A Study about Trade-off between Performance and Security in an Internet Audio Mechanism

Alessandro Aldini[1] and Roberto Gorrieri[2]

[1] Istituto di Scienze e Tecnologie dell'Informazione,
Università di Urbino, Italy
aldini@sti.uniurb.it
[2] Dipartimento di Scienze dell'Informazione,
Università di Bologna, Italy
gorrieri@cs.unibo.it

Abstract. We study the nature of the relationship between performance measures and privacy guarantees in the case study of an adaptive protocol for the secure transmission of real-time audio over the Internet. The analysis is conducted on a process-algebraic description of the audio mechanism by following a methodology that allows the modeler to (*i*) employ the noninterference approach to information flow theory for the analysis of security requirements, and (*ii*) derive performance measures obtained through markovian analysis techniques. The main result we present is that the analysis of performance properties helps to estimate the effectiveness (and to find a related countermove) of an attack that is captured by the security analysis.

1 Introduction

The analysis of Quality of Service (QoS) properties and of security conditions are two important problems that arise in the modeling phase of computer systems, especially when dealing with applications working over public, untrusted networks and with strict functional and performance requirements [SDS01]. In this paper, we focus on the potential relationship between these two aspects in the context of a protocol for audio communications over IP, developed in a software tool called BoAT [RGPSB01,AGR01,AMR03,AGR03]. We chose this case study because the success of voice over IP services strictly depends on their capability of coping with unforeseeable environment constraints, typical of public wide area networks, such as variable queueing delays, packet loss, and lack of security guarantees. In particular, BoAT aims at providing an audio quality that is comparable to that of the circuit-switched telephone system, and a security level comparable to that of a private channel, by following two main strategies. On the one hand, BoAT employs a mechanism that adaptively adjusts the playout of the received audio packets to the fluctuating network delays in order to offer at the receiving site the same audio quality as that produced at the sending site under any scenario. The core of such a mechanism is represented by a handshaking protocol, which is used to exchange estimations of the traffic conditions between

the involved parties. A simulative comparison between this novel mechanism and other existing adaptive algorithms revealed that BoAT succeeds in offering an adequate QoS [ABGR01]. On the other hand, BoAT employs a cryptographic protocol, which adopts a lightweight securing mechanism based on the use of a stream cipher (see, e.g., [Schn96]) and of a sequence of secret keys needed to secure the conversation. The brief lifetime of each secret key (which is exchanged in the same handshaking packets used by the adaptive playout algorithm and is used by the stream cipher to encrypt few hundreds of bytes only) is the main feature of BoAT that strengthens the robustness of the cryptographic algorithm against cryptanalysis attacks. Moreover, as experimental studies have emphasized [AGR01,AGR03], such an approach suffers a computational overhead that is quite negligible with respect to other securing tools proposed in the literature.

The formal analysis we conduct in this paper aims at studying the effectiveness of the securing mechanism of BoAT in the light of the considerations above. The results we obtain have a twofold interest. On the one hand, the security analysis reveals an attack by an adversary that tries to intercept all the handshaking packets containing the secret keys, thus blocking the re-keying mechanism. On the other hand, the performance analysis shows that the throughput of the playout control algorithm, expressed in terms of number of audio packets delivered (and played out) per sec, and the throughput of the privacy infrastructure, expressed in terms of number of secret keys exchanged per sec, are strictly coupled. In particular, by analyzing the relation between these two measures, we can estimate the privacy level of the system and possibly detect the attack described above. Moreover, it is worth noting that the results of the same formal analysis help not only to reveal the weaknesses of the audio protocol but also to single out and evaluate a strategy that makes it vain the attack of the adversary.

The study is conducted on a formal description of BoAT expressed in a probabilistic process algebra [BA03]. Timed, probabilistic, and stochastic extensions of process algebras (see, e.g., [HHHMR94,HS95,BDG98]) have been introduced that formally describe both functional and performance aspects on the same system model, in order to bridge the gap between formal verification and quantitative validation. In this setting, the novelty of our approach is that both performance related properties and information flow security properties can be evaluated on the same system model. From a performance standpoint, we can derive a Discrete Time Markov Chain from the algebraic specification of BoAT and then we can evaluate steady-state based performance measures (expressible by attaching rewards to actions) through markovian analysis techniques [Ber99]. Such an analysis is automatically conducted with the software tool TwoTowers [BCSS98]. From a security viewpoint, the same system model is analyzed by employing a probabilistic extension [ABG02] of the noninterference approach [GM82,FG95] to the information flow theory.

The rest of the paper is organized as follows. In Sect. 2, we briefly recall BoAT and its main features. In Sect. 3, we describe the probabilistic process-algebraic framework based on which we conduct the analysis. Then, in Sect. 4 we present the algebraic specification of BoAT and we report on the results obtained

by analyzing such a model. Finally, in Sect. 5 some conclusions terminate the paper.

2 A Secure Real-time Audio Protocol: BoAT

In this section, we briefly describe BoAT, an adaptive protocol proposed for the trusted, private transmission of real-time audio over public networks like the Internet [RGPSB01,AGR01,AMR03,AGR03].

On the one hand, BoAT provides an adaptive control mechanism that supports quality guarantees of Internet voice software in spite of highly fluctuant transmission delay variation and packet loss. The goal of providing a synchronous playout of audio packets at the receiving site is typically achieved by buffering the received audio packets and by delaying their playout time in order to compensate for variable network delays. With respect to other adaptive audio algorithms (see, e.g., [Schu92,HSK98]), BoAT dynamically adapts the 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.

On the other hand, BoAT embodies a privacy infrastructure that provides authentication of the two involved parties, secrecy and integrity of the protocol data and of the audio conversation. This is obtained with a minimal per-packet communication overhead that does not jeopardize the performance guaranteed by the adaptive playout mechanism.

Succinctly, the core of the mechanism of BoAT is based on a three way handshake protocol periodically performed during the conversation between the sender and the receiver. Thanks to such a packet exchange, usually performed once a second, the two parties obtain (i) an estimation (called Δ) of the *upper bound* for the packet transmission delay experienced during the audio communication, and (ii) a new secret key used to secure the conversation.

A correct estimation of Δ represents the key factor for the success of the playout control mechanism. Indeed, Δ directly influences the talkspurt playout delay, which is dynamically set at the receiving site from one talkspurt to the next one on the basis of the result of the handshaking phase. A description of such a mechanism is as follows. The sender begins the synchronization policy by sending a *probe* packet timestamped with the time value t_s shown by its own clock. At the reception of this packet, the receiver sets its own clock to t_s and sends immediately back a *response* packet timestamped with the same value t_s. Upon receiving the response packet, the sender checks if it is related to the last *probe* message sent to the receiver and, in such a case, computes the value of the round trip time (RTT) by subtracting the value of the timestamp t_s from the current value of its local clock. At that moment, the difference between the sender clock and the receiver clock is equal to an unknown quantity (say t_0), which may range from a theoretical lower bound of 0 (i.e., all the RTT has been consumed on the way back from the receiver to the sender), and a theoretical upper bound

of RTT (i.e., all the RTT has been consumed during the transmission of the *probe* packet). The final packet of the handshaking phase sent by the sender to the receiver is an *installation* packet, with attached the calculated RTT value and the timestamp t_s. Upon receiving this packet, the receiver sets the time of its local clock, by subtracting from the current value of its local clock the value of the transmitted RTT. At that moment, the difference between the sender clock and the receiver clock is equal to a value given by $\Delta = t_0 + RTT$, where Δ ranges in the interval $[RTT, 2 \times RTT]$, depending on the unknown value of t_0, that in turn may range in the interval $[0, RTT]$. Usually, the installation of a new clock value at the receiving site does not occur during a talkspurt since it may artificially alter the comprehension of the conversation. Therefore, the *install* message is not sent as soon as the *response* packet is received; instead, it is sent during a silence period. Finally, the last step of the handshaking protocol is given by the transmission of a timestamped *acknowledgement* packet from the receiver to the sender. At the end of the protocol, the difference between the sender clock and the receiver clock represents the estimate of an upper bound for the transmission delay that is used to dynamically adjust the playout delay and the buffer size. In particular, each audio packet is timestamped with the value of the sender clock at its generation instant, and such a value also represents the playout instant that must be scheduled by the receiver. Hence, a maximum transmission delay equal to the difference between the two clocks is left to each audio packet to arrive at the receiver in time for its playout. The reader interested in more details and proofs concerning this adaptive playout control mechanism should refer to [RGPSB01,ABGR01].

The other goal of the handshaking protocol is to provide privacy of the audio communication. This is done as follows. A preliminary authentication phase is carried out by the two parties before the conversation (e.g., by resorting to a digital signature scheme). During this step, the trusted parties agree on a secret key, which is used to encrypt the packet exchange of a first handshaking phase that precedes the audio communication. Then, during each handshaking phase i, the authenticated parties agree on a session key K_i (e.g., exchanged in the *install* packet). Whenever the three-way handshake has a positive outcome, K_i is the session key used by a stream cipher to secure the subsequent chunk of conversation. Since the handshaking protocol is periodically started during the conversation, a sequence of keys $\{K_i\}_{i \in \mathbf{N}}$ is generated, where each key has a lifetime equal to the time between two consecutive synchronizations. Details related to such a protocol and to the securing algorithm can be found in [AGR01,AGR03].

Summing up, the synchronization policy is periodically repeated throughout the whole conversation. In particular, in order for the proposed policy to adaptively adjust to the highly fluctuant network conditions, the above mentioned synchronization technique is first carried out prior to the beginning of the conversation, and then repeated about once a second thus preventing the two clocks (possibly equipped with different clock rates) from drifting apart. The proposed protocol guarantees that (i) both playout delay and buffer size are always pro-

portioned to the traffic conditions, and (*ii*) the brief lifetime of each session key makes it harder any cryptanalysis attack conducted by an adversary (see, e.g., [BSW00], where it is shown that a few seconds of conversation are enough to complete a cryptanalysis attack against a stream cipher).

3 A Process-algebraic Framework

3.1 The Probabilistic Calculus

Basic process algebras (and their extensions) are specification languages used to describe in a compositional way the behavior of concurrent systems in order to formally derive their functional (and non-functional) properties. The basic elements of any process algebra are the actions, which in our calculus are syntactically divided into output actions and input actions, and the algebraic operators, which in our calculus are equipped with probabilistic information. The model of probabilities we adopt is a mixture of the generative and reactive approaches of [GSS95]. In particular, we assume the output actions behaving as *generative* actions (a generative process autonomously decides, on the basis of a probability distribution, which action will be executed and how to behave after such an event) and the input actions behaving as *reactive* actions (a reactive process internally reacts to the choice of the action type, say a, performed by the environment, on the basis of a probability distribution associated with the reactive actions of type a it can perform).

Formally, $AType$ is the set of visible action types, ranged over by $a, b, \ldots$. For each visible action type a, we distinguish the (generative) output action a and the (reactive) input action a_*. The set of actions is denoted by Act, ranged over by $\pi, \pi', \ldots$, including the input and the output actions with type in $AType$, and the special action τ, representing the internal, unobservable action. We point out that τ behaves as a generative action, because it expresses an autonomous internal move, which does not react to external stimuli. The set $\mathcal{L}$ of process terms is generated by the syntax:

$$P ::= \underline{0} \mid \pi.P \mid P +^p P \mid P \|^p_S P \mid P\backslash L \mid P/^p_a \mid A$$

where $S, L \subseteq AType$, $a \in AType$, and $p \in]0,1[$. The set $\mathcal{L}$ is ranged over by $P, Q, \ldots$. Constants A are used to specify recursive systems. In general, when defining an algebraic specification, we assume a set of constants defining equations of the form $A \stackrel{\Delta}{=} P$ to be given. In the following, we restrict ourselves to the set of finite state, closed, guarded terms of $\mathcal{L}$, which we call processes [Mil89].

As usual in security models, we distinguish among high-level visible actions and low-level visible actions by defining two disjoint sets $AType_H$ of high-level types and $AType_L$ of low-level types, which form a covering of $AType$, such that the output action a and the input action a_* are high- (low-) level actions if $a \in AType_H$ ($a \in AType_L$). Finally, we say that P is a high-level process if all actions syntactically occurring in the action prefix operators within P are high-level actions.

Now, we informally describe the semantics of the operators and the probabilistic model through some examples (for a formal presentation the reader should refer to [ABG02]).

Example 1. Let us consider the system

$$Writer \parallel^{p}_{\{produce\}} Buffer,$$

described as the interaction of two processes, *Writer* and *Buffer*. The communication interface $\{produce\}$ says that the two processes interact by synchronously executing actions of type *produce*. Each other local action is asynchronously executed by the two processes. Probability p is the parameter of a probabilistic scheduler that, in each system state, decides which of the two processes must be scheduled, i.e. *Writer* with probability p and *Buffer* with probability $1-p$. Now, let us detail each component in isolation. Process *Writer* repeatedly produces new items:

$$Writer \triangleq produce.Writer +^{q} \tau.Writer.$$

The alternative choice operator "$\_ +^{q} \_$" says that process *Writer* can either produce a message (action *produce*) with probability q, or stay idle (action τ) with probability $1-q$, and afterwards behaving as the same process *Writer*. The actions *produce* and τ are *generative*, hence the process itself autonomously decides, on the basis of a probability distribution guided by parameter q, which action will be executed and how to behave after such an event (see Fig. 1(a), showing the labeled transition system associated to process *Writer* in isolation). Process *Buffer*, instead, is ready to accept new incoming items or it stays idle:

$$Buffer \triangleq (produce_{*}.discard.Buffer +^{r} produce_{*}.store.Buffer) +^{r'} \tau.Buffer.$$

The two actions $produce_{*}$ are *reactive*, hence the process reacts internally to the choice of the action type *produce*, performed by its environment, on the basis of a probability distribution associated with the reactive actions of type *produce* it can perform. Whenever the action type *produce* is chosen by the environment, process *Buffer* reacts by choosing either the first action $produce_{*}$ with probability r and then discarding the message (action *discard*), or the second action $produce_{*}$ with probability $1-r$ and then storing the message (action *store*). Alternatively, if process *Buffer* is not accepting items from the environment, the internal action τ is repeatedly executed to model the idle periods of the buffer. The choice between the input actions $produce_{*}$ and such an internal event is nondeterministic (parameter r' is not considered), because the execution of an action $produce_{*}$ is entirely guided by the external environment (see Fig. 1(b), showing the labeled transition system associated to process *Buffer* in isolation).

According to the considerations above, the two processes interact in the composed system as follows. If process *Writer* decides to perform the action *produce*, then process *Buffer* reacts by executing one of its $produce_{*}$ actions. In the initial state of our example (see Fig. 1(c)), the system executes a move of process *Writer* with probability p: it executes either the internal move τ with probability $p \cdot (1-q)$, or the move *produce* with probability $p \cdot q$ (with probability

$p \cdot q \cdot r$ it executes a *produce* action synchronized with the first reactive action of process *Buffer* and with probability $p \cdot q \cdot (1 - r)$ a *produce* action synchronized with the second reactive action of process *Buffer*). On the other hand, the system may schedule with probability $1 - p$ the process *Buffer* by executing its internal action τ (that gets the *entire* probability $1 - p$ associated to process *Buffer*). Afterwards, if, e.g., the winning action is the action *produce* leading to term $Writer \parallel^{p}_{\{produce\}} store.Buffer$, then the system executes either the action *store* with probability $1 - p$, thus reaching the initial state again, or the action τ of process *Writer* (which gets the entire probability p associated to process *Writer*, since it is the only generative action of the left-hand process enabled by the system).

Example 2. Let us consider the system

$$Job \parallel^{q}_{S} Scheduler,$$

where processes *Job* and *Scheduler* interact by synchronizing on actions in the set $S = \{schedule, end\}$. We now detail the several components and their interactions. Process *Job* repeatedly produces new jobs:

$$Job \triangleq schedule.end_{*}.Job.$$

Whenever the synchronizing generative action *schedule* is executed, which models a new job passed to the scheduler component, process *Job* waits for the termination of the job, which is signalled via a synchronization through the reactive action end_{*}, and then behaves as the same process *Job*. Term *Scheduler* is in turn composed of two communicating processes:

$$Scheduler \triangleq (Fetch \parallel^{q'}_{\{pass\}} Exec)/^{p}_{pass}$$

which model a pipeline whose components sequentially execute the received job:

$$Fetch \triangleq schedule_{*}.\tau.pass.Fetch$$

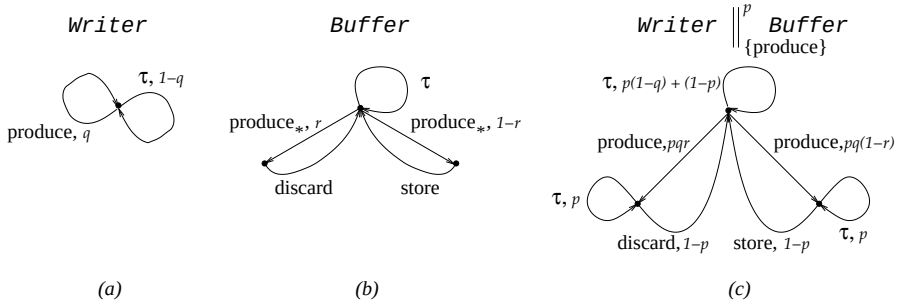

Fig. 1. Labeled transition systems associated to different process terms. Transitions are labeled with an action and a probability, which is equal to 1 if omitted.

fetches a new job (action $schedule_*$), does an internal computation, and then passes the control (action *pass*) to term

$$Exec \stackrel{\Delta}{=} pass_*.\tau.end.Exec,$$

which in turn does some internal computation and then communicates the result by synchronizing with process *Job* (action *end*). The hiding operator "$\_/^p_{pass}$" turns the action *pass*, resulting from the synchronization between processes *Fetch* and *Exec*, into the action τ. This is because the activity modeled by the action *pass* represents an internal computation of term *Scheduler*, which should not be observable by an external component like process *Job*.

In the initial state of our example, the synchronizing action *schedule* (expressing the communication between processes *Job* and *Fetch*) is the only generative action executable by the system, therefore it gets the entire probability 1 to be performed. Moreover, note that process *Exec* is blocked since the action $pass_*$ cannot synchronize. By following the same considerations, we then execute a sequence of τ actions representing the computations of term *Scheduler*, followed by the action *end* leading to the initial state again (see Fig. 2(a)). As a consequence, since probabilistic choices among concurrent processes are never to be performed, parameters q and q', which guide the probabilistic parallel execution of processes *Job*, *Fetch*, and *Exec*, never come into play. Moreover, parameter p is not meaningful (we will shortly explain its use). Now, let us assume that a malfunction prevents the scheduler from informing process *Job* that the current job has been completed. This behavior can be modeled with the restriction operator "$\_\backslash L$" by changing the system as follows:

$$Job \parallel^q_S (Scheduler \backslash \{end\}),$$

where term *Scheduler* is prevented from executing actions of type *end*. Therefore, in term $end_*.Job \parallel^q_S (((Fetch \parallel^{q'}_{\{pass\}} end.Exec)/^p_{pass})\backslash\{end\})$, which is reachable from the initial state by executing the sequence of actions $schedule\,\tau\,\tau\,\tau$, the synchronization on the action of type *end* is not enabled. Since the composed system does not enable other actions that can get the probability of the restricted action *end*, the system deadlocks with probability 1 (see Fig. 2(b)).

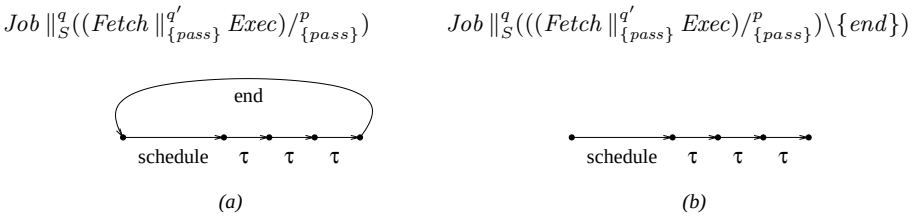

Fig. 2. Labeled transition systems associated to the models of Example 2.

The two examples above put in evidence some features of our probabilistic calculus.

As far as the CSP-like communication policy is concerned, in any binary synchronization at most one generative action can be involved and, in such a case, the result is a generative action of the same type. Instead, in case two reactive actions of type a synchronize, then the result is again a reactive action of type a. We recall that the actions belonging to the communication interface are constrained to synchronize, while all the other actions are locally and independently executed by the processes that compose the system.

As far as the probabilistic model is concerned, the following comments are in order. Probabilistic choices among generative actions (among reactive actions of the same type) are fully probabilistic, while in each other case the choice is completely nondeterministic. This is essentially due to the fact that the reactive actions are underspecified since they are guided by the environment behavior. As a consequence, the parameters that probabilistically guide the choices come into play if and only if a probabilistic choice is really to be performed. Some further details are in order in case of the parallel operator $P \parallel_S^p Q$:

- since the execution of some generative actions of P can be prevented in $P \parallel_S^p Q$ $(P \backslash L)$, the probabilities of executing the remaining generative actions of P are proportionally redistributed (similarly for Q), as shown both in Example 1 and in Example 2 (note that this is a standard approach when restricting actions in the generative model [GSS95]);
- in case of synchronizing generative actions a of P, their probability is distributed among the multiple actions a obtained by synchronizing with reactive actions a_* executable by Q, according to the probability the actions a_* are chosen in Q (symmetrically for Q), as shown in Example 1;
- in case both P and Q can execute some synchronizing actions $a_* \in S$, then $P \parallel_S^p Q$ can execute some actions a_*: the probability of each action a_* executable by $P \parallel_S^p Q$ is the product of the probabilities of the two actions a_* (one of P and one of Q) that are involved in the synchronization.

We point out that in each system state of a process term, the sum of the probabilities of the generative actions (reactive actions of a given type a), if there are any, is always equal to 1.

A final remark is in order for the hiding operator $P/_a^p$, which turns reactive and generative actions of type a into τ actions. Parameter p expresses the probability that actions τ obtained by hiding actions a_* of P are executed with respect to the generative actions previously enabled by P. Hence, p guarantees that the hiding operator does not introduce nondeterminism among generative actions. Instead, parameter p is not used when hiding generative actions (like in Example 2), since the choice among generative actions is already probabilistic. Here, we do not detail the semantics of such an operator when hiding reactive actions, since this particular case does not arise in our case study.

In the rest of the paper we use the following abbreviations. We assume parameter p to be equal to $\frac{1}{2}$ whenever it is omitted from any probabilistic operator.

Moreover, when it is clear from the context, we use the abbreviation P/S, with $S = \{a_1, \ldots, a_n\} \subseteq AType$, to denote the expression $P/_{a_1 \ldots a_n} = P/_{a_1} \ldots /_{a_n}$.

3.2 Security Analysis

Unauthorized disclosure of information in multi-level security systems can be revealed by verifying whether the several components of the system fail to protect a confidential, high-level information by leaking it to a public, unclassified, low-level user. Nondeterministic approaches to the information flow theory analyze such a kind of interference by studying the effect of the confidential activities on the public view of the system behavior (see, e.g., [McL90,FG95]). Anyway, two main problems arise when considering an approach based on pure nondeterminism. On the one hand, such a binary, qualitative notion of information leakage turns out to be too restrictive in several real systems, where high level interferes with low level all the time [RMMG01] and the effort of the designer consists of minimizing such a kind of undesirable interactions. On the other hand, the analysis of nondeterministic security properties is not appropriate to reveal those interferences that are not solely nondeterministic, since they may depend on additional information, like probabilities and time [Gra92]. Along this line, in [ABG02] we have proposed a probabilistic extension to the nondeterministic information flow theory of [FG95] based on the probabilistic process algebra surveyed above, which is intended to:

- capture those information flows which are not observable in a purely nondeterministic setting;
- deal with the general case where the tolerance for information leakage is given by a quantitative estimate expressed by probabilistic measures.

The analysis of a given security property, say SP, in a process algebraic setting roughly consists of:

1. deriving two models from the algebraic specification of the system at hand;
2. checking the semantic equivalence between such derived models.

The definition of the submodels to be compared depends on the definition of SP. One of the most intuitive properties described in [ABG02] is the Probabilistic Bisimulation Nondeducibility on Composition property ($PBNDC$), which informally says that the probabilistic low-level view of a system P in isolation is not to be altered when considering the potential interactions of P with the high-level activities offered by the external environment. The definition of $PBNDC$ is formalized as follows.

Definition 1. *$P \in PBNDC$ if and only if*

$$P \backslash AType_H \approx_{PB} ((P \parallel^{p}_{\{h_1,\ldots,h_n\}} \Pi)/^{q_1}_{h_1} \ldots /^{q_n}_{h_n}) \backslash AType_H$$

$\forall\ p, q_1, \ldots, q_n \in]0,1[$, $\forall \{h_1, \ldots, h_n\} \subseteq AType_H$, and $\forall$ high-level process Π.

Term $P\backslash AType_H$, where the high-level actions are prevented, models the low-level behavior of the system P in isolation, i.e. without high-level interactions with the environment. Term $((P \parallel^p_{\{h_1,\ldots,h_n\}} \Pi)/^{q_1}_{h_1} \ldots /^{q_n}_{h_n})\backslash AType_H$ models, from the low-level standpoint, the behavior of P when interacting, through the communication interface $\{h_1, \ldots, h_n\}$, with the high-level activities offered by the external environment, represented by any process Π (enabling high-level actions only) put in parallel with P. Finally, $\approx_{PB}$ is the equivalence relation, called weak probabilistic bisimulation [ABG02], which is a probabilistic version (inspired by [BH97]) of the classical weak bisimulation of [Mil89]. If the two views of the system are indistinguishable from the standpoint of an external observer that can access the low-level part only, then no unwanted information leakage occurs and the system is considered to be secure.

3.3 Performance Analysis

As far as the temporal aspect is concerned, we now describe an interpretation of the probabilistic process algebra in the context of discrete time [Bra02,BA03], i.e. where time is represented by a sequence of discrete steps, like in Discrete Time Markov Chains (DTMCs), and the duration of each step is given by a fixed time unit. In such a framework, the parallel composition operator we adopt allows (i) processes with different *probabilistic advancing speeds* (mean number of actions executed per time unit) to be modeled, and (ii) several processes based on different time units to be composed in parallel by preserving their temporal behavior.

In our discrete time setting, $P \parallel^p_S Q$ models a system where p $(1-p)$ is the probabilistic advancing speed of P (Q), i.e., at each system state a *probabilistic scheduler* schedules for execution an action of P with probability p and an action of Q with probability $1-p$. Now, on the basis of the mean action frequency or, in other words, of the time unit adopted by each process in isolation, we can adequately calculate a global time unit for the composed model and a suitable probabilistic parameter for the parallel operator in such a way that each process preserves its mean action frequency. More precisely, $P \parallel^p_S Q$ can be interpreted as being a description of the actual concurrent execution of two processes P and Q specified with respect to different action durations. This is done as follows. If f_P is the mean action frequency in process P (i.e. each action takes time $1/f_P$ on average to be executed) and f_Q is the mean action frequency in process Q, the mean action frequency of the parallel composition of P and Q is $f = f_P + f_Q$. Therefore, the time unit for $P \parallel^p_S Q$ is $u = 1/(f_P + f_Q)$. Now, given that p is the probabilistic advancing speed of P, the mean action frequency of P with respect to u is given by $p/u = p \cdot f$. Therefore, if we take $p = f_P/f = f_P/(f_P + f_Q)$, it follows that the mean action frequency of P within $P \parallel^p_S Q$ is f_P. Similarly, the action frequency $1-p$ of Q with respect to u is $1 - p = f_Q/f = f_Q/(f_P + f_Q)$.

Such an approach holds under the restriction that in each system state reachable from $P \parallel^p_S Q$, a probabilistic choice between P and Q guided by parameter p is to be performed. Then, from the labeled transition system associated to a fully specified process (i.e., not enabling nondeterministic choices), we can derive

a DTMC (by discarding types from transition labels), on which we can apply standard techniques to evaluate performance measures of interest. Finally, we point out that if we are interested in evaluating steady state based performance measures (which are expressible by attaching rewards to actions), the approach described above provides an exact solution even if the advancing speeds are considered to be exact instead of probabilistic [BA03].

As we will show in the next section, we employ such an approach to model the temporal behavior of each component of the audio protocol specification.

4 Performance and Security Analyses of BoAT

In a previous work [ABGR01] we conducted a simulative analysis on an algebraic specification of BoAT (based on the process algebra EMPA$_{gr}$ [Ber99]) to get some performance measures related to the QoS offered by the adaptive playout control algorithm. The reason for resorting to EMPA$_{gr}$ was its expressive power given by a set of features, such as probabilities, priorities, and value-passing. However, the results of such an analysis were limited to the functional and performance properties of BoAT. In this section, we employ the approach described in Sect. 3 in order to formally evaluate also the security level of BoAT.

4.1 The Algebraic Specification of BoAT

In this section, we introduce the algebraic specification of BoAT based on the calculus presented in Sect. 3. To this aim, we resort to the following assumptions. Since all packets are encrypted with secret keys that are not known by external parties, we abstract away from the cryptosystem used within the protocol and we just model the packet exchange. Moreover, for the sake of simplicity, we take into consideration the half-duplex part of the communication during which the so-called *sender* talks and the so-called *receiver* listens.

In Table 1 we show the model of a sender which repeatedly transmits audio packets and periodically performs the three-way handshaking protocol[1]. Process *Sen* models the situation in which a new handshaking phase is to be started: the output action *prepare_packet* expresses the transmission of an audio packet, the output action *prepare_probe* represents the transmission of the first message of the handshaking phase, and the output action *idle_S* denotes the inactivity periods of the sender during which no packets are sent out, e.g. due to a temporary overloaded channel. As far as the reactive behavior is concerned, process *Sen* is ready to accept messages coming from the receiving site. Since we just model the part of the communication during which the sender talks and the receiver listens, the only messages originated by the receiving site can be those related to the handshaking protocol, i.e. *response* messages and *ack* messages. In particular, in process *Sen* the reception of a *response* message, modeled by the

[1] We will discuss the values of the parameters associated to the operators in the next section, where we will consider the temporal behavior of the system.

Table 1. BoAT – Model of the sending site

$$
\begin{aligned}
Sen &\triangleq ((prepare\_packet.Sen +^{0.96} prepare\_probe.Sen') + \\
&\quad idle\_S.Sen) + (trans\_response_*.Sen + ignore\_ack_*.Sen) \\
Sen' &\triangleq ((prepare\_packet.Sen' +^{0.96} prepare\_probe.Sen') + \\
&\quad idle\_S.Sen') + (trans\_response_*.Sen'' + ignore\_ack_*.Sen') \\
Sen'' &\triangleq (((prepare\_packet.Sen'' +^{in} prepare\_install.Sen''') +^{0.96} \\
&\quad prepare\_probe.Sen') + idle\_S.Sen'') + \\
&\quad (trans\_response_*.Sen'' + ignore\_ack_*.Sen'') \\
Sen''' &\triangleq ((\tau.Sen''' +^{s} prepare\_packet.Sen'''') +^{0.96} \\
&\quad prepare\_probe.Sen') + (trans\_response_*.Sen''' + trans\_ack_*.Sen) \\
Sen'''' &\triangleq ((prepare\_packet.Sen'''' +^{0.96} prepare\_probe.Sen') + \\
&\quad idle\_S.Sen'''') + (trans\_response_*.Sen'''' + trans\_ack_*.Sen)
\end{aligned}
$$

input action $trans\_response_*$, refers to an old unsuccessful handshaking phase. Therefore, it is simply ignored (note that the action of type *trans_response* does not cause a change of state). Similarly, in process *Sen*, *ack* packets related to previous synchronization protocols are not expected to be received. In this case, we model the reception of an ack message via the input action of type *ignore_ack*, which denotes an *ack* packet received and discarded by the sender (i.e., related to a failed synchronization). In case of successful completion of the handshaking protocol, we will use the action type *trans_ack*. The motivation for such an explicit distinction is that we will be interested in quantifying failed and successful handshaking phases.

When a new *probe* message is transmitted, the sender waits for a *response* message from the receiver in term Sen'. Note that, in each term of the sender model, we allow a new synchronization phase to be started via the execution of the action *prepare_probe*, which leads to term Sen', since the original audio protocol discards those handshaking phases that (*i*) are not completed within a second (e.g. due to sudden spikes in end-to-end delays), and (*ii*) are deadlocked because of some lost handshaking packets. With respect to term *Sen*, in term Sen' the reception of a *response* message through the input action $trans\_response_*$ expresses the completion of the first step of the handshaking protocol after which the sender prepares an *install* packet to be sent in term Sen''.

After the execution of the action *prepare_install*, the sender waits for the final ack from the receiver in term Sen'''. Since usually the *install* packet is sent during a silence period, term Sen''' models the inactivity phase of the sender by repeatedly executing an internal action τ. Alternatively, the execution of the action *prepare_packet*, which leads to term Sen'''', represents the termination of the idle period and the beginning of a new talkspurt. For both terms Sen''' and Sen'''' the execution of the action *trans_ack* represents the final step of a three-

way handshake completed with success, whose effect is that the initial state *Sen* is reached again.

We now describe a (possible) model for the receiver and for the channel (see Table 2). As far as the network is concerned, since we concentrate on the half-duplex audio communication between the sender and the receiver, we explicitly model the channel that transmits packets from the sending site to the receiving site. Instead, the packets originated by the receiver and destined to the sender are directly passed between these two terms, so that we abstract away from the related channel.

Term *Ch* is a fully reactive process modeling a perfect channel that does not lose packets and is always ready to accept packets originated by the sender. The action $idle\_R_*$ is enabled to inform the receiver that currently no packet is ready to be delivered. Once a packet is transmitted from the sender to the channel, term *Ch* passes the control to one of terms *For_packet*, *For_probe*, or *For_install*, which are in charge of forwarding the packet to the receiver. In each of these terms, the channel is not ready to accept further packets from the sender, so that the action $idle\_S_*$ is enabled to inform the sender of such a situation.

Term *Rec* models a receiver that either is idle (and repeatedly executes action *idle_R*) or accepts any arriving packet. If a handshaking packet is delivered, term *Rec* passes the control to one of the following terms: term Rec' transmits a *response* message in case a *probe* packet has been received; term Rec'' transmits an *ack* message in case an *install* packet has been received.

Finally, in Table 2 we also report the overall system $BoAT$, expressing the parallel execution of the three models specified so far, together with the related communication interfaces.

4.2 Security Analysis of BoAT

From a security standpoint, we want to verify if the handshaking protocol (which is the core of the securing mechanism of BoAT) is robust against external attacks. To this aim, in order to apply the methodology described in Sect. 3 we have to single out the high-level actions and the low-level ones, so that the high and low behaviors of the system can be specified.

According to an approach proposed in [FGM00] for the analysis of noninterference properties of cryptographic protocols, the high level expresses the external, possibly dishonest environment, where the intruders act in order to interfere with the activities of the protocol. Hence, it is reasonable to assume that all actions that are used to model the protocol, which interacts with the environment, are high-level actions. Instead, the low-level actions are extra observable actions that we include into the protocol specification in order to observe the properties of the protocol itself.

In the context of our case study, we have to add some low-level actions that allow an external low-level observer to analyze the behavior of the three-way handshake, that is the core of the protocol under analysis. We point out that the successful execution of a handshaking phase starts with the transmission of a *probe* message and terminates with the reception of an *ack* message. Therefore,

Table 2. BoAT – Model of a perfect channel and of the receiving site

$$
\begin{array}{rcl}
Ch & \triangleq & prepare\_packet_*.For\_packet + \\
 & & prepare\_probe_*.For\_probe + \\
 & & prepare\_install_*.For\_install + \\
 & & idle\_R_*.Ch \\
For\_packet & \triangleq & trans\_packet_*.Ch + idle\_S_*.For\_packet \\
For\_probe & \triangleq & trans\_probe_*.Ch + idle\_S_*.For\_probe \\
For\_install & \triangleq & trans\_install_*.Ch + idle\_S_*.For\_install \\
 & & \\
Rec & \triangleq & (trans\_probe.Rec' + trans\_install.Rec'') + \\
 & & (trans\_packet.Rec + idle\_R.Rec) \\
Rec' & \triangleq & trans\_response.Rec \\
Rec'' & \triangleq & trans\_ack.Rec + ignore\_ack.Rec \\
 & & \\
BoAT & \triangleq & Sen \parallel_S^p (Ch \parallel_R Rec) \\
S & \triangleq & \{prepare\_packet,\ prepare\_probe,\ prepare\_install, \\
 & & \ trans\_response,\ trans\_ack,\ ignore\_ack,\ idle\_S\} \\
R & \triangleq & \{trans\_packet,\ trans\_probe,\ trans\_install,\ idle\_R\}
\end{array}
$$

if we include in the sender model a low-level action **init_synch** immediately after the execution of each action of type *prepare_probe*, and a low-level action **end_synch** immediately after the execution of each action of type *trans_ack*, a low-level observer may infer the result of any handshaking phase and potentially realize that an adversary is trying to interfere with the synchronization policy. As an example, term *Sen* of Table 1 should be changed as follows (similarly for the other ones):

$$
\begin{array}{rcl}
Sen & \triangleq & ((prepare\_packet.Sen +^{0.96} prepare\_probe.\mathbf{init\_synch}.Sen') + \\
 & & idle\_S.Sen) + (trans\_response_*.Sen + ignore\_ack_*.Sen).
\end{array}
$$

In the following, we denote by Sen_l the sender model enriched with the low-level actions as specified above. Now, we are ready to apply the methodology described in Sect. 3 for the security analysis of BoAT. Since we are assuming that both participants have been previously authenticated, potential interferences may come from the channel only. Therefore, the security property we intend to check can be informally defined as follows.

BoAT is secure if and only if the execution of the protocol without external interferences is invariant with respect to the execution of the protocol in an untrusted channel possibly under the control of the adversary.

On the one hand, we observe that term *BoAT* of Table 2 expresses the execution of the protocol without interferences. Indeed, we recall that term *Ch* of

Table 2 models a perfect, private channel with no intruders. Therefore, it follows that the low-level view of the system in the sense specified above is expressed by term $BoAT_l \stackrel{\Delta}{=} (Sen_l \parallel_S^p (Ch \parallel_R Rec))/(S \cup R)$, where all the high-level actions denoting the protocol activities are hidden.

On the other hand, if we assume that the network is under the control of the adversary, then we have to consider the system for any model of the communication channel, which may include external attacks. Therefore, we can formalize the security property in a *PBNDC* style as follows[2]:

$$BoAT_l \backslash AType_H \approx_{PB} ((Sen_l \parallel_S^q (Ch' \parallel_R^{q'} Rec))/_{\mathbf{s}\cdot\mathbf{r}}^{\mathbf{p}}) \backslash AType_H$$

$\forall q, q' \in]0,1[$, $\forall \mathbf{p} \in Seq_{]0,1[}^{|S \cup R|}$ and $\forall$ high-level process Ch'.

The formula above says that the low-level view of $BoAT_l$ in isolation is to be the same as that observed when the sender and the receiver perform their protocol steps by transmitting their packets over an untrusted (potentially controlled by the adversary) channel (modeled by any high-level term Ch', which may include dishonest strategies). Note that in $BoAT_l \backslash AType_H$ the final restriction on the set $AType_H$ is redundant, since in term $BoAT_l$ the high-level actions are either hidden (if they result from a synchronization) or restricted by the parallel operators (if they cannot synchronize).

The low view of term $BoAT_l$ consists of a sequence of actions **end_synch**, each one preceded by at least one action **init_synch**. This correctly represents the expected behavior of BoAT, which periodically starts a new handshaking phase (denoted by action **init_synch**) and each of these phases may be completed with success (denoted by action **end_synch**).

If we take into consideration only the possible behaviors of the system[3], then the low behavior of BoAT is not altered if we consider intruders that interfere by capturing some of the transmitted packets (replace, e.g., term Ch' by term *Ch_lossy* of Table 3). In fact, even if an adversary blocks some handshaking packets, the observable low-level view is given by a sequence of actions **end_synch**, each one preceded by at least one action **init_synch**. On the other hand, it is easy to see that a denial-of-service attack conducted by an adversary (which eavesdrops the channel and discards each transmitted packet – consider, e.g., term *Ch_blind* of Table 3) is the only kind of attack that is responsible for altering the expected low view of the system. In fact, in such a case, the low-level action **end_synch** is never enabled. However, if we take into consideration the probabilistic information, we observe that the probability of observing the successful handshakes (with respect to those that fail) depends on the probabilistic behavior of the intruder. For instance, if the considered channel is term *Ch_lossy* of Table 3, then the probability of observing the low-level action **end_synch**

[2] Given $S \subseteq AType$, $\mathbf{s}$ denotes the sequence, in alphabetic order, of types contained in S, while $\mathbf{s} \cdot \mathbf{s}'$ denotes the catenation of the two sequences $\mathbf{s}$ and $\mathbf{s}'$; we also denote by Seq_D^k the set of k-length sequences with domain D.

[3] We can apply the noninterference theory for nondeterministic processes if we ignore the probabilistic information reported in the algebraic specification of BoAT.

Table 3. Some models of possible channels

$$
\begin{array}{rcl}
Ch\_lossy & \triangleq & (prepare\_packet_*.Ch\_lossy +^{d_a} prepare\_packet_*.For\_packet) + \\
 & & ((prepare\_probe_*.Ch\_lossy +^{d_p} prepare\_probe_*.For\_probe) + \\
 & & (prepare\_install_*.Ch\_lossy +^{d_i} prepare\_install_*.For\_install)) + \\
 & & idle\_R_*.Ch\_lossy \\
For\_packet & \triangleq & trans\_packet_*.Ch\_lossy + idle\_S_*.For\_packet \\
For\_probe & \triangleq & trans\_probe_*.Ch\_lossy + idle\_S_*.For\_probe \\
For\_install & \triangleq & trans\_install_*.Ch\_lossy + idle\_S_*.For\_install \\
 & & \\
Ch\_blind & \triangleq & ((prepare\_packet_*.Ch\_blind + prepare\_probe_*.Ch\_blind) + \\
 & & (prepare\_install_*.Ch\_blind + idle\_R_*.Ch\_blind))
\end{array}
$$

depends on parameters d_a, d_p, and d_i, which probabilistically model the loss percentage of audio packets, probe packets, and install packets, respectively.

Based on these considerations, we conclude that $BoAT$ does not satisfy the security property. Now, we are interested in estimating how the security level of BoAT is affected by the adversary strategy. More precisely, we want to evaluate if the honest participants are able to detect any external attack and how to behave in such a case. To this aim, we pass to the performance model of BoAT in order to derive performance measures of interest.

4.3 Mixed Performance/Security Analysis of BoAT

Before introducing the performance model of BoAT, we specify the kind of attack whose effects we are interested in evaluating. Since we concentrate on the half-duplex part of the audio communication from the sender to the receiver, we take into consideration all possible attacks performed by an adversary that eavesdrops (and possibly captures) the packets generated by the sending site. Moreover, we consider the preliminary authentication phase preceding the audio conversation as a secure step. Therefore, since all packets are enriched with timestamps and encrypted with secret keys, we consider forgery, authentication and replication attacks by any external party as not meaningful. Instead, an adversary can try to conduct a cryptanalysis attack in order to compromize the privacy of the conversation. Since the secrecy level of BoAT trusts on the short duration of each session key (about a second of conversation, i.e. the time between two consecutive handshaking phases), and the robustness of the stream cipher used by BoAT may depend on the quantity of data encrypted with the same key (see, e.g., [BSW00]), then the probability of cracking a session key increases if several consecutive handshaking protocols fail, because in such a case the same key is used to encrypt several seconds of conversation. With this in view, a strategy for a dishonest adversary consists of intercepting and blocking the packets of

the handshaking protocol, in order to weaken the secrecy condition of BoAT by extending the lifetime of each session key.

In the previous section, we have described a model of the channel (see the fully reactive term *Ch_lossy* of Table 3) that expresses such a kind of attack. More precisely, it is easy to verify that, from the viewpoint of a low-level external observer, the probabilistic behavior of term *Ch_lossy* (expressed by parameters d_a, d_p, and d_i only) within the overall system is responsible for affecting the probability distribution of the successful handshakes. In this section, we quantify the difference between the behavior of the system without intruders and the behavior of the system under the attack specified by term *Ch_lossy*. To this end, we first pass to a performance model by considering the temporal behavior of processes *Sen* and *Rec*, and then we analyze the composed system $((Sen \parallel^p_S (Ch\_lossy \parallel_R Rec))/(S \cup R))\backslash AType_H$, by varying the probabilistic behavior of term *Ch_lossy*. Note that the two limiting scenarios are represented by (*i*) the channel that blocks all the transmitted packets (see term *Ch_blind* of Table 3), and (*ii*) the channel that forwards each transmitted packet (see term *Ch* of Table 2). The goal of our analysis is the evaluation of the throughput of the handshaking protocol (i.e. the mean number of handshakes completed with success per time unit) for each adversary strategy between the two limiting scenarios. In the following, we show how to model the temporal behavior of each component according to the features of BoAT specified in [ABGR01,RGPSB01].

As far as the sender model is concerned, we consider a sending site that produces 25 audio packets per sec. To this end, the time unit we adopt for process *Sen* is 40 ms, i.e. each action of term *Sen* takes 40 ms on average to be executed or, equivalently, the mean action frequency of term *Sen* is 25 actions per sec. Moreover, we assume that the time between two consecutive handshaking phases is about 1 sec. The probabilistic parameters shown in Table 1 exactly reflect such temporal assumptions. In particular, since action *prepare_probe* has to be executed once a second on average and the mean action frequency for process *Sen* is 25 actions per sec, in each discrete step the action to be executed is *prepare_probe* with probability $1/25 = 0.04$ and *prepare_packet* with probability $1 - 0.04 = 0.96$. If such actions are enabled in the composed system, then the action *idle_S* is not; on the contrary, if such actions are not enabled, then the action *idle_S* is performed with probability 1.

In term Sen'', the probability 0.96 of sending a packet different from a *probe* message has to be distributed between the two possible events: the transmission of either an audio packet or an *install* message. Since the *install* message is not sent as soon as the *response* message is received, but only during a silence period, we employ parameter *in* to express the probability of executing the action *prepare_packet* with respect to the action *prepare_install*. In practice, as parameter *in* increases, the probability of being in a talkspurt increases as well. Since experimental studies show that the duration of a silence period is about the 30% of the duration of a talkspurt [HSK98,ABGR01], in the following we assume $in = 0.7$, meaning that the probability of transmitting the *install* message (instead of an audio packet) is equal to 30%.

In term Sen''', parameter s expresses the probability of being in a silence period between two consecutive talkspurts. More precisely, the duration of the inactivity phase of the sender is probabilistically modeled according to a Bernoulli distribution with parameter s: the sender either stays in its idle period with probability $s \cdot 0.96$ or starts a new talkspurt with probability $(1-s) \cdot 0.96$. If the sender is not allowed to send packets (i.e. actions *prepare_packet* and *prepare_probe* are not enabled), the action τ expressing the idling period is executed with probability 1. As far as the experimental scenario is concerned, we assume $s = 0.7$, namely in term Sen''' we start a new talkspurt (instead of staying idle) with probability 30%.

As far as the receiving site is concerned, the time unit adopted for term *Rec* is 1 ms, i.e. each action of term *Rec* takes 1 ms on average to be executed or, equivalently, the mean action frequency of term *Rec* is 1000 actions per sec. This choice expresses the fact that the receiving site is always ready to accept packets and, if necessary, to immediately send back a handshaking packet.

As far as the overall system is concerned, we now compute the global time unit u and the parameters of the probabilistic parallel operators for the composed system $((Sen \parallel_S^p (Ch\_lossy \parallel_R Rec))/(S \cup R)) \setminus AType_H$. The global time unit u is the inverse of the global action frequency of the composed system, which in turn is equal to 25 (i.e., the action frequency of term *Sen*) + 1000 (see term *Rec*) = 1025 actions per sec. Note that the process modeling the channel is completely reactive (that means we abstract away from the transmission delays experienced along the channel), so that it does not express a process with its own advancing speed. Moreover, we also have that in each system state both processes *Sen* and *Rec* can execute at least a generative action. Hence, parameter p represents the advancing speed of term *Sen* within the overall system and is given by the ratio of the action frequency of term *Sen* over the global action frequency of the composed system, i.e. $p = 25/1025 \approx 0.02439$.

From the algebraic specification of the composed system we can derive a DTMC, since the related labeled transition system is fully specified in the sense that all the choices are fully probabilistic or, in other words, reactive actions are never enabled. Therefore, in order to obtain steady state based performance measures, we adequately attach rewards to actions and we analyze the related DTMC. To this end, we resorted to the software tool TwoTowers [BCSS98], which has been extended to support the generative-reactive approach of our probabilistic calculus. Such a tool also implements the algebraic reward based method needed to specify and derive performance measures. On the one hand, we evaluate the throughput of the handshaking protocol at the sending site, i.e. the number of handshaking phases completed with success, expressed in terms of occurrences of actions of type *trans_ack*. This is done by attaching a reward equal to 1 to the action *trans_ack* and a reward equal to 0 to each other action. On the other hand, we also evaluate the throughput for the receiver, i.e. the number of audio packets arrived at the receiving site, in terms of occurrences of actions of type *trans_packet*, by attaching a reward equal to 1 to the above action and a reward equal to 0 to each other action.

The performance results we are going to present have been obtained by varying the probabilistic behavior of the channel model. As we have shown, an adversary that is interested in extending the lifetime of each session key tries to intercept and discard the handshaking packets. Given that all packets are encrypted, an adversary cannot distinguish the audio packets from the handshaking packets originated by the sender. Hence, in a first scenario we assume that an intruder can just try to randomly discard some of the transmitted packets. In particular, we vary from 0% to 100% the percentage of packets captured and blocked along the channel by the dishonest intruder. The extreme value 0% corresponds to the behavior modeled by term *Ch* of Table 2, while the extreme value 100% corresponds to the behavior modeled by term *Ch_blind* of Table 3. For each other value, we consider term *Ch_lossy* and, by assuming $d_a = d_p = d_i$ (expressing the probability of discarding audio, probe, and install packets, respectively), we vary such parameters from 0 to 1.

In Fig. 3 we show the tradeoff between the percentage of packets discarded by an adversary that eavesdrops the channel and the throughput at the sending site in terms of occurrences of actions of type *trans_ack*, which expresses the completion with success of the handshaking phase. As an expected result, the number of synchronizations completed in a second decreases as the percentage of lost packets tends to 1. In particular, in case of a perfect and private channel, the audio protocol completes about 0.866 handshaking phases per sec. For a loss rate less than 10% such a throughput is still tolerable (greater than 0.7 phases completed per sec), but as the loss rate increases (20% and more) the throughput rapidly converges under 0.5 phases completed per sec, i.e. less than one synchronization every 2 sec. In particular, in Fig. 3 we also show the tolerable area beyond which the number of successful synchronizations is so low that (*i*) the estimated playout delay cannot represent an accurate evaluation of the current state of the channel [HSK98,RGPSB01], and (*ii*) the lifetime of each session key is noticeably greater than 1 sec. The main result that we derive from such an experiment is that the probabilistic behavior of the environment (adversary) affects the throughput of the three-way handshake protocol as measured by the sender. Similarly, we can infer the effect of the adversary behavior on the receiving site. In Fig. 4 we show the tradeoff between the percentage of packets discarded by an adversary that eavesdrops the channel and the throughput at the receiving site in terms of occurrences of actions of type *trans_packet*. Once again, the number of packets received in a second decreases as the percentage of lost packets tends to 1. In Fig. 4 we also report the area denoting the performance that is desired by the authenticated parties, in the sense that beyond such an area (i.e., for loss rates greater than 10%) the quality of the perceived audio dramatically jeopardizes the comprehension of the conversation [RGPSB01,ABGR01].

The analysis conducted above clearly shows that each change in the probabilistic behavior of the (hostile) environment is reflected upon the performance behavior of BoAT as measured by the honest participants. Such an information can be exploited in order to quantify the risk for the encrypted data to be cracked by a dishonest third party. By comparing Fig. 3 and Fig. 4 we observe

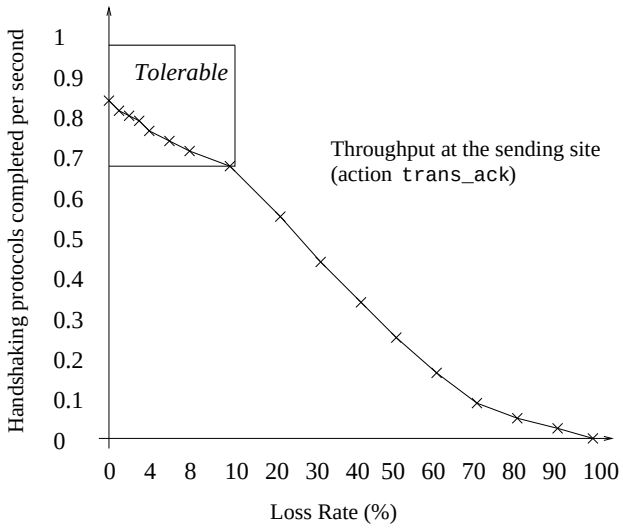

Fig. 3. Handshaking protocol throughput

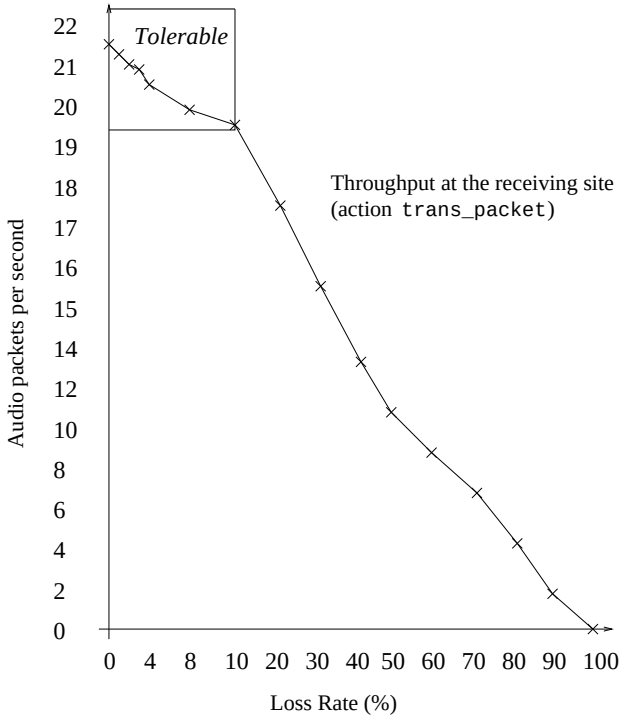

Fig. 4. Audio packet throughput at the receiving site

that an intruder that captures packets in a random way affects the performance of both audio packet throughput at the receiving site and handshaking throughput at the sending site. If the behavior of the intruder makes both throughputs come out from the tolerable area, the honest participants decide to cut off the communication due to the scarce QoS. If this is not the case, the throughputs maintain high values and, as a consequence, both audio quality and data secrecy are not compromized.

A more interesting result can be obtained by dealing with a clever adversary that somehow is able to intercept the handshaking packets transmitted by the sender. If this is the case, the intruder attack can make the throughput of the handshaking protocol decreased, without substantially altering the audio packet throughput at the receiving site. For instance, an adversary may try to exploit the fact that the *install* packet is sent during a silence period in order to intercept and block such a handshaking packet. However, this attack can be easily avoided by a sender that generates and transmits encrypted dummy packets during the silence phases between consecutive talkspurts, so that the adversary cannot detect the *install* packet. Alternatively, an intruder may try to guess the instant in which the *probe* packet is transmitted by the sender. Such a strategy can cause serious damage to the security level of the audio protocol if exactly 1 sec passes between the transmission of two consecutive *probe* messages, as in the original proposal of BoAT. Indeed, by assuming such a behavior of the sending site, the intruder can get a good approximation of the transmission time of the *probe* messages by eavesdropping the conversation from the beginning.

With this in view, here we evaluate the trade-off between the throughput of the synchronization protocol and the capability of the intruder of guessing the *probe* packets. To this end, we make the following assumptions. Supposed that the ith handshaking phase starts approximately i sec after the beginning of the conversation, we assume the instant of the transmission of the related *probe* message to follow a gaussian distribution with mean value i and standard deviation *dev*. By varying parameter *dev* and by fixing the width of the temporal interval around time i within which the intruder discards all the transmitted packets, we employ the normal distribution tables [Bey90] to measure the probability for the intruder of stopping exactly the *probe* message (such a probability is assigned to parameter d_p of term *Ch_lossy* of the algebraic specification). In a first scenario, we assume that the intruder discards 5 packets transmitted around time i, by covering a time interval of 160 ms. If one of them is the *probe* packet, then the percentage of lost audio packets (modeled by parameter d_a) is 16%. In a second scenario, we assume that the intruder discards 3 packets only, by covering a time interval of 80 ms. This corresponds to $d_a = 8\%$. Finally, parameter d_i, which models the percentage of lost *install* packets, is set to 0, because the transmission instant of such a packet is out of the time interval within which the adversary captures packets.

In Fig. 5, we show the results that derive from the analysis of the BoAT specification where the term modeling the channel is changed according to the behavior above. We point out that the throughput of the handshaking protocol

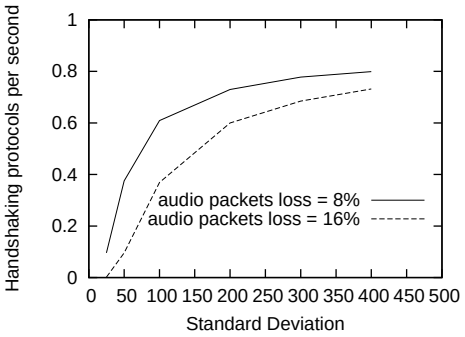

Fig. 5. Trade-off between handshaking protocol throughput and intruder strategy

is computed exactly as explained in case of Fig. 3. The curves are obtained by varying the probability for the intruder of guessing the *probe* message according to different values of parameter *dev*, which varies from 25 to 400 ms. A low value of the standard deviation *dev* means that the approximation made by the intruder is accurate with high probability, because the ith *probe* packet is indeed transmitted around time i, while a high value of *dev* means that the intruder has a lower chance of guessing the *probe* message. Hence, by increasing parameter *dev* the throughput of the handshaking protocol tends to its limiting value 0.866, which represents the case in which no packets are stopped by the intruder. In the first scenario, i.e. when the intruder discards 5 packets per sec, we observe that for a standard deviation *dev* less than 100 ms, the throughput of the handshaking protocol rapidly converges to values corresponding to a lifetime of each session key noticeably greater than 5 sec (instead of 1 sec as expected by the protocol), which in many cases is more than enough to crack the session key and the encrypted data (see, e.g., [BSW00]). Anyway, in such a scenario we also have that the audio packet loss measured at the receiving site is about 16%, which represents a limiting performance typical of highly overloaded channels. Therefore, the audio communication will be likely terminated by the participants because of such a poor QoS. Instead, in the second scenario, i.e. when the intruder discards 3 packets per sec, the audio packet loss measured at the receiving site is about 8%, which is a performance acceptable by the honest participants. In spite of this, a clever intruder can reduce the throughput of the handshaking protocol up to about 0.096 and, even in case of parameter *dev* equal to 50 ms, the lifetime of each session key is about three times the expected value.

The unwanted behavior described above can be avoided by implementing a version of BoAT proposed in [AGR01], which suggests to vary, during the conversation lifetime, the time interval between two consecutive handshaking phases. More precisely, in order to make difficult for the intruder a precise evaluation of the time instant in which the *probe* message is sent, we introduce a random factor in the computation of such a time instant. To this end, we can employ the results of the formal analysis conducted above in order to evaluate the relation-

ship between the choice of such a random factor and the number of expected successful handshaking phases. In particular, instead of sending the ith *probe* message exactly i sec after the beginning of the conversation, we could decide to send such a packet at a time instant sampled according to a gaussian distribution with mean value i and standard deviation *dev*. The choice of parameter *dev* affects the handshaking protocol throughput and, as a consequence, the lifetime of each session key and the secrecy level of the audio protocol. By following the results depicted in Fig. 5, it is easy to see that a value of parameter *dev* greater than 200 ms is more than enough to guarantee a throughput of the handshaking protocol that falls in the tolerable area put in evidence in Fig. 3, independently of the behavior of any clever intruder.

The mixed security/performance analysis revealed that an attack that aims at weakening the session keys can be easily prevented by changing the algorithm followed by the sending site to originate the *probe* messages. We conclude by observing that the effectiveness of such an attack and the related countermove cannot be viewed if we just employ a nondeterministic approach to the information flow theory. This is because in a nondeterministic setting we can reveal a denial-of-service attack only, while we have seen that the analysis of the performance behavior of BoAT is needed to give a quantitative estimate of the capability of a probabilistic adversary of compromizing the secrecy level of the audio communication.

5 Conclusion

We conclude by summarizing the two main results presented in this paper. On the one hand, we have emphasized that a nondeterministic approach is not enough to analyze the security level of real systems for which a quantitative estimate of the unwanted information flows is more significant. On the other hand, we have seen that performance behavior and security level can be tightly connected. To formally evaluate such a relation, an approach that allows both aspects to be described and analyzed on the same model is needed.

Finally, it is worth noting that in this paper we have considered a secrecy property whose analysis should help the modeler to reveal unwanted conditions in which cryptanalysis attacks can be successfully completed. We did not explicitly modeled the weaknesses of the keys by evaluating, e.g., the probability of guessing a message encrypted through a session key with a certain lifetime. To do this, we intend to extend our process algebraic approach in order to deal with cryptographic operations and imperfect cryptography.

Acknowledgement

We are grateful to Marco Roccetti, developer of BoAT, for his suggestions and comments. This work has been partially funded by Progetto MEFISTO (Metodi Formali per la Sicurezza e il Tempo) and supported by the DEGAS (Design Environments for Global Applications) project funded by the IST Programme, FET Proactive Initiative on Global Computing.

References

[AMR03] A. Aldini, A. Amoroso, M. Roccetti. A Secure Protocol for Voice-Operated E-Commerce Systems over IP Networks. *Int. Journal of Pure and Applied Mathematics* 4(2):121-142, Academic Publications, 2003.

[ABGR01] A. Aldini, M. Bernardo, R. Gorrieri, M. Roccetti. Comparing the QoS of Internet Audio Mechanisms via Formal Methods. *ACM Transactions on Modelling and Computer Simulation* 11(1):1-42, ACM Press, 2001.

[ABG02] A. Aldini, M. Bravetti, R. Gorrieri. A Process-algebraic Approach for the Analysis of Probabilistic Non-interference. *Journal of Computer Security*, to appear.

[AGR01] A. Aldini, R. Gorrieri, M. Roccetti. An Adaptive Mechanism for Real-time Secure Speech Transmission over the Internet. In *2nd IP-Telephony Workshop (IP-Tel'01), H. Schulzrinne ed.*, pp. 64-72, 2001.

[AGR03] A. Aldini, R. Gorrieri, M. Roccetti. On Securing Real Time Speech Transmission over the Internet: An Experimental Study. *EURASIP Journal on Applied Signal Processing*, Special Issue on Digital Audio for Multimedia Communications, Hindawi Publishing Corporation, to appear.

[BH97] C. Baier, H. Hermanns. Weak Bisimulation for Fully Probabilistic Processes. In *9th Int. Conf. on Computer Aided Verification (CAV'97), LNCS* 1254:119-130, Springer, 1997.

[Ber99] M. Bernardo. Theory and Application of Extended Markovian Process Algebra. *Ph.D. Thesis*, University of Bologna, Italy, 1999. `ftp://ftp.cs.unibo.it/pub/techreports/`

[BCSS98] M. Bernardo, W.R. Cleaveland, S.T. Sims, W.J. Stewart. TwoTowers: A Tool Integrating Functional and Performance Analysis of Concurrent Systems. In *Joint Int. Conf. on Formal Description Techniques for Distributed Systems and Communication Protocols and Protocol Specification, Testing, and Verification (FORTE-PSTV'98)*, pp. 457-467, Kluwer, 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. *Information and Computation* 144:83-154, 1998.

[Bey90] W.H. Beyer. Standard Probability & Statistics Tables & Formulae. Boca Raton, FL: CRC Press, 1990.

[BSW00] A. Biryukov, A. Shamir, D. Wagner. Real Time Cryptanalysis of A5/1 on a PC. In *Fast Software Encryption Workshop*, LNCS 1978, Springer, 2000.

[Bra02] M. Bravetti. Specification and Analysis of Stochastic Real-Time Systems. *Ph.D. Thesis*, University of Bologna (Italy), 2002. `ftp://ftp.cs.unibo.it/pub/techreports/`

[BA03] M. Bravetti, A. Aldini. Discrete Time Generative-reactive Probabilistic Processes with Different Advancing Speeds. *Theoretical Computer Science* 290(1):355-406, 2003.

[FG95] R. Focardi, R. Gorrieri. A Classification of Security Properties. *Journal of Computer Security* 3(1):5-33, 1995.

[FGM00] R. Focardi, R. Gorrieri, F. Martinelli. Non Interference for the Analysis of Cryptographic Protocols. In *27th Int. Colloquium on Automata, Languages and Programming (ICALP'00)*, LNCS 1853:354-372, Springer, 2000.

[GSS95] R.J. van Glabbeek, S.A. Smolka, B. Steffen. Reactive, Generative and Stratified Models of Probabilistic Processes. In *Information and Computation* 121:59-80, 1995.

[GM82] J.A. Goguen, J. Meseguer. Security Policy and Security Models. In *Symposium on Security and Privacy (SSP'82)*, pp. 11-20, IEEE CS Press, 1982.

[Gra92] J. W. Gray III. Toward a Mathematical Foundation for Information Flow Security. In *Journal of Computer Security* 1:255-294, 1992.

[HSK98] V. Hardman, M.A. Sasse, I. Kouvelas. Successful Multi-Party Audio Communication over the Internet. *Communications of the ACM* 41:74-80, 1998. `http://www-mice.cs.ucl.ac.uk/multimedia/software/rat/`

[HS95] P. Harrison, B. Strulo. Stochastic Process Algebra for Discrete Event Simulation. In *Quantitative Methods in Parallel Systems, ESPRIT Basic Research Series*, pp. 18-37, Springer, 1995.

[HHHMR94] H. Hermanns, U. Herzog, J. Hillston, V. Mertsiotakis, M. Rettelbach. Stochastic Process Algebras: Integrating Qualitative and Quantitative Modelling. In *7th Conf. on Formal Description Techniques (FORTE'94)*, pp. 449-451, Chapman & Hall, 1994.

[McL90] J. McLean. Security Models and Information Flow. In *IEEE Symposium on Research in Security and Privacy*, pp. 180-189, 1990.

[Mil89] R. Milner. *Communication and Concurrency*, Prentice Hall, 1989.

[RGPSB01] M. Roccetti, V. Ghini, G. Pau, P. Salomoni, M. E. Bonfigli, Design and Experimental Evaluation of an Adaptive Playout Delay Control Mechanism for Packetized Audio for Use over the Internet. *Multimedia Tools and Appl., an Int. Journal* 14(1):23-53, Kluwer Academic Publ., 2001.

[RMMG01] P.Y.A. Ryan, J. McLean, J. Millen, V. Gligor. Non-interference: who needs it? In*14th Computer Security Foundations Workshop (CSFW'01)*, pp. 237-238, IEEE CS Press, 2001.

[Schn96] B. Schneier. *Applied Cryptography, 2nd Edition*, John Wiley & Sons, 1996.

[Schu92] H. Schulzrinne. Voice Communication across the Internet: a Network Voice Terminal. *Tech. Rep.*, University of Massachusetts, Amherst (MA), 1992. `http://www.cs.columbia.edu/~hgs/rtp/nevot.html`

[SDS01] R. Steinmetz, J. Dittman, M. Steinebach (Eds.). *Communications and Multimedia Security Issues of the New Century*, Kluwer Academic Publishers, 2001.