# Adaptive Playout Mechanisms for Packetized Audio Applications in Wide-Area Networks*

Ramachandran Ramjee, Jim Kurose, Don Towsley
Department of Computer Science
Univ. of Massachusetts
{ramjee,kurose,towsley}@cs.umass.edu

Henning Schulzrinne
GMD-Fokus
Berlin, Germany
hgs@fokus.gmd.de

## Abstract

*Recent interest in supporting packet-audio applications over wide area networks has been fueled by the availability of low-cost, toll-quality workstation audio and the demonstration that limited amounts of interactive audio can be supported by today's Internet. In such applications, received audio packets are buffered, and their playout delayed at the destination host in order to compensate for the variable network delays. In this paper we investigate the performance of four different algorithms for adaptively adjusting the playout delay of audio packets in an interactive packet-audio terminal application, in the face of such varying network delays. We evaluate the playout algorithms using experimentally-obtained delay measurements of audio traffic between several different Internet sites. Our results indicate that an adaptive algorithm which explicitly adjusts to the sharp, spike-like increases in packet delay which we observed in our traces can achieve a lower rate of lost packets for both a given average playout delay and a given maximum buffer size.*

## 1 Introduction

During the past several years, there has been a significant increase in interest in the use of packetized audio over wide-area, packet-switched networks. For example, live audio from several sessions of the March 1992 Internet Engineering Task Force (IETF) meeting were "audiocast" over the Internet to participants at 20 sites [3]. More recently, sessions from the IETF meeting in Amsterdam were audiocast (and videocast) to over 500 users, using a semi-permanent multicast testbed, known as the mbone, layered over the Internet. Much of this interest has been fueled by the availability of low-cost, toll-quality audio on workstations and the realization (via demonstration) that limited amounts of interactive audio can indeed be supported by today's Internet.

From an application standpoint, a principal challenge in supporting interactive audio over a wide-area network is the need to provide synchronous playout of audio packets in the face of stochastic end-to-end network delays. This is typically achieved by buffering received audio packets (i.e., delaying their playouts) for enough time so that "most" of the packets will have been received before their scheduled playout times. This additional artificial delay until playout can either be fixed throughout the duration of an audio call, or may vary adaptively during a call's lifetime [4, 9, 14]. Packets which are not received before their scheduled playout time are considered lost. Packet loss rates of between 1 and 10% can be tolerated, depending on the manner in which voice is coded and missing packets are masked [7].

In this paper we investigate the performance of four different algorithms for adaptively adjusting the playout delay of audio packets in an interactive packet-audio terminal application, in the face of such varying network delays. This work is motivated in part by recent Internet delay measurements [2, 12] indicating that end-to-end delays may fluctuate rapidly and significantly over small intervals of time. In comparing these algorithms, our results indicate that those adaptive algorithms which adjust rapidly to these changing delays can achieve a lower rate of lost packets (due to late packet arrivals) for both a given average playout delay and a given maximum buffer size. We evaluate these playout algorithms using experimentally-obtained delay measurements of audio traffic between several different Internet sites. We note that we do not consider here the network-level issue of how to control these delays, but rather the host-level issue of how to adaptively respond to the variable delays incurred as packets traverse the network.

The remainder of this paper is structured as follows. In section 2 we describe the problem of determining playout delays for packetized voice and survey past research in this area. In section 3, we describe the four adaptive playout delay adjustment algorithms considered in this paper. In section 4 we describe the manner in which the audio delay traces were obtained and how these traces were then used to evaluate the four algorithms. We then quantitatively compare the performance of these four algorithms and discuss our results. Section 5 concludes the paper.

## 2 Background

The use of wide-area, packet-switched networks to carry packetized voice was an active research area in the late 70's and early 80's [4, 7, 9, 14] and has again recently seen a surge in activity, perhaps as a result of recent demonstrations of carrying such traffic over the

*Supported in part by the National Science Foundation under grant NCR-911618.

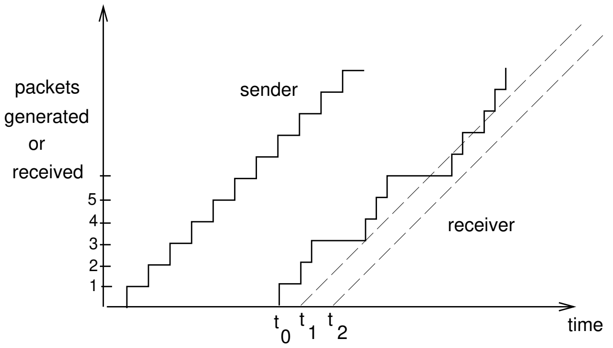


Figure 1: Generation and reconstruction of packetized voice

Internet [3, 13]. Throughout this time, the underlying model and assumptions regarding the generation and (to a somewhat lesser extent) the playout of packetized voice have remained unchanged.

Figure 1, adapted from [4], shows the operation of the sending and receiving hosts while taking part in an audio session. At the sender, packets are periodically generated as a result of the periodic sampling of an audio source. When the audio source is active (i.e., sound is being produced) packets containing the audio samples are generated and sent into the network. The staircase nature of the sender in Figure 1 indicates that packets are being generated periodically at the source. In the NeVoT audio terminal program [13] used in our Internet experiments, one 160 byte audio packet is generated approximately every 20 milliseconds when the speaker is in talkspurt. The average talkspurt length is typically on the order of several 100's of milliseconds, although the lengths can vary with different silence detection thresholds.

Packets incur random delays while traversing the network. This is illustrated by the decidedly non-staircase nature of the number of received packets as a function of time in Figure 1. In order to smooth out such delay jitter, a receiving host can delay the initiation of periodic playout of received packets for some time. For example, in Figure 1, if the receiver delays the beginning of playout until $t_2$, all packets will have been received by the time their playout is scheduled. The 45 degree line emanating from $t_2$ indicates the playout time of packet $i$ under a periodic playout strategy which begins playout at $t_2$. On the other hand, if the playout delay begins at $t_1$, there is a shorter playout delay, but packets 6, 7, and 8 will be lost at the receiver, having arrived *after* their scheduled playout time. This illustrates the tradeoff between the delay that an audio application is willing to tolerate and the packet loss suffered as a result of the late arrival of packets.

Figure 1 shows a playout strategy in which the playout delay is *fixed.* If both the propagation delay and the distribution of the variable component of network delay are known, a fixed playout delay can be computed such that no more than a given fraction of arriving packets are lost due to late arrival. This approach is advocated in [1], where this playout delay is fixed either for the length of the audio call [9, 4], or is recalculated at the beginning of each talkspurt. One potential problem with this approach is that the propagation delay is not known (although it can be estimated and typically remains fixed throughout the duration of the audio call). A more serious concern is that the end-to-end delay distribution of packets within a talkspurt is not known, and can change over relatively short time scales [2].

An approach to dealing with the unknown nature of the delay distribution is to estimate these delays and adaptively respond to their change by dynamically adjusting the playout delay. In the following section, we define four receiver-based algorithms for performing such delay estimation and dynamic playout delay adaptation. As we will see, these algorithms determine a playout delay on a per-talkspurt basis. Within a talkspurt, packets are played out in a periodic manner, thus reproducing their periodic generation at the source. However, the algorithms may change the playout delay from one talkspurt to the next, and thus the silence periods between two talkspurts at the receiver may be artificially elongated or compressed (with respect to the original length of the corresponding silence period at the sender). Compression or expansion of silence by a small amount is not noticeable in the played-out speech [9].

## 3 Four adaptive playout delay adjustment algorithms

In this section we define four adaptive playout delay adjustment algorithms. In describing these algorithms, the notation in Figure 2 will be useful. Figure 2 shows the various times associated with the sending and receiving of packet $i$ within an audio call. We make no assumptions here about the synchronization of the host sender and receiver clocks.

We introduce the following times associated with packet $i$:

$t_i$ : the time at which packet $i$ is generated at the sending host,

$a_i$ : the time at which packet $i$ is received at the receiving host,

$p_i$ : the time at which packet $i$ is played out at the receiving host,

$D_{prop}$ : the propagation delay from the sender to the receiver, which is assumed to be constant throughout the lifetime of an audio connection,

$v_i$ : the queueing delay experienced by packet $i$ as it is sent from the source to the destination host,

$b_i$ : the amount of time that packet $i$ spends in the buffer at the receiver, awaiting its scheduled playout time, $b_i = p_i - a_i$,

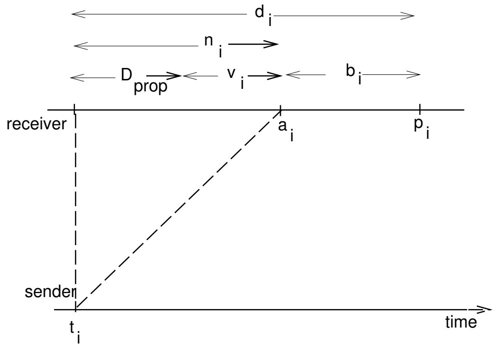


Figure 2: Timings associated with packet $i$

$d_i$ : the amount of time from when the $i$th packet is generated by the source until it is played out at the destination host, $d_i = p_i - t_i$ (this will be referred to as the "playout delay" of packet $i$),

$n_i$ : the total "delay" introduced by the network, $n_i = a_i - t_i$. Because we have not assumed that the sender and receiver clocks are synchronized, $n_i$ may not be equal to the actual delay experienced by the packet ).

The idea behind the four playout algorithms described in this paper is simple; all follow the so-called *absolute timing method* as defined by Montgomery [9]. In determining the playout point for packet $i$, we consider two cases, depending on whether or not it is the first packet in a talkspurt:

- If packet $i$ is the first packet of a talkspurt, its playout time, $p_i$ is computed as:

$$p_i = t_i + \hat{d}_i + 4 * \hat{v}_i, \qquad (1)$$

  where $\hat{d}_i$ and $\hat{v}_i$ are estimates of the mean and variation in the end-to-end delay during the talkspurt.

- The playout point for any subsequent packet in a talkspurt is computed as an offset from the point in time when the first packet in that talkspurt was played out. If $i$ was the first packet in a talkspurt and packet $j$ belongs to this talkspurt, the playout point for $j$ is computed as:

$$p_j = p_i + t_j - t_i. \qquad (2)$$

We note that $\hat{d}_i$ and $\hat{v}_i$ are computed for *every* packet received, although they are only used to determine the playout point for the first packet in any talkspurt. The four algorithms described in section 3.1 through 3.4 differ only in the manner in which $\hat{d}_i$ is computed. The computation of $\hat{v}_i$ (which in turn depends on $\hat{d}_i$) is the

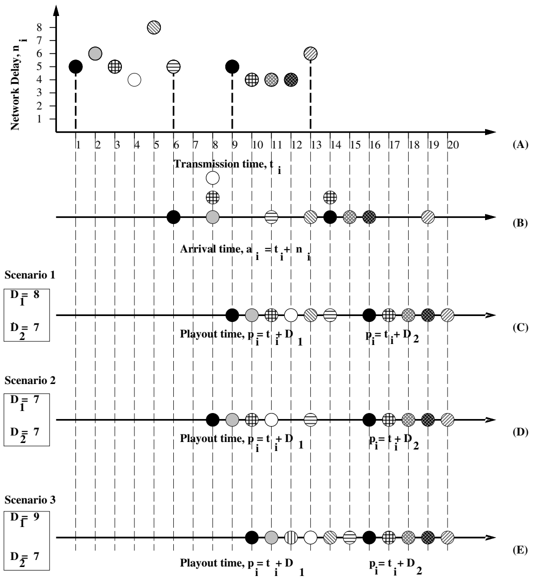


Figure 3: Example illustrating playout mechanisms

same for all the algorithms, and is defined in equation 3. From an intuitive standpoint, the $4 * \hat{v}_i$ term is used to set the playout time to be "far enough" beyond the delay estimate so that only a small fraction of the arriving packets should be lost due to late arrival. A discussion of this variation measure and standard measures of variation, such as standard deviation, can be found in [6].

The playout mechanism is further illustrated in Figure 3. The graph at the top of the figure, labeled A, represents the network delay, $n_i$ (on the y-axis), experienced by packet $i$ transmitted at time $t_i$ (note that the unit of time on the x-axis is the inter-packetization interval, which is 20 ms in the case of our audio experiments). Two talkspurts are shown in the figure, one starting at $t_1 = 1$ and another starting at $t_7 = 9$. The time axis labeled B shows the arrival pattern of the packets at the receiver. For example, packets 2, 3 and 4, shown on top of each other, arrive almost simultaneously at $a_i = 8$, as they experience different network delays. The remaining three axes illustrate the playout behavior for three possible delay adaptation scenarios.

The axis labeled C computes the playout delay for talkspurt 1 to be 8 units and thus schedules the playing of packet 1 at time $p_1 = 9$ units. The remaining packets in talkspurt 1 are scheduled one after another, in the order in which they were generated. In this example, it is then determined that the playout delay for the second talkspurt should be be 7 units. Recall that this playout delay for the packet at the beginning of every talkspurt depends on the $\hat{d}_i$ and the $\hat{v}_i$ which are computed for every packet seen so far and which in turn depend on the delay adaptation algorithm used.

The axis labeled D illustrates a second possible

$$\boxed{\begin{array}{l}\text{if } (n_i > \hat{d}_i) \text{ then} \\ \quad \hat{d}_i = \beta * \hat{d}_i + (1-\beta) * n_i \\ \text{else} \\ \quad \hat{d}_i = \alpha * \hat{d}_i + (1-\alpha) * n_i \end{array}}$$

Figure 4: **Algorithm 4**

playout scenario, in which playout delay for the first talkspurt is determined to be 7 units. Note that this leads to the dropping of packet #5, as it doesn't arrive at the receiver until after its scheduled playout time. The axis labeled E shows yet another scenario in which the playout delay for talkspurt 1 is determined to be 9 units.

It is important to note how the silence period between the two talkspurts differs in these scenarios. In scenario 1, the silence period is one unit of time shorter than what was generated by the audio source; in the third scenario, the silence period is completely eliminated. From Figure 3, it is also clear that if we set the playout delay of a talkspurt to be greater than or equal to the maximum network delay experienced by any packet in that talkspurt, we would not have any late packet loss. Of course this value in not known *a priori*, although one could possible set a playout delay to a large enough value to ensure that a significant percentage of packets would not be lost. On the other hand, setting the playout delay to a high value leads to longer delays between the transmission and the playout of the audio packets; long delays are intolerable with interactive audio [7]. Thus we desire a playout adaptation mechanism which has low loss rate as well as low playout delay.

We now describe the four delay adaptation algorithms considered in this paper.

### 3.1 Algorithm 1

In our first algorithm, the delay estimate for the $i$th packet is calculated based on the RFC793 algorithm [11] and a measure of the variation in the delays is calculated as suggested by Van Jacobson [6] in the calculation of round-trip-time estimates for the TCP retransmit timer. Specifically, the delay estimate for packet $i$ is computed as

$$\hat{d}_i = \alpha * \hat{d}_{i-1} + (1-\alpha) * n_i$$

and the variation is computed as

$$\hat{v}_i = \alpha \hat{v}_{i-1} + (1-\alpha)|\hat{d}_i - n_1| \qquad (3)$$

This algorithm is basically a linear recursive filter and is characterized by the weighting factor $\alpha$. In our experimental studies, the value of $\alpha$ was chosen to be 0.998002 to conform with the existing implementation of this algorithm in the distribution of NeVoT 1.4 [13].

### 3.2 Algorithm 2

The second algorithm is a small modification to the first algorithm, based on a suggestion by Mills [8] which would allow the TCP retransmission timer estimate to adapt more quickly to short burst of packets incurring long delays. The idea is to use a different weighting mechanism by choosing two values of the weighting factor, one for increasing trends in the delay and one for decreasing trends. The algorithm is given in Figure 4.

In our studies, $\beta$ was chosen as 0.75 and $\alpha$ was 0.998002 as before. The delay variation estimate remains the same as in Algorithm 1.

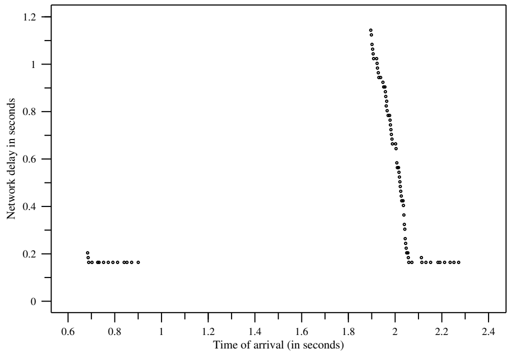


Figure 5: A typical spike

### 3.3 Algorithm 3

This algorithm is the delay adaptation algorithm currently used in NeVoT 1.6. Let $\mathcal{S}_i$ be the set of all packets received during the talkspurt prior to the one initiated by $i$. The delay estimate is computed as

$$\hat{d}_i = min_{j \in \mathcal{S}_i}\{n_j\}$$

### 3.4 Algorithm 4

In the course of our work, we examined a series of traces of the observed network delay of audio packets transmitted point-to-point between several Internet sites. In these traces, we noted frequent occurrences of delay *spikes*. The occurrence of spikes have been reported by Mills and Bolot[8, 2]. Figure 5, adapted from one of these traces, depicts a typical spike. Each point shown represents a packet arriving at the time indicated by its x-axis value, having experienced an end-to-end network delay ($n_i$) equal to the y-axis value.

A *spike* constitutes a sudden, large increase in the end-to-end network delay ($n_i$ in Figure 2), followed by a series of packets arriving almost simultaneously, leading to the completion of the spike. In Figure 5 notice how after the sudden increase in delay, approximately 50 packets arrive within a time span of 200 milliseconds. (Under normal conditions, we would expect a packet approximately every 20 milliseconds or around 10 packets over a period of 200 milliseconds).

The first three delay adaptation algorithms (including Algorithm 1) do not adapt fast enough to such spikes - taking too long to increase their delay estimate on detection of a spike and too long again to decrease their estimate once the spike is over. Algorithm 4, on the other hand does adapt to such spikes.

### 3.4.1 Design

A spike is characterized by a sudden large increase in the delay. Hence, detection of the beginning of a spike is simple - we need only check if the delay between consecutive packets at the receiver is large enough for it to be called a spike. i.e.,

$$if \quad (abs(n_i - n_{i-1}) > spike\_threshold) \quad mode = IMPULSE;$$

The actual formula we used for computing *spike_threshold* is given in the pseudocode for Algorithm 4 in Figure 6.

Once we enter the impulse mode on detection of a spike, it seems natural for us to "follow" the spike. Thus, in impulse mode, we allow our estimate to be dictated only by the most recently observed delay values. Specifically,

$$if \quad (mode == IMPULSE) \quad \hat{d}_i = \hat{d}_{i-1} + n_i - n_{i-1};$$

The detection of the completion of a spike is a bit tricky. For example, we observed that in certain cases the delay on completion of the spike was different from the delay before the beginning of the spike. Nonetheless, one prominent characteristic (see Figure 5) was that a series of packets would arrive one after another almost simultaneously at the receiver, and almost immediately following the observed increase (upward spike) in delay. Since the packets within a talkspurt are transmitted at regular intervals at the sender, near simultaneous arrivals implies that subsequent packets in the burst of arrivals have experienced progressively smaller end-to-end network delays. We thus employ a variable (*var* in the pseudcode in Figure 6) with an exponentially decaying value that adjusts to the slope of spike. When this variable has a small enough value, indicating that there is no longer a significant slope, the algorithm reverts back to *normal* mode.

Recall that $n_i$ is the end-to-end network delay of packet $i$. In algorithm 4, the equation

$$var = var/2 + abs((n_i - n_{i-1})/8 + (n_i - n_{i-2})/8);$$

uses the two most recent delay observations together with the current value, $n_i$, to track the "slope" of the spike. In the case of a spike, $abs(2n_i - n_{i-1} - n_{i-2})$ is going to be non-zero and will determine whether the spike has ended or not. The full description of the algorithm is given in Figure 4. All the parameters involved in the detection of the beginning and end of a spike were chosen based on our examination of the behavior of a number of different spikes from a large set of audio traces (not reported here) that we have collected over time. In algorithm 4, all times are measured in bytes, with 20 milliseconds (the voice packetization interval) of time corresponding to 160 bytes (i.e., 160 bytes of audio data are generated at the sender in 20 ms of time).

## 3.5 Implementation

The delay adaptation algorithm is run for every packet received at the destination host. Typically, the packetization interval is around 20 ms, which implies the that the adaptation algorithm beefficient enough to execute 50 times a second while incurring low overhead.

The low pass filter computation can be optimized, as suggested in [6], by keeping scaled versions of $\hat{d}_i$ and $\hat{v}_i$ and doing the multiplication/division as shift operations. The computation of the variable *var* in Algorithm 4 can also be performed similarly as all fractions are chosen such that they are reciprocal powers of 2.

```
1. n_i = Receiver_timestamp - Sender_timestamp;
2. if (mode == NORMAL) {
     if (abs(n_i - n_{i-1}) > abs(v̂) * 2 + 800) {
       var = 0; /* Detected beginning of spike */
       mode = IMPULSE;
     }
   }
   else {
     var = var/2 + abs((2n_i - n_{i-1} - n_{i-2})/8);
     if (var <= 63) {
       mode = NORMAL; /* End of spike */
       n_{i-2} = n_{i-1};
       n_{i-1} = n_i;
       return;
     }
   }
3. if (mode == NORMAL)
     d̂_i = 0.125 * n_i + 0.875 * d̂_{i-1};
   else
     d̂_i = d̂_{i-1} + n_i - n_{i-1};
   v̂_i = 0.125 * abs(n_i - d̂_i) + 0.875 * v̂_{i-1};
4. n_{i-2} = n_{i-1};
   n_{i-1} = n_i;
   return;
```

Figure 6: **Algorithm 4**

# 4 Algorithm Comparison

## 4.1 Experimental Internet delay measurements

NeVoT [13], an audio conferencing tool, was used to collect experimental data. This tool has options for tracing various network- and audio-dependent events which occur during an audio conversation. The traces were obtained on Sun Sparcstation platforms using vat audio packet format as the audio protocol and unicast UDP as the transport protocol. As noted previously, NeVoT transmits 160 bytes of audio data, approximately every 20 milliseconds.

In all our experiments, we obtained traces both at the source and at the receiver hosts.. Thus, we had a log of when a packet left the source and when it arrived at the receiver. Our traces were obtained by repeatedly playing an audio file at the sending host, with the receiving host "listening" to the received packets. The sites chosen were University of Massachusetts-Amherst(UMASS), INRIA-France (INRIA), University of California-Irvine(UCI) and Osaka University-

Japan(OSAKA). The route between UMASS and OSAKA had 26 hops and was very lossy. We observed packet loss rates of up to 20-30%. The route between UMASS and INRIA had 27 hops but the packet loss rate was around 2-10%. The route between UMASS and UCI had 18 hops and had the lowest loss rate of around 1-4%. All our traces were taken between hosts connected point-to-point and while the host machines were relatively idle. In this paper, we present our analysis based on the traces shown in Table 4.1.

### 4.2 A comparison and discussion of results

In order to compare the four algorithms, we wrote a simulator which takes the received packet trace file generated by running NeVoT and simulates the behavior of the playout algorithms. The simulator ran the four algorithms on the same set of traces and we were thus able to compare the performance of the algorithms under identical network conditions.

One of our primary performance metrics is the percentage of packets lost *at the receiving host.* Such loss results from either the late arrival of a packet (i.e., the arrival of a packet after its scheduled playout time) or an extremely premature arrival of a packet. In the latter case, packet loss results from a limitation on the size of the (finite) audio storage buffer.

In our simulations, we adopted a circular buffering scheme (the one implemented in NeVoT) in which packets are played out *sequentially* and *periodically* from the buffer. When a packet arrives, its playout time is computed (according to one of the algorithms described in sections 3.1 - 3.4) and the packet is placed in the appropriate location in the playout buffer. Thus, if the $i$th packet is currently being played out and the packet buffer can hold up to $k$ audio packets, any packet arriving at this time with a number of $i + k$ or greater will be discarded, having arrived too far *in advance* of its playout time to be buffered. We study the effect that the length of this audio playout buffer has on the performance of the four playout delay adaptation algorithms.

Figures 7 to 13, compare the performance of the four algorithms for the traces shown in Table 4.1. Let us first consider Figure 7. The upper graph plots the percentage packet loss versus the average playout delay of successfully played-out packets. The average playout delay was computed by taking the average of the $d_i$ values of the successfully played-out packets and subtracting the smallest value of $d_i$ in the entire trace. (The subtraction of this constant value was done to give an idea of the average delay incurred by packets as a result variable network delay and buffering at the receiving host – delays beyond the immutable propagation delay). Each curve in the upper plot is parameterized by a different maximum buffer size. That is, for a given maximum buffer size and the given trace (trace # 1 in this case), each algorithm results in a certain packet loss and a certain average playout delay; these points are plotted in the upper graph in Figure 7. As indicated in the lower graph in Figure 7 this maximum buffer size ranged from 160 bytes to 4K bytes.

The upper graph in figure 7 indicates that, for this trace, Algorithm 4 performs slightly better than Algorithms 1 and 3 in the range of loss rates of interest (less than 10% loss). Algorithm 2 performs quite poorly with this trace, as with other traces as well. Although this algorithm may be effective in reducing the number of superfluous retransmissions in the case of TCP, it produces long playout delays in the packetized audio environment with no appreciable reduction in losses. This is further evidence that what is good for one domain need not necessarily be good in another domain, even though the idea behind both (estimating delays) is the same.

The lower graph in figure 7 plots packet loss as a function of the maximum buffer size. By this comparison, Algorithm 4 again performs slightly better than Algorithms 1 and 3 in the regions of interest.

Figures 8 though 13 plot loss versus average delay, for the remaining traces shown in Table 4.1. Although we also have results on loss versus buffer size, for these traces, we do not present them here for lack of space. Generally, we note that Algorithm 4 performs as well as, or much better than, the other three algorithms. In Figures 8 and 9, Algorithms 3 and 4 perform a little better than Algorithm 1. In Figures 12 and 13, Algorithm 4 outperforms Algorithms 1 and 3 by a significant margin. For example, in Figure 13, the average playout delay is about 300 milliseconds less *for the same loss rate* under Algorithm 4 than under the other playout strategies. Under traces 4 and 5 (Figures 10 and 11), Algorithms 3 and 4 perform slightly worse then Algorithm 1, although a difference in average playout delay if less than 10 milliseconds is not that significant. The reason that algorithm 4 performs much better under traces 6 and 7 and not as well under traces 4 and 5 can be understood if we look at Figures 14 and 15. The jitter in the delays in the former is extremely large while almost negligible in the latter. Algorithm 4 has a significant performance advantage in the case of large delay jitters (i.e., the spike-like delay behavior shown in Figures 5 and 14) as a result of its spike detection/adaptation algorithm.

A direct comparison between Algorithms 1 and 3 is more difficult; in some cases, Algorithm 3 performs a little better and in some cases Algorithm 1 does better. A point in favor of Algorithm 3 is the minimal computation involved. Note that Algorithm 3 doesn't fare well in cases where the jitter is high. For example, under trace 7 (Figure 13), the average playout delay under Algorithm 3 is even worse than the that under Algorithm 2 for the same loss rate. This indicates that taking the minimum of delay values of packets in the previous talkspurt may not be a robust delay estimate when delay varies significantly from talkspurt to talkspurt.

## 5 Conclusions

In this paper we have investigated the performance of four different algorithms for adaptively adjusting the playout delay for audio packets in the face of varying network delays. We evaluated these playout algorithms using experimentally-obtained delay measurements of audio traffic between several different In-

| Trace # | Sender | Receiver | Start time(Receiver) | Day | Duration of trace |
|---|---|---|---|---|---|
| 1. | INRIA | UMASS | 09:20 P.M. Aug 26 | Thursday | 580 secs |
| 2. | INRIA | UMASS | 10:35 A.M. Aug 27 | Friday | 302 secs |
| 3. | UCI | UMASS | 08:28 P.M. Aug 24 | Tuesday | 132 secs |
| 4. | UCI | UMASS | 01:20 P.M. Aug 25 | Wednesday | 634 secs |
| 5. | UCI | INRIA | 09:00 P.M. Sep 18 | Saturday | 1091 secs |
| 6. | OSAKA | UMASS | 07:48 A.M. Sep 08 | Wednesday | 125 secs |
| 7. | OSAKA | UMASS | 00:35 A.M. Sep 24 | Friday | 649 secs |

Table 1: Trace Details

ternet sites. In comparing these algorithms, our results indicate that an adaptive algorithm (Algorithm 4) which explicitly adjusts to the sharp, spike-like increases in packet delay which we observed in our traces can achieve a lower rate of lost packets for both a given average playout delay and a given maximum buffer size.

We have compared these algorithms from the perspective of number of packets dropped. We are currently investigating possibilities for quantifying the distortion which occurs when silence periods are artificially contracted or expanded, and comparing the algorithms based on this factor. Delay adaptation in the face of varying network delays is crucial in determining the audio quality at the receiver; additional research is needed in formalizing and arriving at a better understanding of the dynamics of network delays.

Acknowledgment

The authors also acknowledge Tatsuya Suda (of UC Irvine), Zhen Liu (of INRIA, Sophia Antipolis, France), and Hiromi Okada and Miki Yamamoto (of Osaka University) for providing host machines which made the Internet voice measurements possible. Thanks also to Maya Yajnik for help with NeVoT and many illuminating discussions about packet voice

## References

[1] F. Alvarez-Cuevas, M. Bertran, F. Oller and J. Selga, "Voice Synchronization in Packet Switching Networks," *IEEE Networks Magazine,* Vol. 7, No. 5 (Sept., 1993), pp. 20 - 25.

[2] J.C. Bolot, "End-to-End Packet Delay and Loss Behavior in the Internet," *Proc. 1993 ACM SIGCOMM Conf.,* (Sept. 1993, San Francisco), pp. 289-298.

[3] S. Casner and S. Deering, "First IETF Internet Audiocast," *ACM Computer Communication Review,* Vol. 22, No. 3 (July 1992), pp. 92 - 97.

[4] D. Cohen, "Issues in Transnet Packetized Voice Communication," *Proc. Fifth Data Communications Symposium,* (Sept., 1977, Snowbird UT), pp. 6.10 - 6.13.

[5] John. G. Gruber and Nguyen H. LE, "Performance requirements for integrated voice/data networks," *IEEE J. on Selected Areas in Communications,* Sol. SAC-6, No. 1 (Dec. 1983), pp. 981 - 1005.

[6] V. Jacobson, "Congestion avoidance and control," *Proc. 1988 ACM SIGCOMM Conf.,*(Aug. 1988, Stanford), pp. 314-329.

[7] N. Jayant, "Effects of Packet Loss on Waveform Coded Speech," *Proc. Fifth Int. Conference on Computer Communications,* (Oct. 1980, Atlanta GA), pp. 275 - 280.

[8] D. Mills, "Internet Delay experiments," *ARPANET Working Group Requests for Comment,* (December 1983), RFC 889.

[9] W. Montgomery, "Techniques for Packet Voice Synchronization," *IEEE J. on Selected Areas in Communications,* Sol. SAC-6, No. 1 (Dec. 1983), pp. 1022 - 1028.

[10] A. Mukherjee, "On the Dynamics and Significance of the Low Frequency Components of the Internet Load," TR CIS-92-83, University of Penn., Philadelphia, PA, December 1992.

[11] Jon Postel, editor, "Transmission Control Protocol specification," *ARPANET Working Group Requests for Comment,* (September 1981), RFC 793.

[12] D. Sanghi, O. Gudmundsson, A. Agrawala, "Experimental Assessment of of end-to-end Behavior on Internet," *Proc. IEEE Infocom93,* (Apr. 1993, San Francisco, CA), pp. 867 - 874.

[13] H. Schulzrinne, "Voice Communication Across the Internet: a Network Voice Terminal," Technical Report, Dept. of Computer Science, U. Massachusetts, Amherst MA, July 1992. (available via anonymous ftp to gaia.cs.umass.edu in pub/nevot/nevot.ps.Z)

[14] C. Weinstein and J.W. Forgie, "Experience with Speech Communication in Packet Networks," *IEEE J. on Selected Areas in Communications,* Sol. SAC-6, No. 1 (Dec. 1983), pp. 963 - 980.

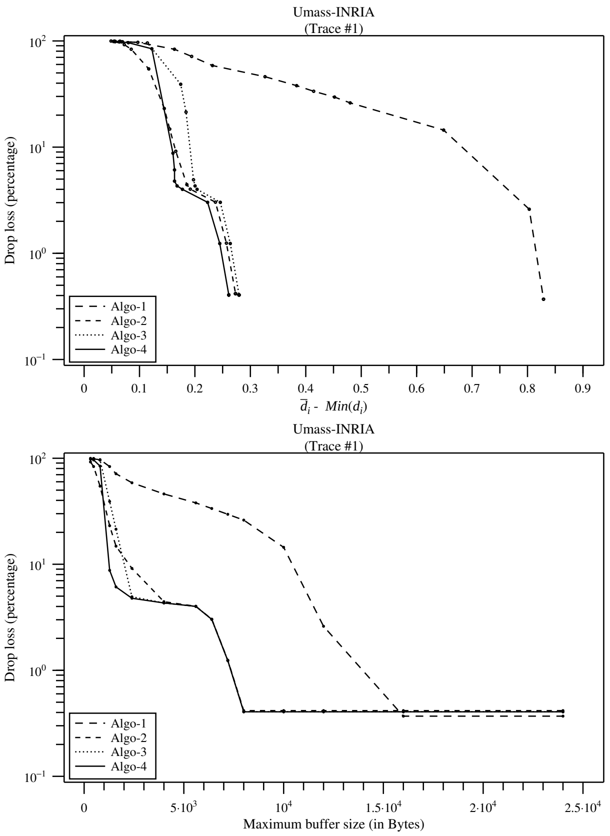


Figure 7: Comparison using trace 1 (UMASS-INRIA)

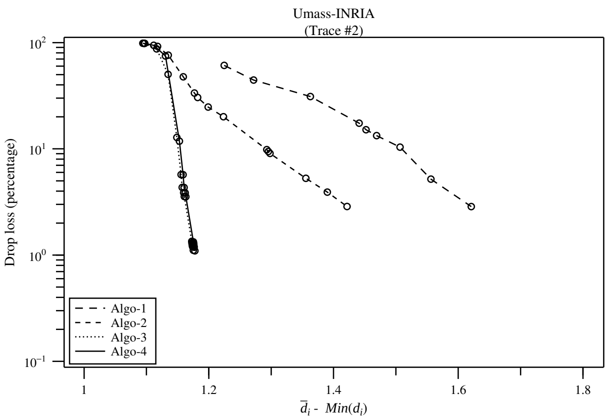


Figure 8: Comparison using trace 2 (UMASS-INRIA)

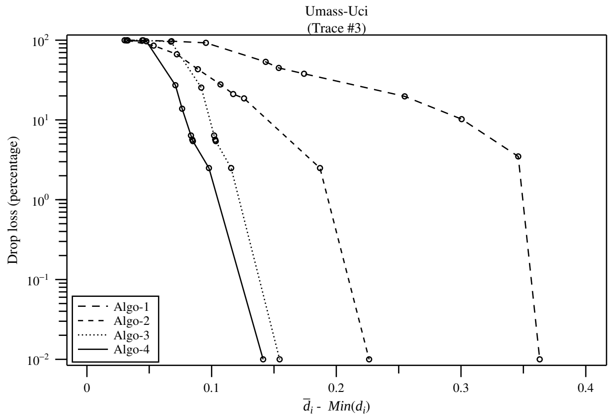


Figure 9: Comparison using trace 3 (UMASS-UCI)

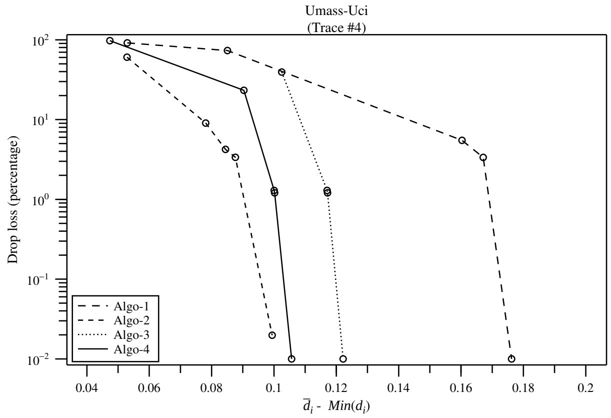


Figure 10: Comparison using trace 4 (UMASS-UCI)

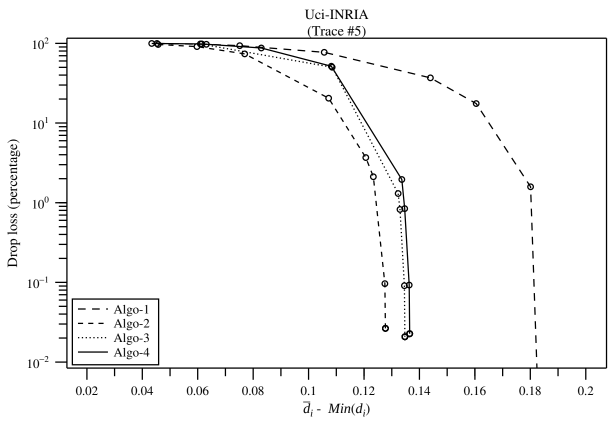


Figure 11: Comparison using trace 5 (UCI-INRIA)

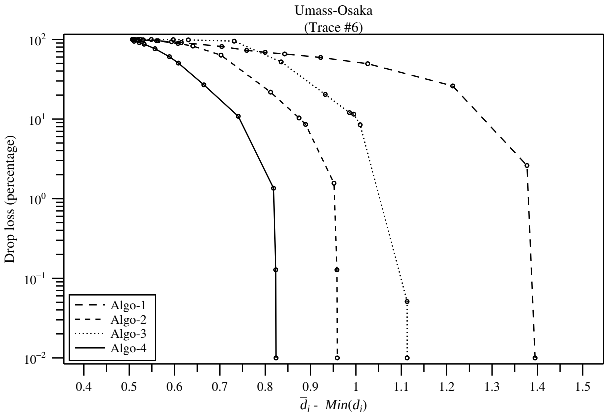


Figure 12: Comparison using trace 6 (UMASS-OSAKA)

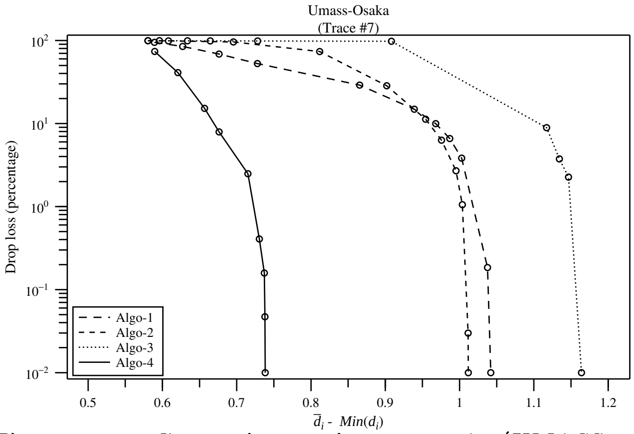


Figure 13: Comparison using trace 7 (UMASS-OSAKA)

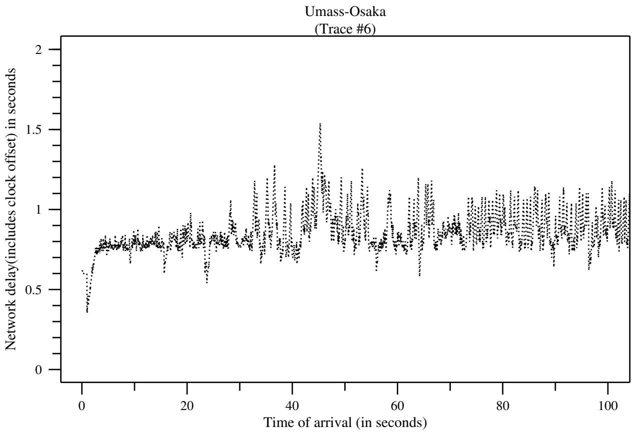


Figure 14: Delay experienced by the audio packets (UMASS-OSAKA)

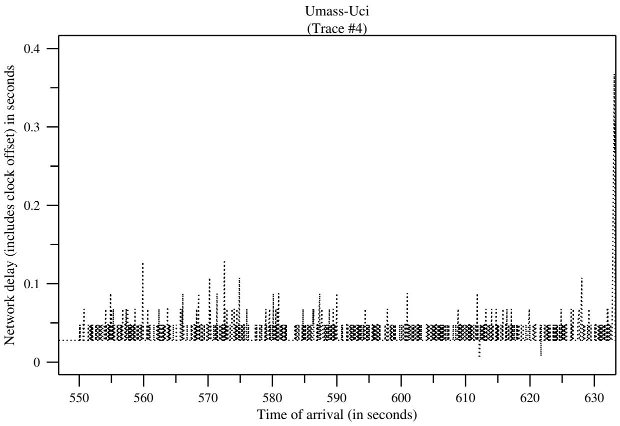


Figure 15: Delay experienced by the audio packets (UMASS-UCI)