Multimedia Systems (1998) 6:17–28

Multimedia Systems
© Springer-Verlag 1998

Packet audio playout delay adjustment: performance bounds and algorithms

Sue B. Moon, Jim Kurose, Don Towsley

Department of Computer Science, University of Massachusetts at Amherst, Amherst, MA 01003, USA; {sbmoon,kurose,towsley}@cs.umass.edu

Abstract. In packet audio applications, packets are buffered at a receiving site and their playout delayed in order to compensate for variable network delays. In this paper, we consider the problem of adaptively adjusting the playout delay in order to keep this delay as small as possible, while at the same time avoiding excessive "loss" due to the arrival of packets at the receiver after their playout time has already passed. The contributions of this paper are twofold. First, given a trace of packet audio receptions at a receiver, we present efficient algorithms for computing a bound on the achievable performance of *any* playout delay adjustment algorithm. More precisely, we compute upper and lower bounds (which are shown to be tight for the range of loss and delay values of interest) on the optimum (minimum) average playout delay for a given number of packet losses (due to late arrivals) at the receiver for that trace. Second, we present a new adaptive delay adjustment algorithm that tracks the network delay of recently received packets and efficiently maintains delay percentile information. This information, together with a "delay spike" detection algorithm based on (but extending) our earlier work, is used to dynamically adjust talkspurt playout delay. We show that this algorithm outperforms existing delay adjustment algorithms over a number of measured audio delay traces and performs close to the theoretical optimum over a range of parameter values of interest.

Key words: Packetized audio – Playout delay – Multimedia – Packet loss – Dynamic programming – Computer networks

1 Introduction

In the 20 years that have passed since the early Arpanet experiments with packetized voice [Coh77], packetized audio has blossomed into an application that many Internet users now use regularly. For example, the audio (and video and whiteboard) segments of many technical conferences and workshops are now carried over the MBone multicast network [CD92, Jac94, MB94]. Smaller, more interactive, group meetings are also frequently conducted over the Internet using these multimedia tools.

Packet audio tools such as NeVoT [Sch92] and vat [JM] operate by periodically gathering audio samples generated at the sending host, packetizing them, and transmitting the resulting packet (via UDP unicast/multicast) to the receiving site(s). For efficiency, a source's audio is typically divided into "talkspurts" (periods of audio activity) and "silence periods" (periods of audio inactivity, during which no audio packets are generated). In order to faithfully reconstruct the audio at a receiving site, data in packets within a talkspurt must be played out in the same periodic manner in which they were generated.

If the underlying network is free of variations (jitter) in packet delays, a receiving site can simply play out an audio packet as soon as it is received. However, jitter-free, in-order, on-time packet delivery rarely, if ever, occurs in today's packet-switched networks. In order to compensate for these variable delays, a smoothing buffer is thus typically used at a receiver. Received packets are first queued into the smoothing buffer and the periodic playout of packets within a talkspurt is delayed for some amount of time beyond the reception of the first packet in the talkspurt. Informally, we refer to this delay as the *playout delay* of the talkspurt. Clearly, the longer the playout delay, the more likely it is that a packet will have arrived before its scheduled playout time. Excessively long playout delays, however, can significantly impair human conversations. There is thus a critical tradeoff between the length of playout delay and the amount of loss (due to late packet arrival) that is incurred. Generally, delays between talkspurt generation and receiver playout of less than 400 ms [ITU93] and a loss percentage of up to 5% [Jay80] are considered to be quite tolerable in human conversations. The talkspurt playout delays themselves can be either fixed for the duration of the audio session (an approach examined in [Mon83, Coh77]), or adaptively adjusted from one talkspurt to the next, with intervening silence periods artificially elongated or compressed accordingly – the approach taken in the NeVoT and vat audio tools.

In this paper, we focus on this tradeoff between packet playout delay and packet loss. The main contributions of

Correspondence to: D. Towsley

this paper are twofold. First, given a trace of packet audio receptions at a receiver, we present efficient algorithms for computing upper and lower bounds on the optimum (minimum) average playout delay for a given number of packet losses (due to late arrivals) at the receiver for that trace. These bounds, which we show to be tight for a range of loss and delay values of interest, are of particular importance as they provide a bound on the achievable performance of *any* adaptive playout delay adjustment algorithm. Our second significant contribution is the development of a new adaptive delay adjustment algorithm that tracks the network delay of recently received packets and efficiently maintains delay percentile information. This information, together with a "delay spike" detection algorithm based on (but extending) our earlier work [RKTS94], is used by the new algorithm to dynamically adjust talkspurt playout delay. We show that this new algorithm generally outperforms existing delay adjustment algorithms over a number of measured audio delay traces and performs close to the theoretical optimum over a range of parameter values of interest.

While the work reported in this paper is based on an Internet service model which only provides best effort service, such adaptive audio applications are of importance in both future Internet and ATM network architectures as well. For example, the Integrated Services Working Group of the IETF has issued Internet Drafts [Wro95, SPW95, **?**] for predictive and controlled service classes in which adaptive applications may respond to the varying networks delays. In ATM ABR service, the delays seen by an application during a connection may vary as well.

The remainder of this paper is structured as follows. Section 2 provides additional background for our work, including an extended discussion of the observed delay spikes in the packet audio traces reported earlier in [RKTS94] as well as in new, more recent experimental traces reported here. In Sect. 3, we describe the algorithms used to compute bounds on the optimum average playout delay for a given loss. In Sect. 4, we present our new adaptive playout delay adjustment algorithm and examine its performance. Section 5 concludes this paper.

2 Background

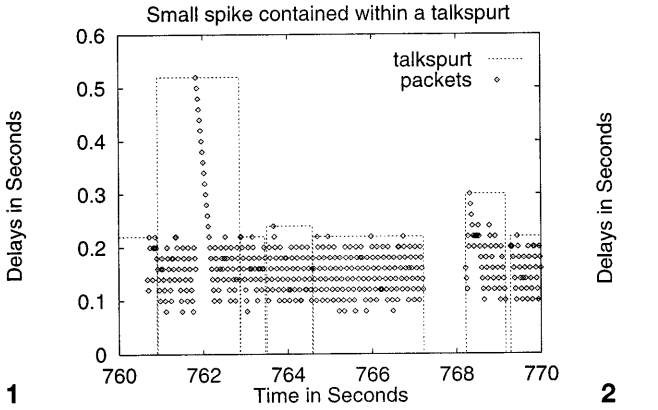
As discussed above, a receiving site in an audio application typically buffers packets and delays their playout [ACBOS93, Mon83] in order to compensate for variable network delays. The playout delay can be constant throughout the entire audio session or can be adaptively adjusted during the session from one talkspurt to the next. In the Internet, end-to-end delays fluctuate significantly [Bol93, SGAJ93] and a constant, non-adaptive, playout delay would thus likely yield unsatisfactory audio quality for interactive audio applications. There are two approaches for adaptive playout adjustment: per-talkspurt and per-packet adjustment. The former approach uses the same playout delay throughout a talkspurt (and, as a result, faithfully reconstructs the original periodic nature of the received audio data from the sender), but allows different playout delays from one talkspurt to another. While this may result in artificially elongated or compressed silence periods, this is not noticeable in played-out speech if the change is reasonably small [Mon83]. In the latter approach, the playout delay varies from packet to packet. A per-packet adaptive adjustment introduces gaps inside talkspurts and is cited as being damaging to the audio quality [ACBOS93, Coh77].

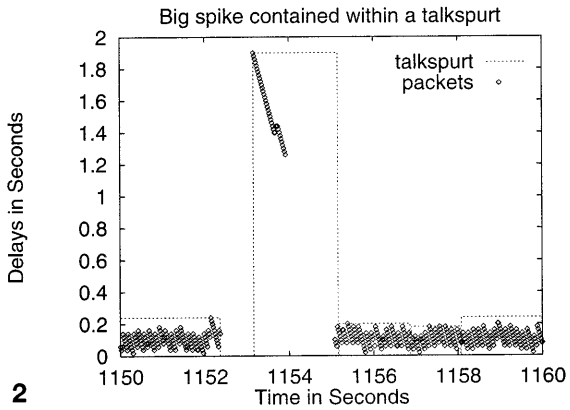
Because it is the variable nature of network delays that gives rise to the need for playout delay adjustment algorithms, an understanding of network delays and their effects on packet audio at both the individual packet and talkspurt level is important. It will thus be instructive to first informally examine a few traces of actual audio traffic and identify a number of characteristic aspects of the interaction between network delay and packet audio playout.

Figures 1–3 plot the variable portion of the delay between a packet's transmission at a sender and its reception at a receiver as a function of the time at which the packet was transmitted at the sender. The propagation component of the end-to-end delay has been removed by subtracting out the minimum of the measured end-to-end delays in the entire delay trace (presumably the case in which there is little or no queueing of the packet in intermediate routers). Note that, by considering only the variable delay component, the issue of sender and receiver clock synchronization can be avoided. The variable delay component of each packet is plotted as a diamond on the graph. Dotted-line rectangles are used to distinguish talkspurts from each other showing which packets belong to which talkspurt. The width of a rectangle in the figures represents the length of a talkspurt and its height represents the largest variable portion of network delay over all packets within that talkspurt. A packet is generated every 20 ms during a talkspurt, and hence a missing dot at a 20-ms interval within a talkspurt indicates a lost packet within the network.

The delay traces shown in Fig. 1, as well as all other traces reported in this paper, were collected using NeVoT [Sch92], an audio conferencing tool that allows both point-to-point or multicast connections. NeVoT has a tracing mechanism that can collect timestamps of packets sent and received, RTP sequence numbers [SCFJ95], and vat virtual timestamps of packets. In our experiments and simulations we used vat virtual timestamps Packet audio was encoded in 8-KHz PCM mode and the packetization unit time was 20 ms. The sending and receiving hosts, the start time and date of the trace, the trace length, and an indication of whether packets were sent as unicast or multicast packets are indicated in Table 1. Traces 4, 5, and 6 are from our earlier work, and are described further in [RKTS94]. Traces 1 through 3 are new traces consisting of the transmission of the audio component of a recording with both female and male voices. Figures 1–3 are all taken from Trace 1 in Table 1.

Delay spikes are evident in Figs. 1, 2, and 3. Figure 1 shows a spike whose delay is less than an order of magnitude greater than other "baseline" delays and whose duration is short enough to be contained in a single talkspurt. Figures 2 and 3 show larger spikes with delay peaks that are almost an order of magnitude larger than the "baseline" delays. A large spike can either be contained in one talkspurt, as in Fig. 2, or can span several talkspurts, as in Fig. 3. In Trace 1 of Table 1, there are 23 such conspicuously large spikes; 10 of





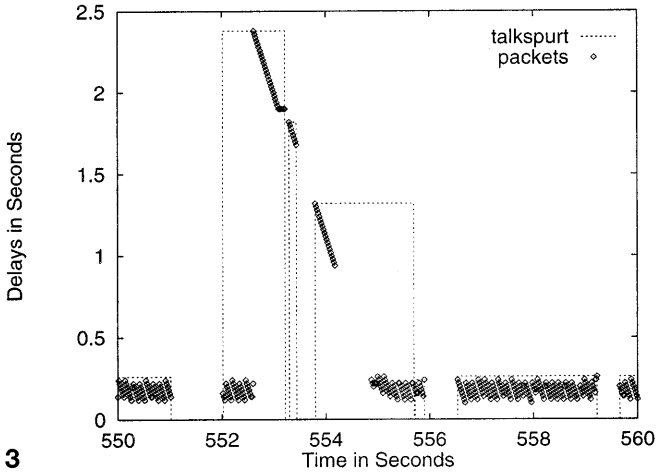


Fig. 1. Small spike

Fig. 2. Big spike

Fig. 3. Spike spanning several talkspurts

Table 1. Trace details

Trace #	Sender	Receiver	Start time(Sender)	Duration	Multicast
1	UMass	GMD Fokus	08:41pm 6/27/95(Tu)	1348 secs	No
2	UMass	GMD Fokus	09:58am 7/21/95(Fr)	1323 secs	Yes
3	UMass	GMD Fokus	11:05am 7/21/95(Fr)	1040 secs	No
4	INRIA	UMass	09:20pm 8/26/93(Th)	580 secs	No
5	UCI	INRIA	09:00pm 9/18/93(Sa)	1091 secs	No
6	UMass	Osaka Univ.	00:35am 9/24/93(Fr)	649 secs	No

these are contained in one talkspurt, 9 span two talkspurts, and the remaining 4 span three talkspurts.

Previous studies [Bol93, SGAJ93, RKTS94] have indicated the presence of "spikes" in end-to-end Internet delays. Bolot [Bol93] conjectures that, with periodically generated packets (as is the case with our audio packets and as was the case in [Bol93, SGAJ93, RKTS94]), the initial steep rise in the delay spike and the linear, monotonic decrease spike after the initial rise, is due to "probe compression" – the accumulation of a number of packets from the connection under consideration (the audio session, in our case) in a router queue behind a large number of packets from other sources. We note that probe compression is a plausible *conjecture* about the cause(s) of delay spikes. Validation of this conjecture would require careful measurements of packet traffic and its delay at the routers where congestion occurs. Our work [KKT96] discusses the many difficulties involved in making such measurements without privileged access to the routers.

Note that when a delay spike is properly contained within a talkspurt, the next opportunity to change the playout delay (i.e., at the beginning of the next talkspurt) occurs *after* the delay spike terminates. In such a case, it is not possible to adaptively react to the delay spike, since the delay spike is already over (i.e., the delay has returned to its baseline value) by the next talkspurt and any packets that were so excessively delayed during the delay spike that they missed their playout time have already been lost. In cases where a delay spike spans multiple talkspurts, however, it *is* advantageous to quickly react to the delay spike, as discussed in [RKTS94]. Note also that the "baseline" delays fluctuate less compared to spikes and as a result their delay distribution does not change significantly over time.

These two observations form the basis for the new delay adaptation algorithm to be presented in Sect. 4. First, however, we address the question of determining the playout delays incurred under a theoretically optimum playout delay adjustment algorithm. We do this in the following section.

3 Optimum average playout delay

In previous works [Coh77, Mon83, WF83, RKTS94], the tradeoff between the average playout delay and loss due to late packet arrival is used as the performance measure in comparing one adaptive playout delay adjustment algorithm with another – a tradeoff which we also use in this paper. We have chosen to consider loss and delay on a per-packet rather than per-talkspurt basis for two reasons. First, we note that the lengths of talkspurts depend on silence detection algorithms and their parameters. Per-talkspurt results are thus closely tied to the silence detection algorithm used. More importantly, different talkspurts have different lengths. One might argue that, in determining an overall performance measure, per-talkspurt measures could be weighted by the length of the talkspurt. In a sense, we are already doing so by considering individual per-packet delay and loss measures, and requiring that all packets within the same talkspurt be played out periodically.

Here, a playout delay (or, more accurately, end-to-end application-to-application delay) is defined to be the difference between the playout time at the receiver and the generation time at the sender. We refer to Fig. 4 to show the timing information of audio packets and formally define the average playout delay.

Consider a trace consisting of M talkspurts. We define the following quantities:

- t_k^i: sender timestamp of the i-th packet in the k-th talkspurt.
- a_k^i: receiver timestamp of the i-th packet in the k-th talkspurt.
- n_k: number of packets in the k-th talkspurt. Here, we only consider those packets actually received at the receiver.
- N: total number of packets in a trace, $N = \sum_{k=1}^{M} n_k$.

The playout time of a packet depends on which algorithm is used at the receiver to estimate the playout delay of the packet. Consider a playout algorithm A. Then $p_k^i(A)$ is the playout timestamp of the i-th packet in the k-th talkspurt under A. When it is obvious which algorithm is used, we omit the parameter A. If the i-th packet of the k-th talkspurt arrives later than $p_k^i(A)$ (i.e., $p_k^i(A) < a_k^i$), it is considered lost. Otherwise, it is played out with the playout delay of $(p_k^i(A) - t_k^i)$. Let $r_k^i(A)$ be an indicator variable for whether the i-th packet of the k-th talkspurt arrives before its playout time, as computed by playout algorithm A:

$$r_k^i(A) = \begin{cases} 0, \text{ if } \ p_k^i(A) < a_k^i \\ 1, \text{ otherwise.} \end{cases}$$

The total number of packets played out under Algorithm A is denoted as $N(A)$ and computed using $r_k^i(A)$:

$$N(A) = \sum_{k=1}^{M} \sum_{i=1}^{n_k} r_k^i(A).$$

Then the average playout delay of those played-out packets is defined as:

$$\frac{1}{N(A)} \sum_{k=1}^{M} \sum_{i=1}^{n_k} r_k^i(A)(p_k^i(A) - t_k^i).$$

If there are N packets in a trace and, among them, $N(A)$ packets are played out under Algorithm A, the loss percentage l is:

$$l = \frac{N - N(A)}{N} \times 100.$$

Our goal in this section is to present a bound on the optimum (minimum) average playout delay for a given number of packet losses (due to late arrivals) at the receiver for a given packet delay trace. To illustrate this problem, suppose we are given a trace of sender and receiver timestamps of audio packets in an audio session. Suppose now that we are free to set the playout delays of the various talkspurts to whatever values we choose such that only one packet (in the entire trace) will be lost. That is, we want to lose a packet, so that the average playout delay over all played-out packets in the trace is minimized. This provides a bound on the average delay achievable by *any* delay adjustment algorithm, given that only one out of all the received packets in the trace is lost. We then repeat this procedure for two packet losses, and so on.

The obvious way to calculate the exact minimum bound is as follows: for a given number of dropped packets, say i, determine all possible configurations of i lost packets, compute the average playout delay for each configuration, and compute the minimum of these average playout delays. The number of computations of i lost packets grows exponentially in N. We reduce the computational cost by instead deriving upper and lower bounds on the optimum (minimum) playout delay for a given loss percentage, that requires an amount of computation that is polynomial in the number of packets in the trace.

Section 3.1 provides the background needed for presenting these algorithms and defines our notation. In Sects. 3.2 and 3.3, we describe our approach in detail.

3.1 General overview

Below we introduce additional terminology to be used in following sections.

- $\hat{d}_k^i$: delay between the generation of the i-th packet of the k-th talkspurt at the sender and its reception at the receiver, namely $\hat{d}_k^i = a_k^i - t_k^i$. We do not need to assume that the sender and receiver clocks are synchronized, but do need to assume that they do not drift.
- $\hat{d}$: $\hat{d} = \min_{1 \le k \le M, 1 \le i \le n_k} \{\hat{d}_k^i\}$.
- d_k^i: normalized delay of the i-th packet of the k-th talkspurt. This accounts only for the variable portion of the end-to-end delay. We will use this normalized delay (rather than $\hat{d}_k^i$) in calculating the bounds of the optimum average playout delay,

  $$d_k^i = \hat{d}_k^i - \hat{d}.$$

- $d_k^{(i)}$: i-th smallest normalized delay in the k-th talkspurt.

Recall that the playout delay of all packets in the k-th talkspurt should be the *same* due to the periodic nature of packet generation within a talkspurt at the sender and

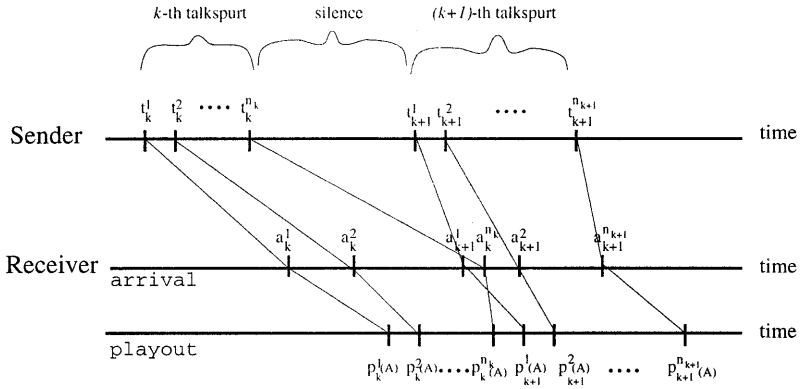


Fig. 4. Timings associated with the i-th packet in the k-th talkspurt

periodic playout at the receiver. Given an algorithm A, we denote the playout delay of the k-th talkspurt as $\hat{p}_k(A)$. The playout time of the i-th packet in the k-th talkspurt is then:

$$p_k^i(A) = t_k^i + \hat{p}_k(A). \quad (1)$$

In later sections, where there is no confusion about which algorithm is used, we will denote $\hat{p}_k(A)$ simply as $\hat{p}_k$.

To successfully play out i packets from the k-th talkspurt, at least i packets during the k-th talkspurt must arrive before their playout time calculated by Eq. 1. To achieve this goal, the playout delay of an algorithm must be set to be larger than or at least equal to $d_k^{(i)}$. In practice (i.e., in an actual on-line implementation), $d_k^{(i)}$ cannot be known in advance before all the packets belonging to the k-th talkspurt arrive, but $p_k^i(A)$ must often be determined before these packets arrive. Since our bounding algorithms are off-line algorithms, we assume that t_k^i and a_k^i are available at the start of their executions.

The packet arrival times in a talkspurt are not the only quantities that determine the playout delay. The long playout delay of one talkspurt may force the playout of packets of the subsequent talkspurt to be further delayed. For example, consider a playout delay algorithm A. Assume that, in order to play out all packets contained in the k-th and $(k+1)$-th talkspurts, algorithm A sets the playout delays $\hat{p}_k(A)$ and $\hat{p}_{k+1}(A)$ to $d_k^{(n_k)}$ and $d_{k+1}^{(n_{k+1})}$, respectively. The playout time of the first packet in the $(k+1)$-th talkspurt, $p_{k+1}^1(A)$, becomes $t_{k+1}^1 + \hat{p}_{k+1}(A)$. If the playout time of the first packet of the $(k+1)$-th talkspurt comes before the playout time of the last packet of the k-th talkspurt, i.e., $t_k^{n_k} + \hat{p}_k(A) > t_{k+1}^1 + \hat{p}_{k+1}(A)$, then the beginning of the $(k+1)$-th talkspurt overlaps the end of the k-th talkspurt at the receiver. We refer to this as a *collision* of the k-th and $(k+1)$-th talkspurts. The condition for a collision can be summarized as

$$t_k^{n_k} + \hat{p}_k(A) > t_{k+1}^1 + \hat{p}_{k+1}(A), \quad \text{or}$$
$$\hat{p}_k(A) > (t_{k+1}^1 - t_k^{n_k}) + \hat{p}_{k+1}(A). \quad (2)$$

In order to avoid such collisions, the playout delay of the subsequent talkspurt must be increased. Note that collisions can occur in a cascade when the above collision condition persists over several talkspurts in a row. We call such a sequence of collisions a *collision train*.

In order to provide a lower bound of the optimum average playout delay, we first simplify the problem by ignoring the effect of collisions and assume that packets of talkspurts in a collision are allowed to overlap. Note that this underestimates the optimum average playout delay (since, in practice, some talkspurts would be further delayed in order to avoid overlapping packets from different talkspurts), and thus represents a potentially unachievable lower bound on the minimum playout delay for a given loss. Later we will account for collisions. Our algorithms are based on dynamic programming [Ber87].

3.2 Off-line algorithm without collisions

Our first algorithm provides a lower bound of the optimum average playout delay for a given loss percentage. Recall that it is obtained by ignoring additional delays due to collisions, i.e., the effect of one talkspurt's long playout delay on that of the subsequent talkspurt is not considered. We define $D(k,i)$ to be the minimum average playout delay possible when choosing i packets to be played out from the k-th to M-th talkspurts. Using dynamic programming, calculating $D(1,i)$ for i from 0 to N generates the lower bounds on the optimum average playout delay for loss percentages of 100% down to 0%. It is described by the following equation.

$$D(k,i) = \begin{cases} 0, & \text{if} \quad i = 0 \\ d_k^{(i)}, & \text{if} \quad k = M \quad \text{and} \quad i \le n_M \\ \infty, & \text{if} \quad k = M \quad \text{and} \quad i > n_M \\ \min\limits_{0 \le j \le i} \left(((i-j)D(k+1, i-j) + j d_k^{(j)})/i \right), & \\ \quad \text{otherwise.} & \end{cases} \quad (3)$$

In the following theorem, we prove that $D(k,i)$ in Eq. 3 is the minimum average playout delay when choosing i packets from k-th to M-th talkspurts to be played out, for the case that collisions are ignored.

Theorem 1. *$D(k,i)$ is the minimum average playout delay of choosing i packets to be played out from k-th to M-th talkspurts.*

Proof. Assume that $D(x,y)$ is minimal for $k+1 \le x \le M, 0 \le y \le i-1$, but that $D(k,i)$ obtained via Eq. 3 is not. Here, j packets are chosen from the k-th talkspurt with the playout delay to be $d_k^{(j)}$, and $(i-j)$ packets from $(k+1)$-th to M-th talkspurts. Those $(i-j)$ packets have a playout delay of $D(k+1, i-j)$. Thus,

$$iD(k,i) > (jd_k^{(i)} + (i-j)D(k+1, i-j)) ,$$

which contradicts the definition of $D(k,i)$ in Eq. 3, namely that:

$jd_k^{(i)} + (i-j)D(k+1, i-j) \geq iD(k,i)$.

Thus, $D(k,i)$ is minimal. ∎

3.3 Off-line algorithm with collisions

The second algorithm computes an upper bound on the optimum average playout delay. It relies on dynamic programming as in Sect. 3.2, but is more complicated due to the manner in which it accounts for collisions. In the first algorithm, the playout delay of a talkspurt is simply $d_k^{(i)}$, given k and i at each step of computation in Eq. 3. To take collisions into account, condition 2 is checked for every k and i in the second algorithm. If there is a collision, the playout delay of the latter of the two colliding talkspurts is adjusted to a larger value to avoid a collision. Checking condition 2 for collisions requires not only the playout delays of two adjacent talkspurts but also the sender timestamps of the last and first packets of each talkspurt. The identity of the first and last packets played out in a given talkspurt vary, depending on which packets are chosen by the bounding algorithm to be played out. To track those packets played out at every step of the computation, we introduce the vector $C(k,i)$, whose components are the sets of packets belonging to talkspurts $k, \ldots, M$ that are played out. Here, i denotes the total number of packets contained within these sets.

An informal description for the second algorithm is as follows. Define $D(k,i)$ as before. Given k and i, assume that $D(x,y)$ is known for $k+1 \leq x \leq M, 0 \leq y \leq i-1$. The calculation of $D(k,i)$ consists of choosing j packets from the k-th talkspurt, and $(i-j)$ packets from the $(k+1)$-th to M-th talkspurts to be played out so as to minimize the average playout delay of those packets. If playing out j packets from the k-th talkspurt results in a collision between k-th and $(k+1)$-th talkspurts, then the playout delay of the $(k+1)$-th talkspurt becomes larger and is accounted for when calculating the average playout delay. Choosing j packets from the k-th talkspurt may cause a cascade of collisions, in which case the playout delays are increased for all talkspurts involved in the collision, and if so, more delays are added to calculate the average playout delay. $C(k,i)$ records which packets are chosen for the minimum total sum at this step of computation.

Let us now introduce the additional notation used in the second algorithm. Throughout, $\mathbf{S} = (S_1, \ldots, S_k, \ldots, S_M)$ is an M-dimensional vector, where $S_k \subseteq \{1, \ldots, n_k\}$ is a set of packets from the k-th talkspurt. If $|S_k| = l$, then S_k contains the identities (indices) of the l packets with the l smallest normalized delays in the k-th talkspurt. Henceforth, $\mathbf{S}$ will be referred to as a playout vector. Let $\mathbf{e}_k(i,j)$ be an M-dimensional vector whose components are all the empty set, $\emptyset$, except for the k-th component which is the set of packets with the $(i+1)$-th through $(i+j)$-th smallest normalized delays in the k-th talkspurt. Last, if $\mathbf{S}$ and $\mathbf{X}$ are playout vectors whose components are sets, then $\mathbf{S} \cup \mathbf{X}$ is understood to be the vector whose components are the unions of the components in $\mathbf{S}$ and $\mathbf{X}$.

– $s_k(\mathbf{S})$: difference in the sender timestamps of the last packet played out from the k-th talkspurt and of the first packet from the $(k+1)$-th talkspurt, given $\mathbf{S}$. This value is used in adjusting the playout delay in the case of a collision. It is given as

$$s_k(\mathbf{S}) = \begin{cases} 0, & \text{if } k = M, S_k = \emptyset, \text{ or } S_{k+1} = \emptyset \\ \min\{t_{k+1}^i : i \in S_{k+1}\} - \max\{t_k^i : i \in S_k\}, & \\ \quad \text{otherwise.} & \end{cases}$$

$s_k(\mathbf{S})$ can be interpreted as the length of the silence period at the sender between the k-th and $(k+1)$-th talkspurts consisting of S_k and S_{k+1}, respectively.

– $\hat{p}_k(\mathbf{S})$: playout delay of the k-th talkspurt when the playout vector is $\mathbf{S}$. It differs from $d_k^{(|S_k|)}$ in the case of a collision. It is given by the following recursion:

$$\hat{p}_k(\mathbf{S}) = \begin{cases} 0, & \text{if } S_k = \emptyset, \\ d_1^{(|S_1|)}, & \text{if } k = 1, \\ \max\{d_k^{(|S_k|)}, \hat{p}_{k-1}(\mathbf{S}) - s_{k-1}(\mathbf{S})\}, & \\ \quad \text{otherwise.} & \end{cases} \quad (4)$$

If a collision has occurred, the playout delay $\hat{p}_k(\mathbf{S})$ becomes $\hat{p}_{k-1}(\mathbf{S}) - s_{k-1}(\mathbf{S})$. It is $d_k^{(|S_k|)}$ otherwise.

– $\Delta(\mathbf{S}, k, j)$: sum of the increases in the playout delays incurred in the $(k+1)$-th to M-th talkspurts from collisions due to the introduction of j additional packets to be played out in the k-th, talkspurt given that the playout vector was originally $\mathbf{S}$. It is given by the following expression.

$$\Delta(\mathbf{S}, k, j) = \sum_{x=k+1}^{M} |S_x|(\hat{p}_x(\mathbf{S} \cup \mathbf{e}_k(|S_k|, j)) - \hat{p}_x(\mathbf{S})). \quad (5)$$

If the introduction of j additional packets to the k-th talkspurt incurs collisions, the playout delays of the $(k+1)$-th to the last talkspurts in a collision train become larger. The difference between the new larger playout delay and the original playout delay of the talkspurt is multiplied by the number of packets chosen from that talkspurt. These values are summed to obtain the total amount of additional playout delay.

In the second algorithm, not only $D(k,i)$ but also $C(k,i)$ is calculated at each step of computation. The equations for $D(k,i)$ and $C(k,i)$ are as follows:

$$D(k,i) = \begin{cases} 0, & \text{if } i = 0 \\ d_k^{(i)}, & \text{if } k = M, i \leq n_M \\ \infty, & \text{if } k = M, i > n_M \\ \min\limits_{0 \leq j \leq i} \left\{ \frac{(i-j)D(k+1,i-j) + jd_k^{(j)} + \Delta(C(k+1,i-j),k,j)}{i} \right\}, & \\ \quad \text{otherwise.} & \end{cases} \quad (6)$$

The only difference between Eqs. 3 and 6 of the two bounding algorithms is the extra term $\Delta()$ in Eq. 6, which accounts for the extra delays incurred by collisions.

$$C(k,i) = \begin{cases} (\emptyset,\ldots,\emptyset), & \text{if } i = 0 \\ \mathbf{e}_k(0,i), & \text{if } k = M, i \le n_M \\ \mathbf{e}_k(0,n_M), & \text{if } k = M, i > n_M \\ C(k+1,i-j) \cup \mathbf{e}_k(0,j) & \\ \quad \text{where } j = \arg\min_{0\le j\le i} & \\ \quad \left\{ \frac{(i-j)D(k+1,i-j)+j d_k^{(j)}+\Delta(C(k+1,i-j),k,j)}{i} \right\}, & \\ \quad \text{otherwise.} & \end{cases} \quad (7)$$

When j out of i packets are chosen from the k-th talkspurt for some $D(k,i)$, the indices of those packets are in $\mathbf{e}_k(0,j)$. The union of $\mathbf{e}_k(0,j)$ and $C(k+1,i-j)$ is assigned to $C(k,i)$.

The second algorithm accounts for the collisions in its calculation, but does not generate the exact optimum average playout delay. A close look at Eq. 6 reveals why. Consider a case, where, for some $D(k,i)$ resulting from the algorithm, j packets are played out from the k-th talkspurt and cause a collision between the k-th and $(k+1)$-th talkspurt. The j packets from the k-th talkspurt chosen to be played out are those with the j smallest normalized playout delays. The algorithm thus does *not* consider the case where playing out some other j packets from the k-th talkspurt would not incur a collision (or as severe a collision) to the $(k+1)$-th talkspurt.

It is worth noting that it is not necessary to keep track of the identities of every packet to be played out in $C(k,i)$. It suffices to keep track of the number of packets in each talkspurt, the normalized delays, and the sender timestamps of the earliest and latest of the packets being played out. We found it simpler to present the algorithm as described above.

3.4 Computational complexity

The time complexity of the first algorithm is $O(M \cdot N^2)$ and the space complexity is $O(M \cdot N)$. The total number of packets in a trace, N, easily exceeds 30,000 for a trace lasting longer than 10 min. A closer look at Eq. 3 reveals that j can vary only from 0 to $\max_{1\le k\le M}\{n_k\}$ in a real trace. If all talkspurts but one, in a trace, contain only one packet, then $\max_{1\le k\le M}\{n_k\}$ is approximately N, and the time and space complexities are as indicated above. On the other hand, if all talkspurts in a trace have the same number of packets, then $\max_{1\le k\le M}\{n_k\}$ is N/M. This decreases the time complexity to $O(N^2)$. The space complexity also reduces to $O(N)$.

The second algorithm has higher time and space complexities; the time complexity is $O(M^2 \cdot N^2)$, and the space complexity also $O(M^2 \cdot N^2)$.

In some cases, it is possible to partition talkspurts into groups such that no two talkspurts from different groups collide. In this case, we can apply the second bounding algorithm to groups, and treat groups as talkspurts under the first bounding algorithm to obtain the upper bounds. This two-step computation can be used to reduce the algorithm's running time. If such a grouping eventually yields groups with only one talkspurt, the exact optimum average playout delay is calculated using the first bounding algorithm for the trace.

In this section, we have introduced two algorithms that provide the lower and upper bounds on the optimum average playout delay. The results from these two algorithms will be used later in Sect. 4.4 to compare the performance of various on-line adaptive algorithms.

4 On-line adaptive algorithm based on past history

In this section we present a new adaptive, on-line playout delay algorithm and discuss its motivation, design, and implementation. In Sect. 4.1, our observations regarding existing playout delay algorithms, and how they motivated the design of a new algorithm, are discussed. In Sect. 4.2, the new algorithm is presented in pseudo-code. In Sect. 4.3, we look into the implementation issues of the algorithm. In Sect. 4.4, we compare the new algorithm with others and with the bounds presented in Sect. 3.

4.1 Motivation

Let us first consider the playout delay adjustment algorithms Algorithm 1 and Algorithm 4 from [RKTS94]. We renumber these as Algorithms 1 and 2 in this paper. These two algorithms are based on stochastic gradient algorithms used in estimation and control theory [LS83] and operate by estimating two statistics characterizing the network delay incurred by audio packets: the delay itself, and a variational measure of the observed delays. Each of these estimates is recomputed each time a new packet arrives.

$\alpha = 0.998002$
$\hat{u}_k^i = \alpha \hat{u}_k^{i-1} + (1-\alpha)\hat{d}_k^i$
$\hat{v}_k^i = \alpha \hat{v}_k^{i-1} + (1-\alpha)\left|\hat{u}_k^i - \hat{d}_k^i\right|$

Fig. 5. Algorithm 1

IF (`mode` == NORMAL)
  IF ($\left|\hat{d}_k^i - \hat{d}_k^{i-1}\right| > \left|\hat{v}_k^{i-1}\right| * 2 + 800$)
    `var` = 0;
    `mode` = SPIKE;
  ELSE
    `var` = `var`/2 + $\left|(\hat{d}_k^i - \hat{d}_k^{i-1})/8 + (\hat{d}_k^i - \hat{d}_k^{i-2})/8\right|$;
  IF (`var` $\le$ 63)
    `mode` = NORMAL;
  $\hat{d}_k^{i-2} = \hat{d}_k^{i-1}$
  $\hat{d}_k^{i-1} = \hat{d}_k^i$
  return;
IF (`mode` == NORMAL)
  $\hat{u}_k^i = 0.125 * \hat{d}_k^i + 0.875 * \hat{u}_k^{i-1}$;
ELSE
  $\hat{u}_k^i = \hat{u}_k^{i-1} + \hat{d}_k^i - \hat{d}_k^{i-1}$;
$\hat{v}_k^i = 0.125 * \left|\hat{d}_k^i - \hat{u}_k^i\right| + 0.875 * \hat{v}_k^{i-1}$;
$\hat{d}_k^{i-2} = \hat{d}_k^{i-1}$
$\hat{d}_k^{i-1} = \hat{d}_k^i$
return;

Fig. 6. Algorithm 2

Algorithms 1 and 2 are in [RKTS94], but are presented here for completeness. Let $\hat{u}_k^i$ and $\hat{v}_k^i$ be an estimate of the packet delay and variational measure of the i-th packet of

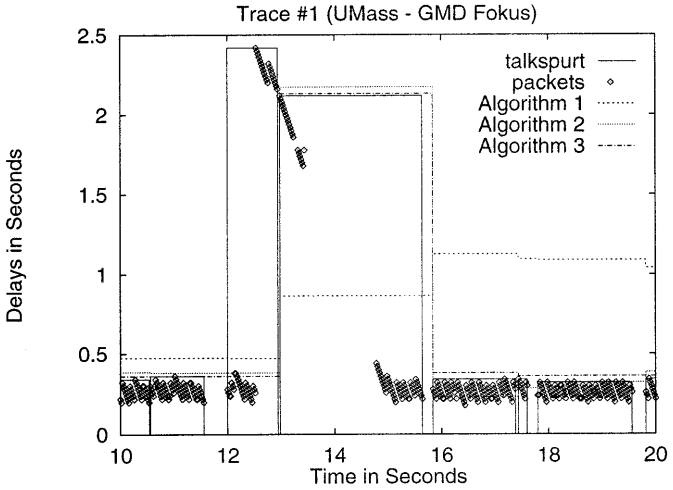


Fig. 7. Delay estimates of three algorithms

the k-th talkspurt. At the beginning of a new talkspurt, the playout delay $\hat{p}_k$ is estimated as follows:

$$\hat{p}_k = \hat{u}_{k-1}^{n_{k-1}} + \beta \hat{v}_{k-1}^{n_{k-1}} \ . \tag{8}$$

Here, β is a variation coefficient and provides some slack in playout delay for arriving packets. The larger the coefficient, the more packets that are played out at the expense of longer playout delays. It is thus a parameter which can be used to control the delay/loss tradeoff incurred under Algorithms 1 and 2. It is used as such later in our simulations.

Algorithms 1 and 2 both use Eq. 8 to determine the playout delay for a talkspurt; they only differ in how they calculate $\hat{u}_k^i$ and $\hat{v}_k^i$. The algorithms themselves are given in Figs. 5 and 6.

Algorithm 1 is a linear filter that is slow in catching up with a change in delays, but is good at maintaining a steady value, when $(1 - \alpha)$, the gain of the estimator, is set to be very low. We use a specific value of α = 0.998002 chosen for NeVoT1.4 in our simulations. The choice of α is further discussed in Sect. 4.4

Algorithm 2 shown in Fig. 6 has two modes of operation, depending on whether a spike has been detected. In normal mode, it operates like Algorithm 1 with a different gain, but in spike-detection mode, u_k^i is updated differently.

Figure 7 plots the playout delay of Algorithms 1 and 2, as well as that of our new algorithm, Algorithm 3 (to be described shortly), for a given delay trace. In Fig. 7, the x-axis indicates the elapsed time since the beginning of a session. A diamond plots the end-to-end queueing delay of a packet received at that point in time. Solid rectangles delineate talkspurt boundaries. The playout delay computed by each of the three algorithms is indicated by a horizontal line that is as long (in the x-dimension) as the talkspurt.

From the figure, we see that Algorithm 1 computes the playout delay in such a way that the playout delay begins to increase only well after the delay spike has occurred. Note that, under Algorithm 1, the packets at the beginning of the talkspurt beginning at approximately 13 s are lost. This is because Algorithm 1 uses a delay estimator that reacts too slowly to delay spikes. Algorithm 2, on the other hand, computes a playout delay that reacts quickly to the delay spike. For example, the playout delay computed via Algorithm 2 for this high-delay talkspurt is such that no packets are lost. In the following talkspurt, however, the playout delay is underestimated by Algorithm 2 and many packets are lost. The problem here is that Algorithm 2 attempts to track the network delays too closely and loses packets whenever its delay estimate is small, and the following talkspurt begins with packets that have suffered an even slightly higher delay (i.e, the talkspurt beginning near time 16 and beyond). In the following section, we discuss the design of a new algorithm based on these observations.

```
( 1) IF (mode == SPIKE)
( 2)   IF (d̂_i^k ≤ tail * old_d) /* the end of a spike */
( 3)     mode == NORMAL;
( 4) ELSE
( 5)   IF (d̂_k^i > head * p̂_k)     /* the beginning of a spike */
( 6)     mode = SPIKE;
( 7)     old_d = p̂_k;          /* save p̂_k to detect the end
                                   of a spike later */
( 8)   ELSE
( 9)     IF (delays[curr_pos] ≤ curr_delay)
(10)       count -= 1;
(11)     distr_fcn[delays[curr_pos]] -= 1;
(12)     delays[curr_pos] = d̂_k^i;
(13)     curr_pos = (curr_pos+1) % w;
(14)     distr_fcn[d̂_k^i] += 1;
(15)     IF (delays[curr_pos ] < curr_delay)
(16)       count += 1;
(17)     WHILE (count < w × q)
(18)       curr_delay += unit;
(19)       count += distr_fcn[curr_pos];
(20)     WHILE (count > w × q)
(21)       curr_delay -= unit;
(22)       count -= distr_fcn[curr_pos];
```

Fig. 8. Algorithm 3

```
(1)   IF (mode == SPIKE)
(2)     p̂_k = d̂_k^1;
(3)   ELSE (mode == NORMAL)
(4)     p̂_k = curr_delay;
```

Fig. 9. Playout Delay Estimation of Algorithm 3

4.2 Design

Let us first informally describe Algorithm 3. The key idea behind our new algorithm is to collect statistics on packets that have already arrived and to use them to estimate the playout delay. Instead of using the linear filter mechanism, each packet's delay is logged and the distribution of packet delays is updated at every packet arrival. When a new talkspurt starts, our algorithm calculates a given percentile point q in the distribution function of the packet delays for the last w packets, and uses it as the playout delay for the new talkspurt. As in Algorithm 2, Algorithm 3 detects spikes and behaves accordingly: once a spike is detected, it stops collecting packet delays and follows the spike until it detects the end of a spike. Upon detecting the end of a delay spike, it resumes its normal operation. As shown in Sect. 2, the

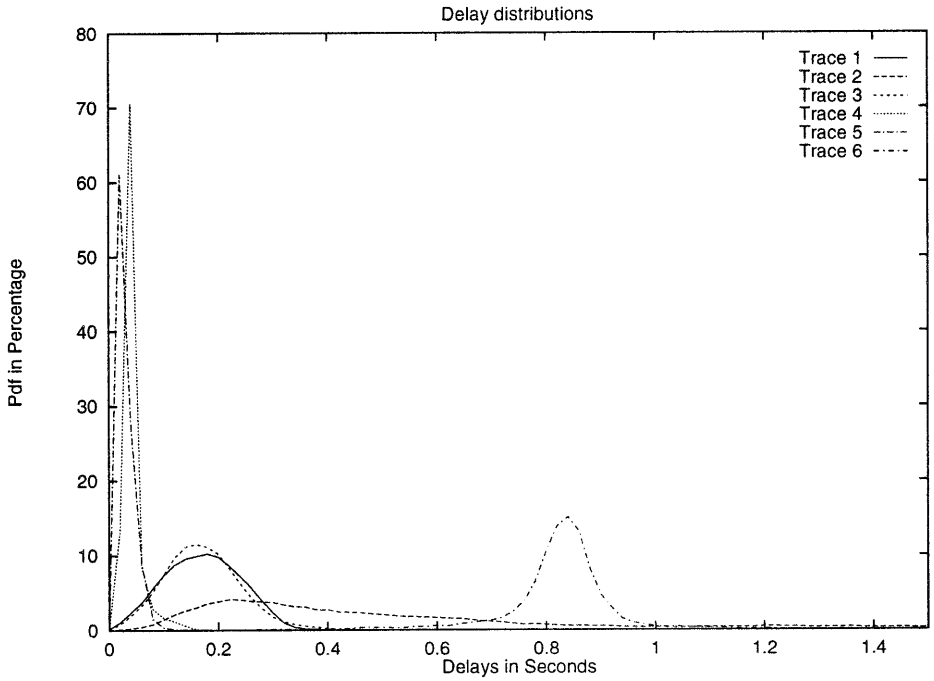


Fig. 10. Delay distribution of traces

delays of packets in a spike decrease in a linear fashion. Thus, it is reasonable to use the delay of the first packet of a talkspurt as the playout delay for the talkspurt, if a new talkspurt begins during a spike. In the next paragraph, we give a high-level description of the algorithm. For ease of understanding, the algorithm is presented in C-language-like pseudo code in Fig. 8, and is referred to during the design description below.

Algorithm 3 operates in two modes. For every packet that arrives at the receiver, the algorithm checks the current mode and, if necessary, switches its mode to the other in lines 1–7 of Fig. 8. Lines 9–22 update the delay distribution in normal mode. If a packet arrives with a delay that is larger than some multiple of the current playout delay, the algorithm switches to spike-detection mode. The end of a spike is detected in a similar way: if the delay of a newly arrived packet is less than some multiple of the playout delay before the current spike, the mode is set back to normal. Two parameters *head* and *tail* are used in lines 5 and 2 of Fig. 8 in detecting the beginning and end of a spike. To determine the sensitivity of the algorithm to these parameters, we varied *head* from 2 to 20 and *tail* from 1 to 4 and evaluated Algorithm 3 using our delay traces. We found the algorithm to be relatively insensitive to values of *head* between 2 and 10, and of *tail* between 1 to 3. We chose 4 and 2 for *head* and *tail* in our simulations, as multiplication by powers of 2 can be implemented as shift operations.

Depending on the current mode, the playout delay for the next talkspurt is estimated differently in each mode, as shown in Fig. 9. In spike-detection mode, the delay of the first packet of a talkspurt becomes the estimated playout delay for the talkspurt. Otherwise, `curr_delay`, which is the given percentile point of delay based on previous statistics of packet delays, is used.

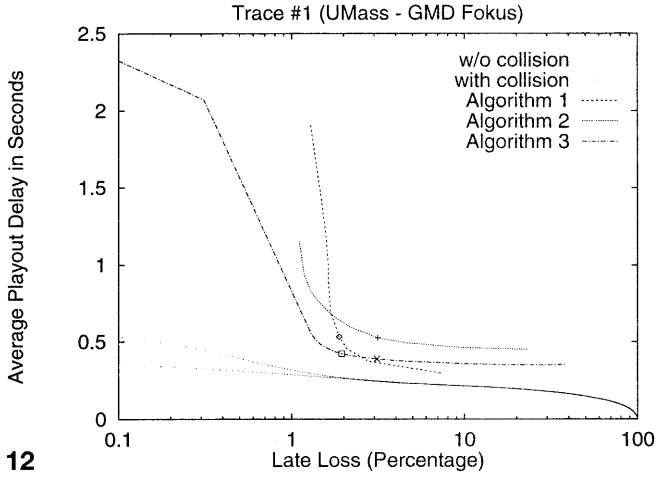
4.3 Implementation

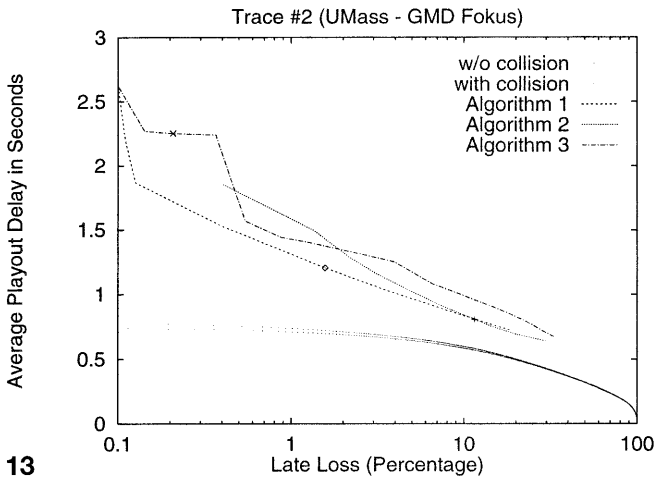
All of the algorithms are executed every time a packet arrives at the receiver. Since the packetization interval of audio packets varies from 16 ms to 32 ms [Jay80], the algorithms should be efficient enough to run 30–60 times a second and need to leave enough processing power for other activities. For Algorithm 3, lines 9–22 consist primarily of updating counters, and are integer operations. Theoretically, its time complexity is proportional to the number of packet delays, *w*, that are stored. However, since most delay distributions are bell-shaped (Fig. 10 plots these distributions from our six traces), it is expected to execute only a few loops, and thus, in practice, we expect that the time complexity per packet for this algorithm to be constant.

```
(1)   WHILE (there is a packet in a trace file)
(2)     fetch a packet;
(3)     first checkpoint;
(4)     packet processing of the algorithm;
(5)     second checkpoint;
```

Fig. 11. Playout delay estimation of Algorithm 3

To verify our hypothesis, a simple experiment was devised to measure the running time of our algorithm as well as that of Algorithm 1. Our measurements were performed as shown in Fig. 11. The difference in time between the first and second checkpoints is accumulated over the entire trace of packets and the sum is divided by the number of packets in a trace. We ran the simulator on an SGI Indy R4600 (134-MHz) IRIX 5.2 and used the `gettimeofday()` system call for checkpointing. If the simulator is interrupted by other processes between two checkpoints, the time difference between two checkpoints includes not only the running time of our simulator, but also that of other processes. This impedes the exact measurement of algorithm running time,





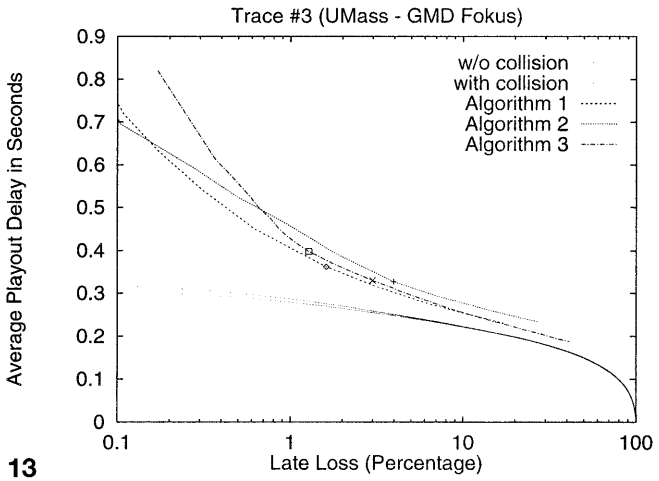
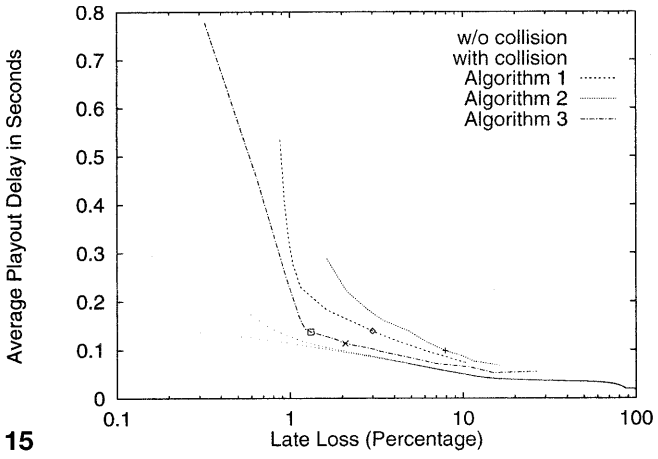
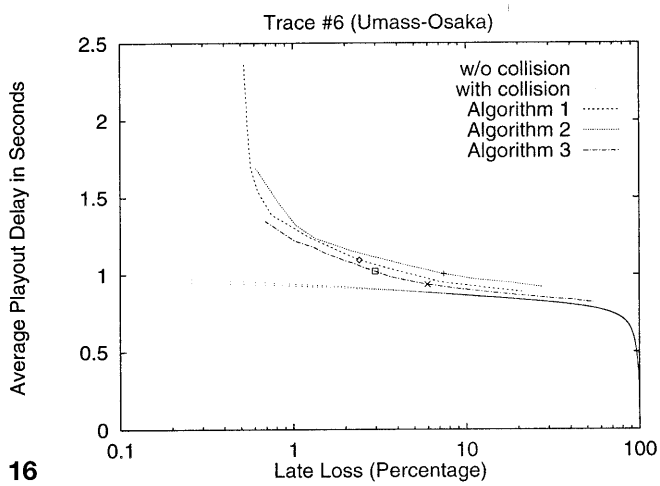


Fig. 12. Trace 1

Fig. 13. Trace 2

Fig. 14. Trace 3





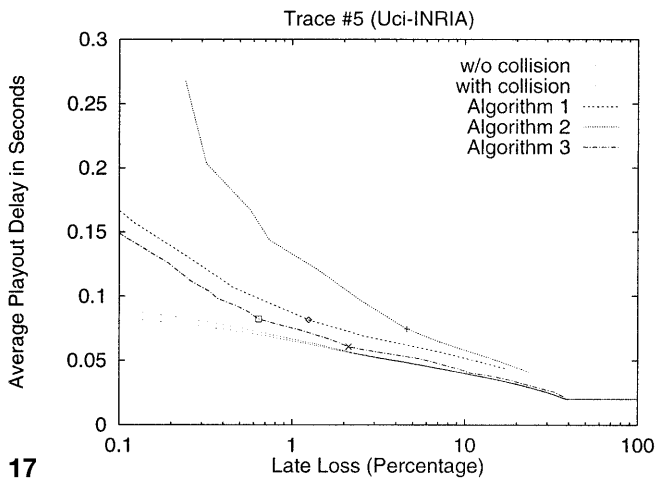


Fig. 15. Trace 4

Fig. 16. Trace 5

Fig. 17. Trace 6

and affects both algorithms. To minimize the effect of this extraneous measurement, our experiments were run under light loads, and also to checkpoint not per packet but rather per 10000 packets. In addition we performed the same measurement for the case that the time to execute the algorithm 10,000 times was measured. All experiments on six traces gave the same order of magnitude value under 200 μs for the per-packet processing of the algorithm – a relatively small amount of time given that packets are generated every 20 ms. Algorithm 1 was also simulated and run through the same

set of experiments. We found no significant difference in the running times of Algorithms 1 and 3.

The space complexity of Algorithm 3 is linear in *w*, because delays of the previous *w* packets must be stored. The length of the history determines how sensitive the algorithm is in adapting to the change. If it is too short, the algorithm will have a myopic view of the past and is likely to produce a poor estimate of the playout delay. If it is too long, the algorithm will keep track of an unnecessarily large amount of past history. One potential weakness of our algorithm is that it may be slow to adapt to a steady increase or decrease in the "baseline" delays in the case of clock drifts. The decision on the length of history was made after evaluating the algorithm with different lengths of history. For lengths of history below 10,000 packets, the performance degraded as the length became shorter. Above 10,000 packets, any performance enhancement was marginal. Thus, in the results reported in the following section, the length of history *w* is set to 10,000 packets, corresponding to 200 s of time in the absence of silence periods. This results in a memory requirement of 40,000 bytes with 4-byte integers – a negligible amount of memory in today's workstations.

4.4 Comparison of delay adaptation algorithms with bounds

As mentioned in the introduction to this paper, the performance metric we use to compare different delay adaptation algorithms is the average playout delay vs. loss percentage. To evaluate Algorithms 1–3, we designed and implemented a simulator that reads in the sender and receiver timestamps of each packet from a trace, determines if it has arrived before the playout time that is computed by a specific algorithm, and executes the algorithm. The simulator calculates the average playout delay and loss percentage for the given trace and outputs them. This allows us to compare the algorithms under the same conditions.

In Figs. 12–17, the average playout delay is plotted as a function of the loss percentage for each algorithm. In the absence of any specific reference, all figures mentioned in this section are Figs. 12–17.

For Algorithms 1 and 2, instead of using the buffer size as the control parameter to be varied to achieve different loss percentages (as was done in [RKTS94]), we here varied β in Eq. 8. The range of values for β varies from 1 to 20 in our simulations. In Figs. 12–17, a diamond for Algorithm 1 and a plus for Algorithm 2 are used explicitly to mark the β value of 4, which was used in [RKTS94].

For Algorithm 1, we ran a set of simulations to determine the sensitivity of the algorithm to the value of α. For $0.90 \leq \alpha \leq 0.999$, the algorithm's performance did not change dramatically. For $\alpha < 0.90$, however, the performance degraded. The specific value of 0.998002 was chosen for NeVoT1.4, and we used this value in our simulations.

Since Algorithm 3 does not use the mean or variational measure, it is parametrized by the percentile point of 50–100% on the figures. On the graphs of Algorithm 3, 99% and 97% points are marked with a square and a cross, respectively.

As the algorithms for bounding performance use normalized delays in their calculation of average playout delay, the average playout delay of Algorithms 1–3 is also normalized by subtracting $\hat{d}$ from it; these normalized average playout delays are plotted in the figures. The lower and upper bounds of the optimum average playout delay are always below the graphs of Algorithms 1–3, since they are the theoretically optimum (minimum) bounds of the average playout delay for any given loss percentage. Algorithms 1–3 yield a single point on the graph of an average playout delay versus a loss percentage for a fixed control parameter (variation coefficient or percentile point). The graphs of Algorithms 1–3 are drawn by connecting a finite set of those points.

Our figures illustrate several interesting points. First, note that the upper and lower bounds on the optimum playout delay versus loss tradeoff are quite close, as long as the loss percentage is 1% or more. Eqs. 3 and 6 thus provide very tight lower and upper bounds on the optimum average playout delay for loss percentages in the range of interest.

Trace 2 in Fig. 13 was collected between two multicast sites on the MBone during busy hours. The network loss percentage between the sender and the receiver is a horrendously high 58%. It also has a long blackout period of almost 2 min, when no packets arrived at the sender. This blackout phenomenon on the MBone is reported in [YKT].

Trace 3 in Fig. 14 was also collected during busy hours, but using unicast connections over the Internet. It suffered a network loss percentage of 17%, which is far lower than that of Trace 2, but still far from the desirable range of 2–5%. For Trace 3, all three algorithms show similar average playout delays near 0.35 s on the y-axis for marked points.

In Figs. 15–17, Algorithm 3 performs best in all points, nearly touching the optimum delay in Fig. 16. All four marked points on the graphs are close in their y-coordinates, but their x-coordinates are somewhat dispersed. The marked points of Algorithm 2 are consistently positioned to the right of other marked points on the x-axis, which means it drops more packets due to late arrival for a given playout delay. This verifies our previous observation that Algorithm 2 underestimates the playout delay after spikes.

5 Conclusion

In this paper, we have focused on the tradeoff between packet playout delay and packet playout loss. We presented algorithms for computing upper and lower bounds on the optimum (minimum) average playout delay for a given number of packet losses (due to late arrivals) at the receiver for a given trace of packet delays. These bounds were shown to be tight for a range of loss and delay values of interest, and are important, as they provide a bound on the achievable performance of *any* adaptive playout delay adjustment algorithm. We also presented a new adaptive delay adjustment algorithm that tracks the network delay of recently received packets and efficiently maintains delay percentile information. Our new algorithm was shown to outperform existing delay adjustment algorithms over a number of measured audio delay traces and performs close to the theoretical optimum over a range of parameter values of interest.

Acknowledgements. We wish to thank Henning Schulzrinne for providing us with the invaluable tool, NeVoT, to collect traces and allowing us to use his machine in Germany for experiments. We also thank Ramachandran Ramjee for use of his traces. This work was supported in part by the National Science Foundation under grants NCR-911618 and NCR-9206908, and the Defense Advanced Research Projects Agency under contract NAG2-578.

References

[ACBOS93] Alvarez-Cuevas F, Bertran M, Oller F, Selga JM (1993) Voice synchronization in packet switching networks. IEEE Networks Mag 7(5):20–25

[Ber87] Bertsekas DP (1987) Dynamic Programming: Deterministic and Stochastic Models. Prentice-Hall, Englewood Cliffs, NJ 07632, USA

[Bol93] Bolot J-C (1993) End-to-end packet delay and loss behavior in the Internet. In: Proceedings of ACM SIGCOMM '93, San Francisco, Calif., pp 289–298

[CD92] Casner S, Deering S (1992) First IETF Internet Audiocast. ACM Comput Commun Rev 22(3):92–97

[Coh77] Cohen D (1977) Issues in transnet packetized voice communication. In: Proc. Fifth Data Communications Symposium, Snowbird, Ut., 6.10–6.13

[ITU93] Telecommunication Standardization Sector of ITU (1993) ITU-T Recommendation G.114. Technical report, International Telecommunication Union

[Jac94] Jacobson V (1994) Tutorial notes: Multimedia conferencing on the Internet. In: Proceedings of ACM SIGCOMM '94, London, UK

[Jay80] Jayant NS (1980) Effects of packet loss on waveform coded speech. In: Proc. Fifth Int. Conference on Computer Communications, Atlanta, Ga., pp 275–280

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

[KKT96] Kasera S, Kurose J, Towsley D (1996) Exploring the dynamic behaviour of the internet using ip options. Technical Report 96-12, Department of Computer Science, University of Massachusetts at Amherst, Amherst, MA 01003, USA

[LS83] Ljung L, Söderstrom T (1983) Theory and Practice of Recursive Identification. MIT Press, Cambridge, Mass.

[MB94] Macedonia M, Brutzman D (1994) Mbone provides audio and video across the internet. IEEE Comput Magazine 21(4):30–35

[Mon83] Montgomery WA (1983) Techniques for packet voice synchronization. IEEE J Select Areas Commun 6(1):1022–1028

[RKTS94] Ramjee R, Kurose J, Towsley D, Schulzrinne H (1994) Adaptive playout mechanisms for packetized audio applications in wide-area networks. In: Proceedings of IEEE INFOCOM '94 Montreal, Canada

[SCFJ95] Schulzrinne H, Casner S, Frederick R, Jacobson V (1996) RTP: A transport protocol for real-time applications. Request for Comments: 1889 Internet Engineering Task Force, Audio-Video Transport WG

[Sch92] Schulzrinne H (1992) Voice communication across the Internet: A Network Voice Terminal. Technical report, Dept. of ECE, Dept. of CS, University of Massachusetts, Amherst, MA 01003, USA

[SGAJ93] Sanghi D, Gudmundsson O, Agrawala A, Jain BN (1993) Experimental assessment of end-to-end behavior on Internet. In: Proceedings of IEEE INFOCOM '93,4 San Francisco, Calif., pp 867–87

[SPDB95] Shenker S, Partridge C, Davie B, Breslau L (1995) Specification of predictive quality of service. INTERNET-DRAFT draft-ietf-intserv-predictive-svc-01, Internet Engineering Task Force

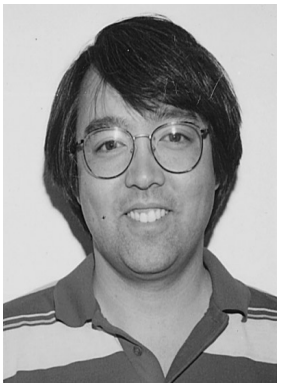
[SPW95] Shenker S, Partridge C, Wroclawski J (1995) Specification of controlled delay quality of service. INTERNET-DRAFT draft-ietf-intserv-control-del-svc-02, Internet Engineering Task Force

[WF83] Weinstein C Forgie JW (1983) Experience with speech communication in packet networks. IEEE J Select Areas Commun 6(1):963–980

[Wro95] Wroclawski J (1995) Specification of the controlled-load network element service. INTERNET-DRAFT draft-ietf-intserv-ctrl-load-svc-01, Internet Engineering Task Force

[YKT] Yajnik M, Kurose J, Towsley D: Packet loss correlation in the Mbone multicast network. In: Proceedings of IEEE Global Internet Mini-Conference, Part of GLOBECOM '96, London, England, pp 94–99

SUE B. MOON received the B.S. and M.S. from Seoul National University, Seoul, Korea, in 1988 and 1990, respectively, all in computer engineering. From 1990 to 1991, she worked for IMIGE Systems, Inc. in Seoul, Korea. Since 1992, she has been a PhD student in the Dept. of Computer Science at the University of Massachusetts at Amherst. Her research interests include multimedia communications and systems, and high-speed networks. She is a student member of ACM and IEEE.



JIM KUROSE received a B.A. degree in physics from Wesleyan University in 1978 and his M.S. and Ph.D. degrees in computer science from Columbia University in 1980 and 1984, respectively. In 1984, he joined the Computer Science faculty at the University of Massachusetts, where he is currently a Professor. He was a Visiting Scientist is the Communications Department at IBM Research during the 1990/91 academic year. Dr. Kurose's current research interests include real-time and multimedia communication, network protocols for parallel machines, operating systems, and modeling and performance evaluation. Dr. Kurose is the past Editor-in-Chief of the IEEE Transactions on Communications and of the IEEE/ACM Transactions on Networking. He has been active in the program committees for both the IEEE Infocom conference the ACM SIGCOMM conference for a number of years. He has also won a number of awards for his teaching, including outstanding teaching awards from the National Technological University (NTU), the College of Natural Science and Mathematics at the University of Massachusetts, and the Northeast Association of Graduate Schools. He is a member of the IEEE, ACM, Phi Beta Kappa, Eta Kappa Nu, and Sigma Xi.

DON TOWSLEY received the B.A. degree in physics and the Ph.D. degree in Computer Science from University of Texas in 1971 and 1975, respectively. From 1976 to 1985 he was a member of the faculty of the Department of Electrical and Computer Engineering at the University of Massachusetts, Amherst. He is currently a Professor of Computer Science at the University of Massachusetts. During 1982-1983, he was a Visiting Scientist at the IBM T.J. Watson Research Center, Yorktown Heights, NY. and during the year 1989-1990, he was a Visiting Professor at the Laboratoire MASI, Paris, France. His research interests include high-speed networks and multimedia systems. He is currently on the editorial boards of Networks and Performance Evaluation and was previously on the editorial boards of IEEE Transactions on Communications, IEEE/ACM Transactions on Networking and Journal of Discrete Event Dynamic Systems. He was a Program Co-chair of the joint ACM SIGMETRICS and PERFORMANCE '92 conference. He was elected Fellow of the IEEE for "contributions in the modeling and analysis of computer networks". He is also a member of the ACM, ORSA and the IFIP Working Groups 6.3 and 7.3.