGSPB98] that shows that the installation of a new synchronization between the sender and the receiver may be conducted during a silence period (detected by the sender) without introducing either gaps or packet collisions inside the talkspurts of the audio conversation, only if the silence period is equal to or greater than a given value δ_{sync}. Denoted respectively with $\Delta_0 = t_0 + RTT_0$ and $\Delta_1 = t_1 + RTT_1$ the values of the differences between the sender's clock and the receiver's clock due to consecutive synchronizations, the value δ_{sync} may be calculated as given by

$$\delta_{sync} = |\Delta_0 - \Delta_1|$$

Note that even if the values of t_0 and t_1, respectively, are not exactly computable, instead their difference may be exactly evaluated at the receiver's site [Cri89]. For the sake of brevity, we do not report here the above mentioned synchronization installation algorithm that is fully detailed in [RGSPB98].

Another possible 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 gets solved in the mechanism proposed in [RGSPB98] using two different modes of operation, depending on whether a delay spike has been detected. In (say) the normal mode (i.e. no delay spike has been detected) the mechanism operates as already described, while in the spike detected mode (i.e. a high delay spike has been detected) the playout delay values obtained from the spike affected synchronization are smoothed out using an *ad hoc* smoothing mechanism [RGSPB98], and each arriving audio

packet is played out using those smoothed playout delay values. In other words, the spike is followed until the end of the spike itself is detected.

Upon detecting the end of a delay spike (e.g., with the next synchronization), the normal mode of operation is resumed using the values of the last executed synchronization. Unfortunately, there is no space here for fully describing the spike detection algorithm that is used. This algorithm is detailed in [RGSPB98] and is, in essence, based on the comparison between the values of the Δ_i (see above) obtained from two consecutive synchronization activities. Succinctly, if the difference between those two values is positive and greater than a given threshold value, a spike is considered to be detected [MKT98].

The adaptive mechanism proposed in [RGSPB98] will be formally modeled in Sect. 4 with EMPA in order to predict its performance. In the next section we therefore recall some concepts about EMPA: such a section can be skipped by readers being acquainted with EMPA.

3. EMPA and TwoTowers

In this section we recall from [BG98, Ber98] some concepts about the stochastically timed process algebra EMPA which will be used in the rest of the paper. In Sect. 3.1 we introduce the structure of actions together with a classification of actions based on their components. In Sect. 3.2 we define the syntax of terms and we explain the meaning of each operator. In Sect. 3.3 we informally present the integrated interleaving semantics as well as its functional and performance projections. In Sect. 3.4 we show how value passing is supported by the language. In Sect. 3.5 we briefly describe the architecture of TwoTowers [BCSS98], the EMPA based software tool which has been used to assess the performance of the audio mechanism introduced in the previous section. Finally, we conclude in Sect. 3.6 by considering simulation, which is the technique we have actually used to predict the percentage of audio packets that are received in time for being played out.

3.1. Structure of Actions

The building blocks of EMPA are *actions*. Each action is a pair $<a, \tilde{\lambda}>$ where:

- a is the *type* of the action. Types divide actions into *external* and *internal* (denoted by action type τ) depending on whether they can be seen by an external observer or not.
- $\tilde{\lambda}$ is the *rate* of the action, i.e. a concise way to represent the random variable specifying its duration. Based on rates, we have the following classification:
  - *Exponentially timed actions* are actions whose rate is a positive real number. Such a number is interpreted as the parameter of the exponentially distributed random variable specifying the duration of the action.
  - *Immediate actions* are actions whose rate, denoted by $\infty_{l,w}$, is infinite. Such actions have duration zero and each of them is given a *priority level* $l \in \mathbf{N}_+$ and a *weight* $w \in \mathbf{R}_+$ useful for expressing prioritized and probabilistic choices, respectively.
  - *Passive actions* are actions whose rate, denoted by $*$, is undefined. The

duration of a passive action is fixed only by synchronizing it with a non-passive action of the same type.

We denote by $Act = AType \times ARate$ the set of actions, where $AType$ is the set of types and $ARate = \mathbb{R}_+ \cup Inf \cup \{*\}$, with $Inf = \{\infty_{l,w} \mid l \in \mathbb{N}_+ \wedge w \in \mathbb{R}_+\}$, is the set of rates.

3.2. Syntax of Terms

Let $Const$ be a set of *constants* and let $ARFun = \{\varphi : AType \longrightarrow AType \mid \varphi(\tau) = \tau \wedge \varphi(AType - \{\tau\}) \subseteq AType - \{\tau\}\}$ be a set of *action relabeling functions*. The set $\mathcal{L}$ of *process terms* of EMPA is generated by the following syntax

$$E ::= \underline{0} \mid <a, \tilde{\lambda}>.E \mid E/L \mid E[\varphi] \mid E + E \mid E \parallel_S E \mid A$$

where $L, S \subseteq AType - \{\tau\}$ and $A \in Const$. We denote by $\mathcal{G}$ the set of guarded and closed terms of $\mathcal{L}$.

The *null term* "$\underline{0}$" represents a termination or deadlocked state. The *prefix operator* "$<a, \tilde{\lambda}>._$" denotes the sequential composition of an action and a term. The *functional abstraction operator* "$\_/L$" hides the actions whose type belongs to L by changing their type to τ. The *functional relabeling operator* "$\_[\varphi]$" changes the types of actions according to φ. The *alternative composition operator* "$\_ + \_$" expresses a choice between two terms: the choice is resolved according to the *race policy* whenever exponentially timed actions are involved (the fastest one succeeds) or according to the *preselection policy* whenever immediate actions are involved (only the actions having the highest priority level are executable and each of these is given an execution probability proportional to its own weight), while the choice is purely *nondeterministic* whenever passive actions are involved. The *parallel composition operator* "$\_ \parallel_S \_$" expresses the concurrent execution of two terms according to the following synchronization discipline: two actions can synchronize if and only if they have the same type belonging to S and at most one of them is not passive. Finally, constants together with their corresponding defining equations of the form $A \triangleq E$ allow for recursion.

3.3. Integrated and Projected Semantics

The integrated semantics for EMPA is represented by a labeled transition system (LTS for short) whose states are in correspondence with terms and whose labels are actions. The semantics is defined in the interleaving style thanks to the memoryless property of the exponential distribution. The problem with the definition of this semantics is that actions have different priority levels. Let us call the potential move of a given term a pair composed of an action executable by that term when ignoring priority levels and the derivative term obtained by executing that action. To cope with the problem mentioned above, the integrated interleaving semantic model is generated by a two-layer semantics: first all the potential moves of a given term are computed by proceeding by induction on the structure of the term, then those moves having lower priority are discarded while the others become the transitions of the semantic model.

As an example, consider term

$$E \equiv E_1 \parallel_{\{a,b\}} E_2$$

where

$$\begin{aligned} E_1 &\equiv <a, \lambda>.\underline{0} + <b, \infty_{1,1}>.\underline{0} \\ E_2 &\equiv <a, *>.\underline{0} + <b, *>.\underline{0} \end{aligned}$$

Term E describes a simple system composed of two parallel processes which are specified by E_1 and E_2, respectively, and synchronize on actions with type a or b. The process described by E_1 can perform either an exponentially timed action with type a or an immediate action with type b. The process described by E_2 can perform two alternative passive actions having type a and b, respectively. By proceeding by induction on the syntactical structure of E, we have that E_1 has potential moves $(<a, \lambda>, \underline{0})$ and $(<b, \infty_{1,1}>, \underline{0})$, while E_2 has potential moves $(<a, *>, \underline{0})$ and $(<b, *>, \underline{0})$. By applying the semantic rules for the parallel composition operator we derive that E has the same potential moves as E_1. Then $(<a, \lambda>, \underline{0})$ is discarded because immediate actions take precedence over exponentially timed actions by virtue of the adoption of the race policy, hence only the transition $E \xrightarrow{b, \infty_{1,1}} \underline{0} \parallel_{\{a,b\}} \underline{0}$ is generated. It is worth noting that if E_2 were $<a, *>.\underline{0}$, then E would have only the potential move $(<a, \lambda>, \underline{0})$ because a synchronization on b would not be possible.

The *integrated interleaving semantics* of $E \in \mathcal{G}$ is a LTS denoted by $\mathcal{I}[\![E]\!]$ whose states are divided into *tangible*, *vanishing*, *open*, and *absorbing* depending on whether they have outgoing exponentially timed transitions, outgoing immediate transitions, outgoing passive transitions, or no outgoing transitions. $E \in \mathcal{G}$ is said to be *performance closed* if $\mathcal{I}[\![E]\!]$ does not contain passive transitions; we denote by $\mathcal{E}$ the set of performance closed terms of $\mathcal{G}$.

From the integrated interleaving semantic model, two projected semantic models can be obtained. The *functional semantics* of $E \in \mathcal{G}$ is still represented as a LTS $\mathcal{F}[\![E]\!]$ now labeled by action types only, derived from $\mathcal{I}[\![E]\!]$ by dropping action rates. The *performance semantics* of $E \in \mathcal{E}$ is represented as a homogeneous Markov chain basically derived from $\mathcal{I}[\![E]\!]$ by dropping action types. To be more precise, the performance model is given by a homogeneous continuous time Markov chain or a homogeneous discrete time Markov chain depending on whether $\mathcal{I}[\![E]\!]$ contains only exponentially timed or immediate transitions. If both kinds of transition coexist, the performance model is a homogeneous continuous time Markov chain built by applying to $\mathcal{I}[\![E]\!]$ a suitable graph reduction rule, which eliminates vanishing states together with their outgoing immediate transitions and modifies rates of transitions entering such states accordingly.

For a more detailed presentation of the semantic rules for EMPA, the interested reader is referred to [BG98].

3.4. Value Passing

EMPA has been equipped with the capability of passing values, thus resulting in EMPA_{vp}. This extension is necessary when dealing with systems where data play a fundamental role. This is the case for the audio mechanism we are going to model, because e.g. the treament of a packet depends on the value of its timestamp.

In order to allow for value passing, the syntax is modified as follows:

- We introduce actions which can read or write values. As a consequence, beside *unstructured actions* of the form $<a, \tilde{\lambda}>$, we have *input actions* of the form

$<a?(\underline{x}), *>$, where $\underline{x}$ is a vector of typed variables, as well as *output actions* of the form $<a!(\underline{e}), \tilde{\lambda}>$, where $\underline{e}$ is a vector of typed expressions.

- A new operator is added which reflects the influence of data on the execution. This is a *conditional operator* of the form *if* (β) *then* E_1 *else* E_2 where β is a boolean expression.
- The capability of storing values is included. This is achieved by means of *parametrized constant definitions* of the form $A(\underline{x}) \stackrel{\Delta}{=} E$, where $\underline{x}$ is a vector of typed variables, and *parametrized constant invokations* of the form $A(\underline{e})$, where $\underline{e}$ is a vector of typed expressions. More precisely, a constant definition header is of the form $A(\underline{p}; \underline{v})$ where $\underline{p}$ is a vector of formal parameters, each preceded by the corresponding type, and $\underline{v}$ is a vector of local variables, each preceded by the corresponding type.

The semantic rules for EMPA_{vp} are similar to those for EMPA. They generate symbolic LTSs, i.e. LTSs where actions are kept symbolic by means of assignments associated with transitions which set the values of variables. Since the presentation of these semantic rules is outside the scope of this paper, the interested reader is referred to [Ber98].

3.5. TwoTowers

TwoTowers is an EMPA_{vp} based tool for the functional and performance analysis of concurrent systems, which implements the integrated approach proposed in [BDG98]. Its architecture is shown in Fig. 1.

The tool driver includes routines for parsing EMPA_{vp} specifications and performing lexical, syntactic and static semantic checks on the specifications.

The purpose of the integrated kernel is to perform those analyses that require both functional and performance information and therefore cannot be performed by factoring out information to be passed to either the functional or performance kernels. The integrated kernel contains the routines to generate the integrated interleaving semantic model of correct EMPA_{vp} specifications, as well as a routine to check two specifications for equivalence and a facility for simulating specifications in order to derive performance measures.

The functional kernel generates the functional semantic model of correct EMPA_{vp} specifications. The analysis of the LTSs as well as the terms themselves is then executed by a version of CWB-NC [CS96] that was retargeted for EMPA_{vp} using PAC-NC [CMS95]. The functional kernel therefore comprises all the analysis capabilities of CWB-NC including interactive simulation of systems, reachability analysis to check safety properties, model checking in the μ-calculus or CTL, equivalence checking, and preorder checking.

Finally, the performance kernel generates the Markovian semantic model of correct EMPA_{vp} specifications. The analysis of the Markov chains is then carried out by means of MarCA [Ste94]: transient and stationary probability distributions are calculated, then performance indexes are derived using rewards as explained in [Ber97].

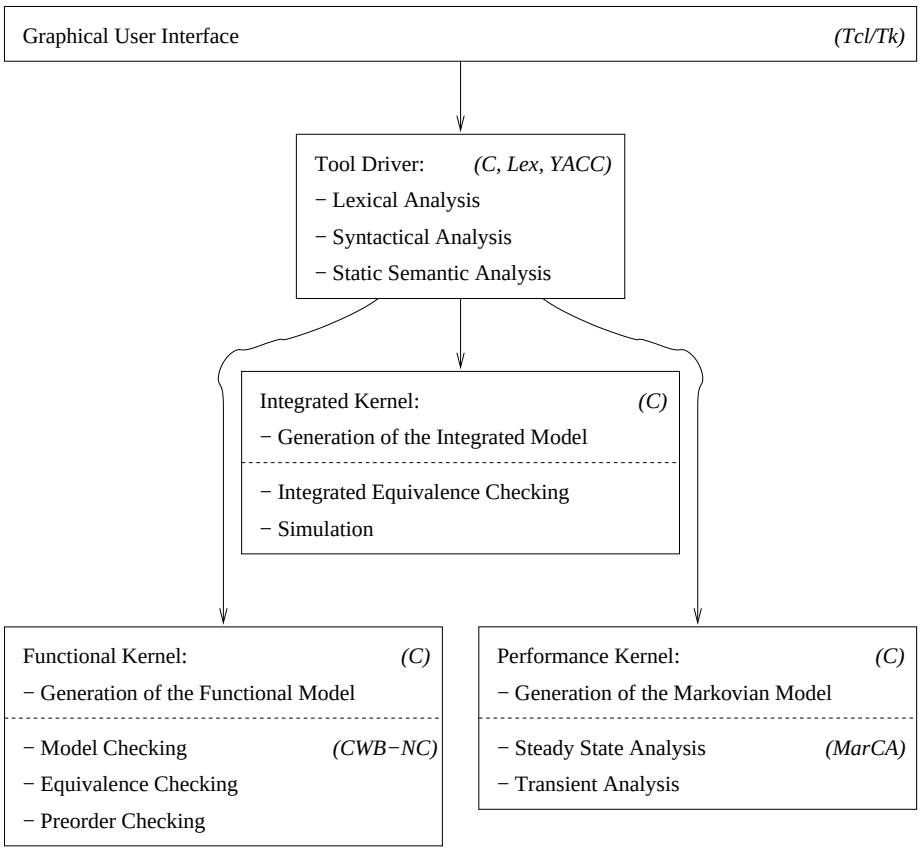

Fig. 1. Architecture of TwoTowers

3.6. Simulation

It is well known that several cases exist for which in general a numerical Markovian analysis cannot be conducted (with the performance kernel embodied in TwoTowers). For example, as mentioned in Sect. 2, packet transmission delays are typically assumed to follow a Gaussian distribution [BCV95]. With EMPA_{vp} Gaussian distributions cannot be modeled within actions, since these can only have either exponentially distributed or zero duration. Instead, Gaussian distributions can be accommodated in EMPA_{vp} using value passing based expressions. However, the performance models derived from the value passing based algebraic terms with the performance kernel of TwoTowers would be Markov chains where information about Gaussian distributions is missing. Consequently, a numerical analysis based on the above mentioned Markov chains would be meaningless. Hence the decision of carrying out a performance simulative analysis. To this purpose, for our case study we have exploited the simulator of the integrated kernel. Such a simulator is equipped with a set of random number generators and implements the method of independent replications [Wel83].

4. Predicting the Performance of the Audio Mechanism

In this section we formally describe the audio mechanism sketched in Sect. 2 with EMPA_{vp} and we conduct with TwoTowers a simulative analysis of its performance.

As mentioned in Sect. 1, the use of EMPA_{vp} has played a fundamental role in the design process of the audio mechanism. As detailed in Sect. 2, the initial idea behind the audio mechanism was that of synchronizing the receiver's and the sender's clocks (taking the instantaneous network delay into account) only at the beginning of the first talkspurt period in the audio conversation. So, this scenario was modeled using EMPA_{vp} in order to predict its performance. However, the obtained simulative results revealed the inefficiency of the approach. Based on those results, we decided to modify the audio mechanism by periodically repeating the synchronization activity in order to account for the highly fluctuant network jitter delay. Also this scenario was modeled with EMPA_{vp} and its performance evaluated. In this case, the obtained simulative results were satisfactory.

In the remainder of this section the algebraic models of the two above mentioned scenarios are presented together with the related simulative results. Finally, such results are contrasted with results obtained via experiments that have been conducted on the field in order to definitely assess the adequacy of the audio mechanism.

4.1. Initial Synchronization

Three entities are involved in the modeling of our mechanism for transmitting packetized audio: the sender (termed *ISSender* in the specification below), the receiver (termed *ISReceiver*), and the channel (termed *ISChannel*) over which packets are transmitted. The interaction between the sender and the channel is modeled through the set SC of the following actions:

- *prepare_packet* denoting the fact that a packet is put on the channel.
- *prepare_probe* denoting the fact that a probe message is put on the channel.
- *trans_response* denoting the reception of a synchronization response at the sender.
- *prepare_install* denoting the fact that an install message is put on the channel.
- *trans_ack* denoting the reception of an acknowledgement at the sender.

The interaction between the receiver and the channel is modeled through the set CR of the following actions:

- *trans_packet* denoting the reception of a packet at the receiver.
- *trans_probe* denoting the reception of a probe message at the receiver.
- *prepare_response* denoting the fact that a synchronization response is put on the channel.
- *trans_install* denoting the reception of an install message at the receiver.
- *prepare_ack* denoting the fact that an acknowledgement is put on the channel.

Hence, based on the above actions, the overall scenario for our audio mechanism can be modeled as follows:

- $PacketizedAudio(int\ clock\_s, int\ clock\_r;) \triangleq$
  $ISSender(clock\_s) \parallel_{SC} ISChannel \parallel_{CR} ISReceiver(clock\_r)$
- SC={$prepare\_packet$,$prepare\_probe$,$trans\_response$,$prepare\_install$,$trans\_ack$}
- CR = {$trans\_packet$,$trans\_probe$,$prepare\_response$,$trans\_install$,$prepare\_ack$}

As mentioned in Sect. 2, at the beginning the clock of the sender and the clock of the receiver get synchronized. To accomplish this, the sender sends to the receiver a so called probe message carrying as a timestamp the value of its clock. When receiving such a probe message, the receiver sets its clock to the value of the timestamp and sends a response message back to the sender. Upon reception of the response, the sender transmits to the receiver an install message carrying as a timestamp the difference between the current value of its clock and the value of its clock when it sent the probe message (i.e. the RTT value). Finally, upon reception of the install message, the receiver decreases its clock by the value of the timestamp and sends an acknowledgement back to the sender. The initial synchronization of the two clocks may be therefore modeled as follows:

- $ISSender(int\ clock\_s;) \triangleq$
  $<prepare\_probe!(clock\_s), \infty_{1,1}>.ISSender'(clock\_s + 1, clock\_s)$
  $ISSender'(int\ clock\_s, int\ old\_clock\_s;) \triangleq$
  $<trans\_response, *>.<prepare\_install!(clock\_s - old\_clock\_s), \infty_{1,1}>.$
  $ISSender''(clock\_s + 1)\ +$
  $<elapse\_tick, \infty_{1,1}>.ISSender'(clock\_s + 1, old\_clock\_s)$
  $ISSender''(int\ clock\_s;) \triangleq$
  $<trans\_ack, *>.Sender(clock\_s + 1)\ +$
  $<elapse\_tick, \infty_{1,1}>.ISSender''(clock\_s + 1)$
- $ISChannel(; int\ timestamp) \triangleq$
  $<prepare\_probe?(timestamp), *>.ISChannel'(timestamp, 0, gauss(100, 7))$
  $ISChannel'(int\ timestamp, int\ clock\_c, real\ delivery\_time;) \triangleq$
  if $(delivery\_time \leq clock\_c)$ *then*
  $<trans\_probe!(timestamp), \infty_{1,1}>.<prepare\_response, *>.$
  $ISChannel''(0, gauss(100, 7))$
  else
  $<elapse\_tick, \infty_{1,1}>.ISChannel'(timestamp, clock\_c{+}1, delivery\_time)$
  $ISChannel''(int\ clock\_c, real\ delivery\_time; int\ timestamp) \triangleq$
  if $(delivery\_time \leq clock\_c)$ *then*
  $<trans\_response, \infty_{1,1}>.<prepare\_install?(timestamp), *>.$
  $ISChannel'''(timestamp, 0, gauss(100, 7))$
  else
  $<elapse\_tick, \infty_{1,1}>.ISChannel''(clock\_c + 1, delivery\_time)$
  $ISChannel'''(int\ timestamp, int\ clock\_c, real\ delivery\_time;) \triangleq$
  if $(delivery\_time \leq clock\_c)$ *then*
  $<trans\_install!(timestamp), \infty_{1,1}>.<prepare\_ack, *>.$
  $ISChannel''''(0, gauss(100, 7))$
  else
  $<elapse\_tick, \infty_{1,1}>.$
  $ISChannel'''(timestamp, clock\_c + 1, delivery\_time)$
  $ISChannel''''(int\ clock\_c, real\ delivery\_time;) \triangleq$

$$\begin{array}{l} \quad \textit{if } (\textit{delivery\_time} \leq \textit{clock\_c}) \textit{ then} \\ \quad\quad <\textit{trans\_ack}, \infty_{1,1}>.\textit{Channel}(0, [], []) \\ \quad \textit{else} \\ \quad\quad <\textit{elapse\_tick}, \infty_{1,1}>.\textit{ISChannel}''''(\textit{clock\_c} + 1, \textit{delivery\_time}) \end{array}$$

- $\begin{array}{l} \textit{ISReceiver}(\textit{int clock\_r}; \textit{int timestamp}) \triangleq \\ \quad <\textit{trans\_probe}?(\textit{timestamp}), *>.<\textit{prepare\_response}, \infty_{1,1}>. \\ \quad\quad \textit{ISReceiver}'(\textit{timestamp}) \\ \textit{ISReceiver}'(\textit{int clock\_r}; \textit{int timestamp}) \triangleq \\ \quad <\textit{trans\_install}?(\textit{timestamp}), *>.<\textit{prepare\_ack}, \infty_{1,1}>. \\ \quad\quad \textit{Receiver}(\textit{clock\_r} - \textit{timestamp}, []) \ + \\ \quad <\textit{elapse\_tick}, \infty_{1,1}>.\textit{ISReceiver}'(\textit{clock\_r} + 1) \end{array}$

A few comments about the specification above are now in order. In particular, note that all the nonpassive actions are immediate. Moreover, for the purpose of this paper, we have decided to use a Gaussian distribution with mean 100 msec and standard deviation 7 msec to model the message transmission time in our simulation. The reasons for these choices are the following. First it is worth mentioning that, even if our audio mechanism has been designed and implemented to cope with general network traffic scenarios, a Gaussian distribution has been selected to carry out the simulation, following the widely accepted audio traffic assumption detailed in Sect. 2, so that performance comparisons with other audio mechanisms have been made possible [Roc99]. Second, the specific values of the Gaussian distribution parameters (the mean delay and its associated standard deviation) have been chosen based on several experiments conducted over a UDP connection between Cesena (a remote site of the University of Bologna) and Geneva (Switzerland). Finally, we would like to point out that the modeling of the Gaussian distribution above has been made possible only using the value passing capability embedded in EMPA_{vp}. In fact, even if only exponential distributions might be directly expressed, Gaussian durations have been represented by storing their values into constant parameters (e.g. *delivery_time*) and using clock variables (e.g. *clock_c*) to determine when the corresponding actions (e.g. *trans_probe*) have to be executed. The same modeling technique will be used in the rest of the formal description of the mechanism.

After the initial synchronization has been carried out, the sender begins generating and trasmitting audio packets over the channel every 20 msec. The following is the formal specification of the above mentioned sender's activity:

- $\begin{array}{l} \textit{Sender}(\textit{int clock\_s};) \triangleq \\ \quad \textit{PacketGen}(\textit{clock\_s}, \textit{clock\_s} + 20) \end{array}$
- $\begin{array}{l} \textit{PacketGen}(\textit{int clock\_s}, \textit{int trans\_time};) \triangleq \\ \quad \textit{if } (\textit{clock\_s} = \textit{trans\_time}) \textit{ then} \\ \quad\quad <\textit{prepare\_packet}!(\textit{clock\_s}), \infty_{1,1}>. \\ \quad\quad\quad \textit{PacketGen}(\textit{clock\_s} + 1, \textit{trans\_time} + 20) \\ \quad \textit{else} \\ \quad\quad <\textit{elapse\_tick}, \infty_{1,1}>.\textit{PacketGen}(\textit{clock\_s} + 1, \textit{trans\_time}) \end{array}$

The main activity of the channel consists of: (i) accepting audio packets from the sender, (ii) sampling their delivery time according to the Gaussian distribution above, (iii) queueing them in an unbounded buffer according to

their delivery time, and (iv) finally delivering them at the scheduled time. The specification of the activity of the channel follows below:

- $Channel(int\ clock\_c, list(int)\ packet\_l, list(real)\ time\_l; int\ timestamp) \triangleq$
  $<prepare\_packet?(timestamp), *>.$
  $DeliveryCheck(clock\_c, insert(timestamp, packet\_l),$
  $insert(clock\_c + gauss(100, 7), time\_l))\ +$
  $<elapse\_tick, \infty_{1,1}>.DeliveryCheck(clock\_c, packet\_l, time\_l)$
- $DeliveryCheck(int\ clock\_c, list(int)\ packet\_l, list(real)\ time\_l;) \triangleq$
  $if\ ((time\_l \neq [])\ and\ (first(time\_l) \leq clock\_c))\ then$
  $<trans\_packet!(first(packet\_l)), \infty_{1,1}>.$
  $DeliveryCheck(clock\_c, tail(packet\_l), tail(time\_l))$
  $else$
  $Channel(clock\_c + 1, packet\_l, time\_l)$

The receiver activity may be summarized as follows. The receiver is continuously waiting for audio packets. Upon reception, the packet is discarded if the value of its timestamp (which corresponds to the value of the sender's clock at the time the packet was sent) is smaller of the value of the receiver's clock. Every msec, the queue of accepted packets is checked in order to detect whether there is any packet to be played out. The specification of this activity is shown below:

- $Receiver(int\ clock\_r, list(int)\ packet\_l; int\ timestamp) \triangleq$
  $<trans\_packet?(timestamp), *>.$
  $(if\ (timestamp \geq clock\_r)\ then$
  $PlayoutCheck(clock\_r, insert(timestamp, packet\_l))$
  $else$
  $PlayoutCheck(clock\_r, packet\_l))\ +$
  $<elapse\_tick, \infty_{1,1}>.PlayoutCheck(clock\_r, packet\_l)$
- $PlayoutCheck(int\ clock\_r, list(int)\ packet\_l;) \triangleq$
  $if\ ((packet\_l \neq [])\ and\ (first(packet\_l) = clock\_r))\ then$
  $<playout\_packet, \infty_{1,1}>.PlayoutCheck(clock\_r, tail(packet\_l))$
  $else$
  $Receiver(clock\_r + 1, packet\_l)$

The specification above has been given as an input to TwoTowers in order to assess the percentage of packets that are played out. We conducted 10 experiments, each consisting of 20 simulation runs concerning a period of 30 sec. The results are in Table 1, where also 90% confidence intervals are shown. As it can be seen, about 25% of packets are discarded in case of initial synchronization only. These results revealed during the design of the audio mechanism the inadequacy of performing only an initial synchronization.

4.2. Periodic Synchronization

In this section the activity of periodic synchronization described in Sect. 2 is actually modeled. In fact, unlike the previous scenario, the mechanism of periodic synchronization mentioned above requires the sender's clock and the receiver's clock to get synchronized every second after the initial synchronization has taken place. A nice feature of $EMPA_{vp}$ is that we do not have to start from scratch to

Table 1. Initial synchronization: simulation results

experiment	estimate	90% confidence interval
1	76.527147	[76.329880, 76.724415]
2	73.829673	[73.684253, 73.975093]
3	71.786939	[71.606600, 71.967279]
4	71.598548	[71.406887, 71.790209]
5	78.935170	[78.775455, 79.094885]
6	70.829697	[70.626774, 71.032620]
7	72.368819	[72.159914, 72.577724]
8	78.268155	[78.073999, 78.462310]
9	74.422198	[74.214966, 74.629431]
10	74.518115	[74.290436, 74.745794]

develop the formal description of the mechanism, but we can exploit the model for the case of initial synchronization only. Moreover, since EMPA_{vp} supports compositional modeling, all we have to do in order to accommodate periodic synchronizations is to locally modify the already specified terms *Sender*, *Channel*, and *Receiver*.

In particular, as far as the modeling of the sender is concerned (i.e. *Sender*), track must be kept of the next synchronization time at the sending site which is now in charge of implementing the corresponding synchronization procedure accordingly. The specification reported below represents the above mentioned activity of the sender:

- $Sender(int\ clock\_s;) \triangleq$
  $PacketGen(clock\_s, clock\_s + 20, clock\_s + 1000)$
- $PacketGen(int\ clock\_s, int\ trans\_time, int\ synch\_time;) \triangleq$
  if $(clock\_s \geq synch\_time)$ *then*
  $<prepare\_probe!(clock\_s), \infty_{2,1}>.$
  $PacketGen'(clock\_s + 1, clock\_s, trans\_time, synch\_time)$
  else
  if $(clock\_s \geq trans\_time)$ *then*
  $<prepare\_packet!(clock\_s), \infty_{2,1}>.$
  $PacketGen(clock\_s + 1, trans\_time + 20, synch\_time)$
  else
  $<elapse\_tick, \infty_{2,1}>.$
  $PacketGen(clock\_s + 1, trans\_time, synch\_time)$

  $PacketGen'(int\ clock\_s, int\ old\_clock\_s, int\ trans\_time, int\ synch\_time;) \triangleq$
  if $(clock\_s \geq trans\_time)$ *then*
  $<prepare\_packet!(clock\_s), \infty_{2,1}>.$
  $PacketGen'(clock\_s + 1, old\_clock\_s, trans\_time + 20, synch\_time)$
  else
  $(<trans\_response, *>.<prepare\_install!(clock\_s - old\_clock\_s), \infty_{2,1}>.$
  $PacketGen''(clock\_s + 1, trans\_time, synch\_time)\ +$
  $<elapse\_tick, \infty_{2,1}>.$
  $PacketGen'(clock\_s + 1, old\_clock\_s, trans\_time, synch\_time))$

  $PacketGen''(int\ clock\_s, int\ trans\_time, int\ synch\_time;) \triangleq$
  if $(clock\_s \geq trans\_time)$ *then*
  $<prepare\_packet!(clock\_s), \infty_{2,1}>.$

$$PacketGen''(clock\_s + 1, trans\_time + 20, synch\_time)$$
$$\textit{else}$$
$$(<trans\_ack, *>.$$
$$PacketGen(clock\_s + 1, trans\_time, synch\_time + 1000) \ +$$
$$<elapse\_tick, \infty_{2,1}>.$$
$$PacketGen''(clock\_s + 1, trans\_time, synch\_time))$$

The channel behaves as in the case of initial synchronization only, but in addition it must also manage probe, response, install, and acknowledgement messages due to periodic synchronization activities. To accomplish this, the model for the channel given in Sect. 4.1 may be simply reused for each of the above mentioned types of message, with in addition the value of the delivery time of the message stored in an appropriate variable. Thus, the formal specification of *Channel* is given below. Note that, for the sake of brevity, in the following specification only the terms corresponding to the probe and response messages are reported. The specification for the install and acknowledgement messages can be obtained straightforwardly.

- $Channel(int\ clock\_c, list(int)\ packet\_l, list(real)\ time\_l; int\ timestamp) \triangleq$
  $<prepare\_packet?(timestamp), *>.$
  $DeliveryCheck(clock\_c, insert(timestamp, packet\_l),$
  $insert(clock\_c + gauss(100, 7), time\_l)) \ +$
  $<prepare\_probe?(timestamp), *>.$
  $DeliveryCheckProbe(clock\_c, packet\_l, time\_l, timestamp,$
  $clock\_c + gauss(100, 7)) \ +$
  $<prepare\_response, *>.$
  $DeliveryCheckResponse(clock\_c, packet\_l, time\_l,$
  $clock\_c + gauss(100, 7)) \ +$
  $<prepare\_install?(timestamp), *>.$
  $DeliveryCheckInstall(clock\_c, packet\_l, time\_l, timestamp,$
  $clock\_c + gauss(100, 7)) \ +$
  $<prepare\_ack, *>.$
  $DeliveryCheckAck(clock\_c, packet\_l, time\_l,$
  $clock\_c + gauss(100, 7)) \ +$
  $<elapse\_tick, \infty_{2,1}>.DeliveryCheck(clock\_c, packet\_l, time\_l)$
  $ChannelProbe(int\ clock\_c, list(int)\ packet\_l, list(real)\ time\_l,$
  $int\ timestamp\_synch, real\ delivery\_time; int\ timestamp) \triangleq$
  $\textit{if } (delivery\_time \leq clock\_c) \textit{ then}$
  $<trans\_probe!(timestamp\_synch), \infty_{2,1}>.$
  $DeliveryCheck(clock\_c, packet\_l, time\_l)$
  $\textit{else}$
  $(<prepare\_packet?(timestamp), *>.$
  $DeliveryCheckProbe(clock\_c, insert(timestamp, packet\_l),$
  $insert(clock\_c + gauss(100, 7), time\_l),$
  $timestamp\_synch, delivery\_time) \ +$
  $<elapse\_tick, \infty_{2,1}>.$
  $DeliveryCheckProbe(clock\_c, packet\_l, time\_l,$
  $timestamp\_synch, delivery\_time))$
  $ChannelResponse(int\ clock\_c, list(int)\ packet\_l, list(real)\ time\_l,$
  $real\ delivery\_time; int\ timestamp) \triangleq$

$$
\begin{array}{l}
\quad \textit{if } (\textit{delivery\_time} \leq \textit{clock\_c}) \textit{ then} \\
\quad\quad (<\textit{trans\_response}, \infty_{2,1}>.\textit{DeliveryCheck}(\textit{clock\_c}, \textit{packet\_l}, \textit{time\_l}) \; + \\
\quad\quad <\textit{prepare\_packet?}(\textit{timestamp}), *>.<\textit{trans\_response}, \infty_{2,1}>. \\
\quad\quad\quad \textit{DeliveryCheck}(\textit{clock\_c}, \textit{insert}(\textit{timestamp}, \textit{packet\_l}), \\
\quad\quad\quad\quad\quad\quad \textit{insert}(\textit{clock\_c} + \textit{gauss}(100, 7), \textit{time\_l}))) \\
\quad \textit{else} \\
\quad\quad (<\textit{prepare\_packet?}(\textit{timestamp}), *>. \\
\quad\quad\quad \textit{DeliveryCheckResponse}(\textit{clock\_c}, \textit{insert}(\textit{timestamp}, \textit{packet\_l}), \\
\quad\quad\quad\quad\quad\quad\quad\quad \textit{insert}(\textit{clock\_c} + \textit{gauss}(100, 7), \textit{time\_l}), \\
\quad\quad\quad\quad\quad\quad\quad\quad \textit{delivery\_time}) \; + \\
\quad\quad <\textit{elapse\_tick}, \infty_{2,1}>. \\
\quad\quad\quad \textit{DeliveryCheckResponse}(\textit{clock\_c}, \textit{packet\_l}, \textit{time\_l}, \textit{delivery\_time}))
\end{array}
$$

- $\textit{DeliveryCheck}(\textit{int clock\_c}, \textit{list}(\textit{int}) \textit{ packet\_l}, \textit{list}(\textit{real}) \textit{ time\_l};) \triangleq$

$$
\begin{array}{l}
\quad \textit{if } ((\textit{time\_l} \neq []) \textit{ and } (\textit{first}(\textit{time\_l}) \leq \textit{clock\_c})) \textit{ then} \\
\quad\quad (<\textit{trans\_packet}!(\textit{first}(\textit{packet\_l})), \infty_{2,1}>. \\
\quad\quad\quad \textit{DeliveryCheck}(\textit{clock\_c}, \textit{tail}(\textit{packet\_l}), \textit{tail}(\textit{time\_l})) \; + \\
\quad\quad <\textit{elapse\_tick}, \infty_{1,1}>.\textit{Channel}(\textit{clock\_c} + 1, \textit{packet\_l}, \textit{time\_l})) \\
\quad \textit{else} \\
\quad\quad \textit{Channel}(\textit{clock\_c} + 1, \textit{packet\_l}, \textit{time\_l}) \\
\textit{DeliveryCheckProbe}(\textit{int clock\_c}, \textit{list}(\textit{int}) \textit{ packet\_l}, \textit{list}(\textit{real}) \textit{ time\_l}, \\
\qquad\qquad\qquad \textit{int timestamp\_synch}, \textit{real delivery\_time};) \triangleq \\
\quad \textit{if } ((\textit{time\_l} \neq []) \textit{ and } (\textit{first}(\textit{time\_l}) \leq \textit{clock\_c})) \textit{ then} \\
\quad\quad <\textit{trans\_packet}!(\textit{first}(\textit{packet\_l})), \infty_{2,1}>. \\
\quad\quad\quad \textit{DeliveryCheckProbe}(\textit{clock\_c}, \textit{tail}(\textit{packet\_l}), \textit{tail}(\textit{time\_l}), \\
\quad\quad\quad\quad\quad\quad\quad \textit{timestamp\_synch}, \textit{delivery\_time}) \\
\quad \textit{else} \\
\quad\quad \textit{ChannelProbe}(\textit{clock\_c} + 1, \textit{packet\_l}, \textit{time\_l}, \textit{timestamp\_synch}, \\
\quad\quad\quad\quad\quad\quad \textit{delivery\_time}) \\
\textit{DeliveryCheckResponse}(\textit{int clock\_c}, \textit{list}(\textit{int}) \textit{ packet\_l}, \textit{list}(\textit{real}) \textit{ time\_l}, \\
\qquad\qquad\qquad\qquad \textit{real delivery\_time};) \triangleq \\
\quad \textit{if } ((\textit{time\_l} \neq []) \textit{ and } (\textit{first}(\textit{time\_l}) \leq \textit{clock\_c})) \textit{ then} \\
\quad\quad <\textit{trans\_packet}!(\textit{first}(\textit{packet\_l})), \infty_{2,1}>. \\
\quad\quad\quad \textit{DeliveryCheckResponse}(\textit{clock\_c}, \textit{tail}(\textit{packet\_l}), \textit{tail}(\textit{time\_l}), \\
\quad\quad\quad\quad\quad\quad\quad\quad \textit{delivery\_time}) \\
\quad \textit{else} \\
\quad\quad \textit{ChannelResponse}(\textit{clock\_c} + 1, \textit{packet\_l}, \textit{time\_l}, \textit{delivery\_time})
\end{array}
$$

Finally, in the last part of this section the formal modeling of the receiver (i.e. *Receiver*) is reported. Needless to say, also the receiver is involved in the synchronization procedure. From the receiver's viewpoint, the modeling of the periodic synchronization has a crucial difference with respect to the modeling of the initial synchronization. In particular, the value of the timestamp of the probe message of one of the subsequent synchronizations is not assigned to the receiver's clock but stored into a variable which is incremented each time the receiver's clock is incremented. The new value of the receiver's clock is then installed upon reception of the install message. At that time, all the audio packets in the receiver's queue whose timestamp is smaller than the new value are discarded. The specification of the activity of the receiver is as follows:

- $\textit{Receiver}(\textit{int clock\_r}, \textit{list}(\textit{int}) \textit{ packet\_l}; \textit{int timestamp}) \triangleq$

$$
\begin{array}{l}
\quad <trans\_probe?(timestamp), *>.<prepare\_response, \infty_{2,1}>. \\
\qquad PlayoutCheck'(clock\_r, timestamp, packet\_l) \; + \\
\quad <trans\_packet?(timestamp), *>. \\
\qquad (if\ (timestamp \geq clock\_r)\ then \\
\qquad\quad PlayoutCheck(clock\_r, insert(timestamp, packet\_l)) \\
\qquad else \\
\qquad\quad PlayoutCheck(clock\_r, packet\_l)) \; + \\
\quad <elapse\_tick, \infty_{2,1}>.PlayoutCheck(clock\_r, packet\_l) \\
Receiver'(int\ clock\_r, int\ synch\_time, list(int)\ packet\_l; int\ timestamp) \triangleq \\
\quad <trans\_install?(timestamp), *>.<prepare\_ack, \infty_{2,1}>. \\
\qquad PlayoutCheck''(synch\_time - timestamp, packet\_l) \; + \\
\quad <trans\_packet?(timestamp), *>. \\
\qquad (if\ (timestamp \geq clock\_r)\ then \\
\qquad\quad PlayoutCheck'(clock\_r, synch\_time, insert(timestamp, packet\_l)) \\
\qquad else \\
\qquad\quad PlayoutCheck'(clock\_r, synch\_time, packet\_l)) \; + \\
\quad <elapse\_tick, \infty_{2,1}>.PlayoutCheck'(clock\_r, synch\_time, packet\_l)
\end{array}
$$

- $PlayoutCheck(int\ clock\_r, list(int)\ packet\_l;) \triangleq$
  $if\ ((packet\_l \neq [])\ and\ (first(packet\_l) = clock\_r))\ then$
  $<playout\_packet, \infty_{2,1}>.PlayoutCheck(clock\_r, tail(packet\_l))$
  $else$
  $Receiver(clock\_r + 1, packet\_l)$

  $PlayoutCheck'(int\ clock\_r, int\ synch\_time, list(int)\ packet\_l;) \triangleq$
  $if\ ((packet\_l \neq [])\ and\ (first(packet\_l) = clock\_r))\ then$
  $<playout\_packet, \infty_{2,1}>.$
  $PlayoutCheck'(clock\_r, synch\_time, tail(packet\_l))$
  $else$
  $Receiver'(clock\_r + 1, synch\_time + 1, packet\_l)$

  $PlayoutCheck''(int\ clock\_r, list(int)\ packet\_l;) \triangleq$
  $if\ ((packet\_l \neq [])\ and\ (first(packet\_l) \leq clock\_r))\ then$
  $PlayoutCheck''(clock\_r, tail(packet\_l))$
  $else$
  $Receiver(clock\_r + 1, packet\_l)$

Also the specification above has been analyzed by means of TwoTowers in order to assess the percentage of packets that are played out, by conducting 10 experiments each consisting of 20 simulation runs concerning a period of 30 sec. The results, which are reported in Table 2, show that the percentage of discarded packets falls down 4% when performing a synchronization every second.

4.3. Comparison and Conclusion

As mentioned in Sect. 2, a prototype implementation of the playout control mechanism has been carried out through the Unix socket interface and the datagram based UDP protocol. Using such a prototype implementation, an initial extensive experimentation has been conducted in order to test the real performances of the mechanism. In particular, several experiments of audio conversation have been carried out during daytime using a SUN workstation SPARC 5 situated

Table 2. Periodic synchronization: simulation results

experiment	estimate	90% confidence interval
1	96.702746	[96.335040, 97.070452]
2	96.712566	[96.490167, 96.934966]
3	96.453736	[96.183720, 96.723751]
4	96.583136	[96.243839, 96.922433]
5	96.600180	[96.296200, 96.904160]
6	96.612087	[96.282160, 96.942014]
7	96.481827	[96.183756, 96.779897]
8	96.594055	[96.233550, 96.954561]
9	96.480821	[96.154645, 96.806997]
10	96.745196	[96.461490, 97.028903]

at the Laboratory of Computer Science of Cesena (a remote site of the Computer Science Department of the University of Bologna) and a SUN workstation SPARC 10 situated at the CERN in Geneva (Switzerland).

Probably, reporting complete details on the obtained performance results is beyond the scope of this paper. However, in order to contrast those real performance results with the simulation results presented above, it is worth mentioning that during the cited experimentations an acceptable average playout delay of 180-220 msec was obtained, and only rarely a maximum playout delay exceeding 400 msec was imposed by our mechanism.

In addition, a percentage of no more than 4-5% of loss audio packets was observed due to late arrivals. In order to test the efficacy of the proposed method, also contemporary experiments were conducted on the same testbed without using any playout control mechanism. In those experiments, a percentage of lost audio packets (due to late arrivals) was experienced ranging from about 10% to almost 40% depending on the traffic conditions.

As it can be noted, the results obtained from the simulative analysis of the formal model of the mechanism correctly predicted the actual performance of the mechanism itself measured on the field. This highlights the importance of using formal description techniques in the initial phase of the design of an algorithm, in order to detect in advance errors or inefficiency which may result in cost increases if discovered too late.

Acknowledgements

We would like to thank the anonymous referees for their helpful comments. This research has been partially funded by MURST, CNR Progetto Strategico "Modelli e Metodi per la Matematica e l'Ingegneria", and CNR Grant n. 98.00387.CT12.

References

[AY96] I. Akyildiz, W. Yen, *"Multimedia Group Synchronization Protocols for Integrated Services Networks"*, in IEEE Journal on Selected Areas in Communications 14:162-173, 1996

[Ber97] M. Bernardo, *"An Algebra-Based Method to Associate Rewards with EMPA Terms"*, in Proc. of the *24th Int. Coll. on Automata, Languages and Programming (ICALP '97)*, LNCS 1256:358-368, Bologna (Italy), 1997

[Ber98] M. Bernardo, *"Value Passing in Stochastically Timed Process Algebras: A Symbolic Approach based on Lookahead"*, Tech. Rep. UBLCS-98-10, University of Bologna (Italy), 1998

[BCSS98] M. Bernardo, W.R. Cleaveland, S.T. Sims, W.J. Stewart, *"TwoTowers: A Tool*

Integrating Functional and Performance Analysis of Concurrent Systems", to appear in Proc. of the *IFIP Joint Int. Conf. on Formal Description Techniques and Protocol Specification, Testing, and Verification (FORTE/PSTV '98)*, Paris (France), 1998

[BDG98] M. Bernardo, L. Donatiello, R. Gorrieri, *"A Formal Approach to the Integration of Performance Aspects in the Modeling and Analysis of Concurrent Systems"*, to appear in Information and Computation, 1998

[BG98] M. Bernardo, R. Gorrieri, *"A Tutorial on EMPA: A Theory of Concurrent Processes with Nondeterminism, Priorities, Probabilities and Time"*, in Theoretical Computer Science 202:1-54, 1998

[BV88] G.V. Bochmann, J. Vaucher, *"Adding Performance Aspects to Specification Languages"*, in Proc. of the *8th Int. Conf. on Protocol Specification, Testing and Verification (PSTV VIII)*, North Holland, pp. 19-31, Atlantic City (NJ), 1988

[BCV95] J. Bolot, H. Crepin, A. Vega Garcia, *"Analysis of Audio Packet Loss on the Internet"*, in Proc. of *Network and Operating System Support for Digital Audio and Video*, pp. 163-174, Durham (NC), 1995

[BV96] J. Bolot, A. Vega Garcia, *"Control Mechanism for Packet Audio in the Internet"*, in Proc. of *IEEE SIGCOMM '96*, San Francisco (CA), 1996

[BV98] J. Bolot, A. Vega Garcia, *"The Case for FEC-based Error Control for Packet Audio in the Internet"*, to appear in ACM Multimedia Systems, 1998

[CMS95] W.R. Cleaveland, E. Madelaine, S.T. Sims, *"A Front-End Generator for Verification Tools"*, in Proc. of the *1st Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '95)*, LNCS 1019:153-173, Aarhus (Denmark), 1995

[CS96] W.R. Cleaveland, S.T. Sims, *"The NCSU Concurrency Workbench"*, in Proc. of the *8th Int. Conf. on Computer Aided Verification (CAV '96)*, LNCS 1102:394-397, New Brunswick (NJ), 1996

[Coh77] D. Cohen, *"Issues in Transnet Packetized Voice Communications"*, in Proc. of the *5th Data Communication Symp.*, pp. 6.10 - 6.13, Snowbird (UT), 1977

[Cri89] F. Cristian, *"Probabilistic Clock Synchronization"*, in Distributed Computing 3:146-158, 1989

[Fer86] D. Ferrari, *"Considerations on the Insularity of Performance Evaluation"*, in IEEE Trans. on Software Engineering 12:678-683, 1986

[GHR93] N. Götz, U. Herzog, M. Rettelbach, *"Multiprocessor and Distributed System Design: the Integration of Functional Specification and Performance Analysis Using Stochastic Process Algebras"*, in Proc. of the *16th Int. Symp. on Computer Performance Modelling, Measurement and Evaluation (PERFORMANCE '93)*, LNCS 729:121-146, Rome (Italy), 1993

[HSK98] V. Hardman, M.A. Sasse, I. Kouvelas, *"Successful Multi-Party Audio Communication over the Internet"*, in Comm. of the ACM 41:74-80, 1998

[Har86] C. Harvey, *"Performance Engineering as an Integral Part of System Design"*, in BT Technology Journal 4:143-147, 1986

[Hil96] J. Hillston, *"A Compositional Approach to Performance Modelling"*, Cambridge University Press, 1996

[JM] V. Jacobson, S. McCanne, *vat*, ftp://ftp.ee.lbl.gov/conferencing/vat/

[MB94] M. Macedonia, D. Brutzmann, *"Mbone Provides Audio and Video across the Internet"*, in IEEE Computer Magazine 21:30-35, 1994

[MKT98] S.B. Moon, J. Kurose, D. Towsley, *"Packet Audio Playout Delay Adjustment: Performance Bounds and Algorithms"*, in ACM Multimedia Systems 6:17-28, 1998

[PR97] F. Panzieri, M. Roccetti, *"Synchronization Support and Group-Membership Services for Reliable Distributed Multimedia Applications"*, in ACM Multimedia Systems 5:1-22, 1997

[RKTS94] R. Ramjee, J. Kurose, D. Towsley, H. Schulzrinne, *"Adaptive Playout Mechanisms for Packetized Audio Applications in Wide-Area Networks"*, in Proc. of *IEEE INFOCOM '94*, Montreal (Canada), 1994

[RKR96] P.V. Rangan, S.S. Kumar, S. Rajan, *"Continuity and Synchronization in MPEG"*, in IEEE Journal on Selected Areas in Communications 14:52-60, 1996

[Roc99] M. Roccetti, *"Experimenting with Real Time Packetized Audio for Distance Learning over Wide Area Networks"*, to appear in Proc. of the *1999 Western Multiconf. on Computer Simulation*, San Francisco (CA), 1999

[RGSPB98] M. Roccetti, V. Ghini, P. Salomoni, G. Pau, M.E. Bonfigli, *"Design and Experimental Evaluation of an Adaptive Playout Delay Control Mechanism for Pack-*

etized Audio for Use over the Internet", Tech. Rep. UBLCS-98-4, University of Bologna (Italy), 1998 (submitted for publication)

[Sch92] H. Schulzrinne, *"Voice Communication across the Internet: a Network Voice Terminal"*, Tech. Rep., University of Massachusetts, Amherst (MA), 1992

[SCFJ95] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, *"RTP: A Transport Protocol for Real-Time Applications"*, Request for Comments 1889, IETF, Audio-Video WG, 1995

[Ste94] W.J. Stewart, *"Introduction to the Numerical Solution of Markov Chains"*, Princeton University Press, 1994

[Wel83] P.D. Welch, *"The Statistical Analysis of Simulation Results"*, in *"Computer Performance Modeling Handbook"*, S.S. Lavenberg editor, Academic Press, pp. 267-329, 1983

[YK82] Y. Yemini, J. Kurose, *"Towards the Unification of the Functional and Performance Analysis of Protocols, or Is the Alternating-Bit Protocol Really Correct?"*, in Proc. of the *2nd Int. Conf. on Protocol Specification, Testing and Verification (PSTV II)*, North Holland, 1982

[ZH93] L. Zhang, *"RSVP: A New Resource Reservation Protocol"*, in IEEE Network Magazine 7:8-18, 1993