Delay Reduction Techniques for Playout Buffering

Cormac J. Sreenan, Jyh-Cheng Chen, *Member, IEEE*, Prathima Agrawal, *Fellow, IEEE*, and B. Narendran

***Abstract*— Receiver synchronization of continuous media streams is required to deal with delay differences and variations resulting from delivery over packet networks such as the Internet. This function is commonly provided using per-stream playout buffers which introduce additional delay in order to produce a playout schedule which meets the synchronization requirements. Packets which arrive after their scheduled playout time are considered late and are discarded. In this paper, we present the Concord algorithm, which provides a delay-sensitive solution for playout buffering. It records historical information and uses it to make short-term predictions about network delay with the aim of not reacting too quickly to short-lived delay variations. This allows an application-controlled tradeoff of packet lateness against buffering delay, suitable for applications which demand low delay but can tolerate or conceal a small amount of late packets. We present a selection of results from an extensive evaluation of Concord using Internet traffic traces. We explore the use of aging techniques to improve the effectiveness of the historical information and hence the delay predictions. The results show that Concord can produce significant reductions in buffering delay and delay variations at the expense of packet lateness values of less than 1%.**

***Keywords*— multimedia stream synchronization, playout buffering.**

I. Introduction

THE use of packet networks for delivering continuous media is set to become more and more popular, driven by user demand and network technology advances providing quality of service (QoS) and increased bandwidth to the user terminal. When such networks transport packetized streams of continuous media, receiver synchronization is required. *Single stream* synchronization aims to smooth the transmission delay variations (often called *jitter*) between packet arrivals for a given stream. This problem can be solved at the receiver by using per-stream playout buffers as illustrated in Figure 1. These buffers introduce additional delay with the aim of producing a playout schedule which meets the synchronization requirements. In terms of operation, playout buffers act as a holding area for packets whose scheduled playout time is in the future. Packets which arrive after their scheduled playout time are considered late and are discarded.

Common algorithms for managing a playout buffer take one of two basic approaches. Fixed approaches assume that the range of delays is predictable and use a static buffer size and schedule. Reactive approaches, which are common in the Internet, measure immediate jitter and use that to dynamically adjust the buffer size and schedule to avoid lateness. Fixed approaches have known buffering delay but potentially large packet lateness, while reactive approaches avoid lateness but at the expense of potentially very high buffering delays. This paper works with an alternative predictive approach, which records historical information and uses it to make short-term predictions about network delay with the aim of not reacting too quickly to short-lived variations. This allows an application-controlled tradeoff of packet lateness against buffering delay, suitable for applications which demand low delay but can tolerate or conceal a small amount of late packets. Our solution, known as *Concord*, is notable because it defines a single framework to deal with both forms of synchronization, and operates under the influence of parameters which can be supplied by the application involved. We perform a trade-off of packet lateness against the total end-to-end delay suffered by packets, by applying these parameters to historical data describing packet delay distributions. This paper describes the Concord algorithm and presents a detailed evaluation of its performance using traces for audio and video streams having traversed the Internet. Based on Concord, we also explore the use of aging techniques to improve the effectiveness of the historical information and hence the delay predictions. Three aging algorithms are proposed and compared with a reactive approach commonly used in the Internet. The results show that Concord can produce significant reductions in buffering delay and jitter at the expense of packet lateness values of less than 1%.

The rest of the paper is organized as follows. Section II describes the Concord algorithm and the predictive approach it uses. Section III describes aging techniques as an extension to Concord to improve its effectiveness. Section IV presents results of a trace-driven simulation for Concord (without using aging) in comparison to other well-known techniques for playout buffer operation. Section V

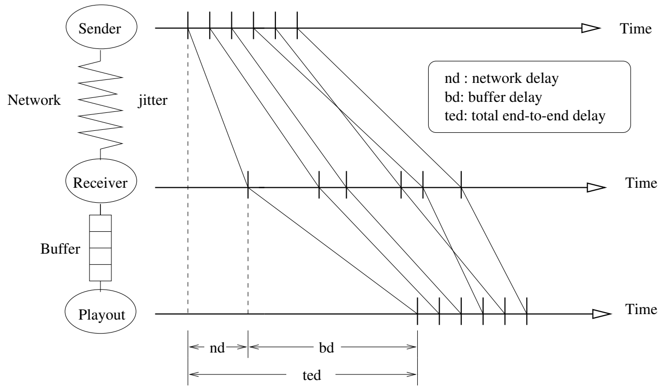

Fig. 1. Stream synchronization

C. J. Sreenan is with the Department of Computer Science, University College Cork, Ireland. E-mail: cjs@cs.ucc.ie.

J.-C. Chen is with Telcordia Technologies, Morristown, NJ 07960, USA. E-mail: jcchen@research.telcordia.com.

P. Agrawal is with Telcordia Technologies, Morristown, NJ 07960, USA. E-mail: pagrawal@research.telcordia.com.

B. Narendran is with Juno Online Services, Inc., New York, NY 10036, USA. E-mail: naren@staff.juno.com.

presents an additional set of simulation results to demonstrate the effectiveness of aging. Section VI summarizes related work, while Section VII concludes the paper. Basic features of the Concord algorithm were originally described in [1], while some initial simulation results appeared in [2]. The aging algorithms presented in this paper build upon our earlier work in [3].

II. Fundamental Features of the Concord Algorithm

This section summarizes the features of the Concord algorithm, which uses a predictive approach to playout buffer management.

A. Approach

For applications like packet telephony, large delays cause a significant negative impact on quality because they reduce interactivity. For example, in packetized voice applications, figures in the 150 - 250 millisecond range are often quoted as being the maximum acceptable total end-to-end delay. It is argued that a better approach than those commonly employed for operating a playout buffer is necessary in order to minimize end-to-end delay. There are two common approaches to operating a playout buffer:

- *Fixed:* If a packet i does not arrive within a certain fixed time bound, the packet is assumed to be lost, and subsequent packets are played back. If i does arrive later it is declared to be late and is thrown away.
- *Reactive:* In this scheme, the receiver waits until a packet i arrives to play back the packet.

The concern here is the total end-to-end delay: the cumulative delay suffered by packets in the network and the buffer. With the fixed approach, if every packet is delayed in the buffer such that it suffers a cumulative delay (in the network and buffer) equal to the maximum network delay, the receiver can reproduce a jitter-free playout. Problems with this approach are that it may be difficult to obtain an accurate estimate of maximum network delay over a stream lifetime, and such a value may be quite large in relation to the mean delay, especially in wide area networks such as the Internet. The reactive approach results in a playout buffer which grows and shrinks to accommodate delay fluctuations. This can also result in large buffering delays, often just to deal with the type of short-lived increases in network delay ("spikes") seen in the Internet.

The predictive approach used in Concord takes advantage of the built in redundancy of a multimedia stream to reduce the buffer delay at the expense of increasing the number of late packets (and hence increasing the overall packet loss rate). Many voice and video coding algorithms can still produce satisfactory output by cleverly concealing missing packets. In Concord, a trade-off is performed of packet lateness against the total end-to-end delay suffered by packets, by taking application-supplied parameters and applying them to probabilistic data describing historical packet delay distributions. The aim is not to react too quickly to short-lived network jitter.

Name	Description
PDD	Packet delay distribution
ted_s	Total end-to-end delay for stream s
mad_s	Maximum acceptable delay for stream s
bs_s	Buffer size for stream s
nd_s^i	Network delay for packet i of stream s
bd_s^i	Buffer delay for packet i of stream s
mlp_s	Maximum late packets (%) for stream s
alp	Actual late packets (%)
Cdf	Cumulative distribution function
$H_i(x)$	Function of histogram after ith aging
$P_i(x)$	PDD function after ith aging ($0 \leq P_i(x) \leq 1$)
F	Aging factor
S	Sum of bin values in histogram
c	Aging coefficient ($0 \leq c < 1$)
f	Aging frequency (in packet spacing)
r_1	Ratio of old aged data to the newly arrived packet
r_2	Ratio of old aged data to subsequent packets until next aging

TABLE I

Basic notation

B. Operation

In Concord we define a stream as a sequence of packets, produced according to some period, each marked with a sequence number. Table I defines the basic notation used throughout the paper. A stream s is characterized by the *maximum acceptable delay* (mad_s) it can suffer, and the *maximum late packets* (mlp_s) percentage it can tolerate. For each stream we construct a *packet delay distribution* (PDD) - a statistical representation of network delays for packets in that stream. The PDD is approximate, but over time it is updated dynamically so that it closely tracks the actual performance. The speed of getting this process established can be accelerated by having an initial approximation for the PDD, perhaps based on recent observations. The basic problem is then to find the minimum buffer size at the receiver that will smooth out network jitter so that the stream's requirements of mlp_s and mad_s are satisfied, if possible. That is, to find bs_s the minimum buffer size satisfying the following:

- For every packet i: $nd_s^i + bd_s^i$ is a value ted_s, where ted_s is the *total end-to-end delay*, nd_s^i is the *network delay* suffered by packet i, and bd_s^i is the induced *buffer delay* for i.
- The chosen ted_s is less than the mad_s of the stream.
- The chosen ted_s does not lead to more than mlp_s percent of packets being thrown away.

We show how to calculate the total end-to-end delay, using the PDD shown in Figure 2 as an example. This PDD shows a normal distribution of packet delays, which is common in packet switched networks, but is not a Concord prerequisite. From this plot it is clear that all packets that suffer more than ted in network delay will be declared

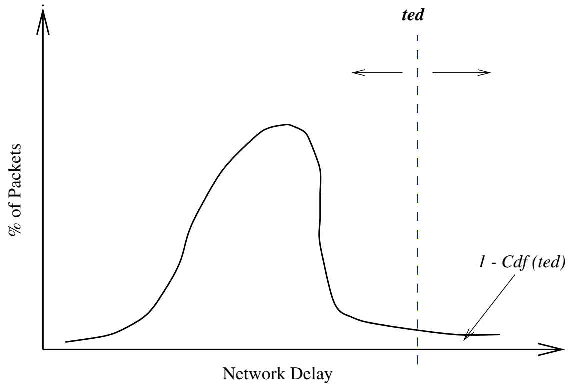

Fig. 2. Example packet delay distribution (PDD)

late. Hence $(1 - Cdf(ted))$ packets will be declared late, where Cdf is the *cumulative distribution function* on the PDD. Now we can influence Concord's behavior by choosing a value of *ted* so that either *mlp* or *ted* is minimized (best visualized by moving the *ted* line in the figure to the right or left, respectively).

III. Dynamic Control of Statistical Trends

There are several known approaches for processing statistical trends based on observed measurements. One approach is the *full aggregation* method wherein data is accumulated into a probability distribution curve throughout the lifetime of the transmission. The second approach is the *flush and refresh* approach in which statistical samples are stored for a period of time and then periodically flushed and refreshed. With the first approach, the recent information and old information are given the same weight in terms of their influence on the probability distribution. Therefore, it is slow to react to changes occurring in the system. With the second approach, the periodic flush results in a complete loss of historic information and can introduce boundary effects at the flush instances.

There is a need for a suitable statistical method which can predict the future network delay by analyzing historical and current information and monitor, maintain, update, and store statistical trends of network delays. The approach taken in Concord is to store and track network delay trends using a measured histogram to approximate the PDD curve. The histogram bins store frequency of delay patterns, where *bin width* means the range of network delays grouped together to represent one pattern in the histogram. Naturally, the number of bins increases directly as the width of the bins is decreased. Since network characteristics vary with time, these statistical trends vary, and current information is necessary for Concord to be effective, that is, it requires an update and operation of each bin to diminish the effect of older samples. This operation is called *aging*.

The next section introduces the *aging function* which ages the older samples gradually to allow an accurate delay estimate. Instead of discarding older information, it is gradually *retired*. The balance between bin width and accuracy is then discussed in a subsequent section.

A. Aging Function

The aging function is an essential element for increasing the effectiveness of the Concord algorithm. *Aging frequency* determines how often the aging function should be invoked. Users or applications can set the rate or frequency of aging. More frequent aging means the data is very current and quick to react to changes. Less frequent aging works with less current data and may react to changes slowly.

Aging may be achieved in several ways. A simplistic approach involves discarding all the data prior to a certain threshold based on time or a packet count. The threshold here is a moving window of fixed size prior to the current time. While this retains more history information than the *flush and refresh* method, it has high overheads and can be awkward to implement, requiring complete data to be kept, rather than a statistical approximation. The approach advocated here involves not discarding the older data entirely, but instead gradually reducing its effect on the statistical distribution. This can be done by periodically scaling down the existing distribution by an *aging coefficient* determined by a user or an application while continuing to add new sample data with a constant weight. This periodic process progressively lessens the influence of the older samples while giving the newer ones a larger effect. Aging coefficient is a parameter provided by the applications, while Concord translates that into a corresponding *aging factor*, which is a mathematical quantity used to scale the PDD histograms. The type of translation is what distinguishes the three different algorithms described later. The *aging coefficient* hereafter is defined as a number between 0 and 1 which is determined by an application and used to diminish the effect of older statistical data. The *aging factor* is defined as a scaling factor used by Concord to scale down the value of each bin in histogram.

Aging may be based on the number of packets received. For example aging may be applied to every packet arrival. In another example a user or an application may choose to age the statistical data every 1000 packets and increment the bin corresponding to each packet's delay by one whenever a packet arrives. There are several ways to deal with the aging factor and coefficient. Three different approaches are presented. Algorithm 1 is the basic version. Algorithm 2 is the improved version of algorithm 1. Algorithm 3 is a variant of algorithm 2.

A.1 Aging Algorithm 1:

In the first algorithm, each histogram bin is multiplied by the aging factor which is defined as:

$$F = c \quad (1)$$

where c is the aging coefficient, and F is the aging factor. *Aging factor* is the real number used to scale down each bin, and *aging coefficient* is the parameter provided by users/applications. Their relationship is identical in algorithm 1, but differs in algorithms 2 and 3. In this first algorithm, the total count is readjusted by the aging coefficient times the previous sum of all the histogram bin contents. The bin corresponding to the new packet's delay is then increased by one. After aging, all subsequent packets to arrive cause the corresponding bin to be incremented by one until aging happens again. Let $H_i(x)$ be the function of the histogram after the ith aging, nd be the network delay of new packet, thus:

$$H_i(x) = \begin{cases} F \times H_{i-1}(x), & 0 \leq x < \infty, x \neq nd \\ F \times H_{i-1}(x) + 1, & x = nd \end{cases} \tag{2}$$

The ratio of old aged data to the newly arrived packet is defined as r_1:

$$r_1 = c \times \int_0^\infty H_{i-1}(x)\,dx \tag{3}$$

The ratio of old aged data to subsequent packets until the next aging is defined as r_2:

$$r_2 = \frac{c}{f} \times \int_0^\infty H_{i-1}(x)\,dx \tag{4}$$

where f is the aging frequency in packet spacing. Let $P_i(x)$ be the PDD function after the ith aging, hence

$$P_i(x) = \begin{cases} \frac{F \times H_{i-1}(x)}{S}, & 0 \leq x < \infty, x \neq nd \\ \frac{F \times H_{i-1}(x)+1}{S}, & x = nd \end{cases} \tag{5}$$

where $S = \int_0^\infty H_i(\hat{x})\,d\hat{x}$. In practical implementations, the continual function ($\int$) can be replaced by a discrete function ($\sum$). Also, y can be limited to the upper and lower bounds of network delay which can be updated whenever every packet arrives. $H_i(x)$ is a function of the actual number of packets in the histogram. $P_i(x)$ normalizes $H_i(x)$ so that $\int_0^\infty P_i(x)\,dx = 1$. mlp must satisfy the following condition:

$$mlp \geq \int_t^\infty P_i(x)\,dx = \begin{cases} \frac{F}{S}\int_t^\infty H_{i-1}(x)\,dx, & nd < t \\ \frac{F}{S}\int_t^\infty H_{i-1}(x)\,dx \;|_{x \neq nd} + \\ \quad \frac{F \times H_{i-1}(nd)+1}{S}, & nd \geq t \end{cases} \tag{6}$$

where t is the chosen ted (fig. 2). Let c_1 and c_2 be two different aging coefficients. If we set the same mlp for both of them, the following equation can be obtained for $nd < t$ in equation (6):

$$c_1 \int_{t_1}^\infty H_{i-1}(x)\,dx = c_2 \int_{t_2}^\infty H_{i-1}(x)\,dx \tag{7}$$

Suppose $c_1 > c_2 > 0$, we then get

$$\int_{t_1}^\infty H_{i-1}(x)\,dx < \int_{t_2}^\infty H_{i-1}(x)\,dx \tag{8}$$

For the case of $nd \geq t$, we can obtain:

$$\begin{aligned} c_1 H_{i-1}(nd) + c_1 \int_{t_1}^\infty H_{i-1}(x)\,dx = \\ c_2 H_{i-1}(nd) + c_2 \int_{t_2}^\infty H_{i-1}(x)\,dx \end{aligned} \tag{9}$$

Since $c_1 > c_2 > 0$, the result of equation (8) can be derived for this case also. Therefore, we can conclude that $t_1 > t_2$ when $c_1 > c_2$ from equation (8). This means decreasing aging coefficient decreases the value of ted.

A.2 Aging Algorithm 2:

In algorithm 1, the bin values are scaled down by the aging coefficient when aging occurs. As the number of samples increase, hence the corresponding bin value increases, the packet arriving after aging contributes less to the histogram in comparison to the old aged data in histogram which have just been scaled down (see equation (3)). For example, if the total of histogram bin values is 10, this is scaled down by 0.9 after aging. The new packet then contributes one to the histogram. The ratio of old aged data to the newly arrived packet is 9. After some time, the sum of the bins in histogram may be 10000. After aging by 0.9, the ratio of old aged data to the newly arrived packet is 9000, which is different with the previous ones. This can be seen in equation (3) that r_1 is changing based on the value of old aged data in histogram.

The intention of the second algorithm is to keep the ratio of old aged data to the newly arrived packet constant through the lifetime of the synchronization stream. When aging occurs, the next packet contributes one to the histogram, but the old statistical data prior to aging are scaled down by the following factor:

$$F = \frac{c}{(1-c) \times \int_0^\infty H_{i-1}(x)\,dx} \tag{10}$$

By scaling down the old bins by equation (10), the ratio of old aged data to the newly added bin is always constant, which is represented by:

$$r_1 = \frac{c}{1-c} \tag{11}$$

Therefore, at each aging, the ratio of old aged data to the new arrival's bin is independent of the lifetime of the stream. Each arriving packet contributes the same weight which is $(1-c)$ to the new statistical data. The old aged data also contributes the same weight which equals c. Comparing to the first algorithm, the advantage is that the data may still be current and quick to react to changes if the stream has a long lifetime. The ratio of old aged data to subsequent packets until the next aging is:

$$r_2 = \frac{c}{f\,(1-c)} \tag{12}$$

Following the same derivation from equations (2), (5), and (6), we get:

$$\frac{c_1}{1-c_1}\int_{t_1}^{\infty} H_{i-1}(x)\,dx = \frac{c_2}{1-c_2}\int_{t_2}^{\infty} H_{i-1}(x)\,dx \tag{13}$$

$$\because\ 1 > c_1 > c_2 > 0,\ \therefore\ \frac{c_1}{1-c_1} - \frac{c_2}{1-c_2} = \frac{c_1 - c_2}{(1-c_1)(1-c_2)} > 0$$

Therefore, the same result in equation (8) can be obtained for the case $nd < t$. A similar procedure can be used to derive an identical result for the case of $nd \geq t$. Hence, the same conclusion in aging algorithm 1 that $t_1 > t_2$ when $c_1 > c_2$ can be obtained.

A.3 Aging Algorithm 3:

In previous algorithms, the scaling factor is related to the aging coefficient only. In the first algorithm, the bins are scaled down by the aging coefficient without considering the ratio of old aged data to a new packet. Algorithm 2 scales down each bin by the factor in equation (10) which keeps the ratio r_1 constant. The ratio, however, does not consider the aging frequency. In the third algorithm, equation (10) is modified as follows:

$$F = \frac{c \times f}{(1-c) \times \int_0^{\infty} H_{i-1}(x)\,dx} \tag{14}$$

where f is the aging frequency in *packet spacing.* By scaling down the bin values by equation (14), the ratio of old aged data to the subsequent packets until next aging is constant through the lifetime of the stream even if the aging frequency is changing dynamically. The statistical data may be aged every f packets. The goal of this algorithm is to keep the sum of all subsequent packets until next aging the same weight to the new statistical data. The ratio of old aged data to subsequent arrived packets is independent of aging frequency which equals

$$r_2 = \frac{c}{1-c} \tag{15}$$

However, the ratio of old aged data to a new packet depends on the aging frequency:

$$r_1 = \frac{c \times f}{1-c} \tag{16}$$

As that in algorithms 1 and 2, the following equation can be obtained:

$$\frac{f_1 \times c_1}{1-c_1}\int_{t_1}^{\infty} H_{i-1}(x)\,dx = \frac{f_2 \times c_2}{1-c_2}\int_{t_2}^{\infty} H_{i-1}(x)\,dx \tag{17}$$

When f is fixed, $t_1 > t_2$ if $c_1 > c_2$. Similarly, $t_1 > t_2$ if $f_1 > f_2$ when c is fixed. This indicates that more frequent aging allows a smaller value of *ted*.

This section has shown three algorithms for the aging function. The difference between these algorithms is only in the action of aging which happens in different frequencies. For example, if all of them do the aging in 100 packets. In the arrival of $100th$ packet, they age the histogram by scaling down each bin with different factors. For the intervening packets between the two successive agings, the corresponding bin in the histogram is increased by one for each packet. Algorithm 3 takes frequency (100 in this example) into account for the factor, but algorithms 1 and 2 do not. The next section discusses histogram bin width, which is strongly related to the accuracy of histogram data.

B. Bin Width of Histogram

The bin width of histogram is directly related to the accuracy of data stored. As defined above, bin width means the range of network delay grouped together to represent one pattern in histogram. For instance, bin width may be set to 10 milliseconds, and 205 milliseconds can be chosen to represent the network delay in the range of 200-209 milliseconds. In this example, all delays in the range of 200-209 milliseconds are stored in the same bin represented by the network delay of 205 milliseconds. Narrow bins carry higher accuracy than wider bins. The number of bins required, however, increases directly as the width of the bins is narrowed. As indicated above, the number of bins may range from 0 to ∞ depending on the network delay. Although it is possible to limit the upper and lower bounds, it is still necessary to reduce the number of bins if the networks suffer heavily congestion which results in a very wide range of delay. To increase the bin width therefore reduces the number of bins, each packet delay being stored to the histogram is as follows:

$$b = \lfloor \frac{nd}{w} \rfloor + 1 \tag{18}$$

where nd is the network delay, w is the bin width, and b is the corresponding bin in histogram. $\lfloor\,\rfloor$ is the floor function. Thus, all network delays in w range are represented by only one bin. After storing the network delay of the packet to the corresponding bin, it is still necessary to recalculate the network delay. It could be done by:

$$nd = \lfloor b \times w - \frac{w}{2} \rfloor \tag{19}$$

This takes the mean value of this range to represent the network delay for this bin. One can also choose the maximum or minimum value which are $(b \times w - 1)$ and

$[(b-1) \times w]$, respectively. The other alternative is to calculate the real mean value of delay in this range. This approach, however, requires additional overheads and memory to keep track and store the mean value of delays for each bin. This may not be suitable since an aim of wider bins is to reduce the memory needs.

The following two sections present an evaluation of Concord based on a trace-driven simulation. Section IV compares the performance of Concord with a fixed approach and a well-known reactive algorithm. In this case Concord is designed to operate without the additional complexity of aging. Section V examines aging, demonstrating the improved performance benefits of employing aging as part of Concord's operation.

IV. Evaluating the Basic Effectiveness of Concord

A simulation of Concord was designed such that it could be used in a trace-driven manner. Actual traces of audio and video were recorded using streams as they traversed the Internet to AT&T in New Jersey (`research.att.com`) from a variety of hosts. These hosts were located on both the East and West coasts and also in the UK. Traces were gathered using each source during the morning, afternoon and evening on different days of the week, and on different weeks in July/August 1995. In this paper we focus on the results using traces from `stanford.edu` (13 hops) to illustrate the general conclusions we observed using the larger set of traces. Audio streams are 64 Kb/s with packets generated every 20 milliseconds; video is 128 Kb/s with packets generated every 200 milliseconds (i.e. 5 frames per second). These values are typical of the conferencing tools currently used on the MBONE. Traces were 10 minutes in duration.

Packets were transmitted using conventional UDP, each containing a sequence number and the transmission time as recorded at the sender. This information was saved for each packet in addition to the recorded time of arrival at the receiver. The sender/receiver clocks were not synchronized, so the differences include any clock offset as well as network delay. This is not a problem since our focus is on delay variations, their distribution, and inter-stream delays, rather than absolute delay measurements. Thus we define network delay as that quantity in excess of the minimum observed delay from sender to receiver for a given stream *or* group of simultaneous streams. The latter is determined by finding the packet with the smallest difference in send/receive times (we assume negligible clock drift over the lifetime of each trace). In addition to the notation used in the previous section, we define *alp* as the actual late packets (%) as measured. This is the number of packets that arrive and are intentionally discarded by Concord because they are deemed to have arrived too late. It does not include packets losses due to network congestion. The next section analyzes Concord for single stream synchronization.

A. Single Stream Synchronization

The Concord algorithm is predictive, in that it uses available information about recent network behavior and/or any service contracts to estimate future delay distributions and calculate buffer delays. This distinguishes it from the time preserving techniques which are fixed in nature and maintain a static buffer delay regardless of changes in network conditions or packet lateness. Similarly, it differs from the data preserving techniques which take a reactive approach, looking at the most recent information on network behavior and increasing the buffer delay to avoid lateness. Thus these approaches constrain either *mad* or *mlp* but not both. Concord, on the other hand, allows a balance of these parameters.

The future Internet might be capable of giving a QoS contract which may be sufficient to allow Concord to set a *ted* value to be used throughout a given stream's lifetime and provide input in constructing a PDD. The behavior of today's Internet [4] is not amenable to such an approach, so it is important that Concord must operate dynamically, revising its PDD and recalculating buffer delays when needed. In this case *ted* can vary throughout a stream lifetime. The following section illustrates the use of fixed and reactive approaches to synchronize a set of audio streams, while Section IV-A.2 examines and compares the Concord scheme. Tables II & III & IV show the results using each algorithm for three traces. Note that a consequence of our definition of network delay is that it is acceptable to compare the performance of different algorithms for any individual stream, but not to compare absolute network delays for different unrelated streams.

A.1 Fixed and Reactive Approaches

The results for the fixed approach were produced using a *ted* value of 200 milliseconds. Note that the percentage of late packets is calculated based on the total number of received packets, rather than the total number of transmitted packets. Similarly, details of burst losses refers only to those packets which arrived but were discarded as being late. Figure 3 illustrates *nd* in a time plot for one of the traces. Packets which appear above the *ted* line are discarded for being late. Figures 4 & 5 show the corresponding segment for reactive and Concord algorithms respectively.

For comparison we show the results from a reactive algorithm. The algorithm we chose was introduced after observing that sudden variations in Internet transit delay are common, and that existing algorithms tend to react too slowly (hence dropping more packets than necessary) [5]. It is based closely on the playout algorithm used in the well-known Nevot audio tool, and very similar to that used in the popular MBONE tool called vat. It is considered a relatively simple technique, with which a considerable amount of practical experience has been gained.

Without repeating the full details here, in essence this algorithm detects sudden variations by comparing network delay for the current packet and the previous packet. It reacts by adjusting the total delay as a function of the most recently observed delay value. The algorithm returns to

	ted (milliseconds)					Burst Loss Lengths		
Algorithm	Min	Mean	Max	Std	*alp* (%)	Min	Mean	Max
Fixed	200	200	200	0.0	0.2	1	1.00	1
Reactive	21	190	387	17.1	0.0	1	1.00	1
Concord	160	195	201	4.4	0.2	1	1.17	3

TABLE II
RESULTS FOR TRACE T1

	ted (milliseconds)					Burst Loss Lengths		
Algorithm	Min	Mean	Max	Std	*alp* (%)	Min	Mean	Max
Fixed	200	200	200	0.0	94.2	1	9.64	99
Reactive	37	257	655	16.0	0.0	0	0.00	0
Concord	247	256	257	1.0	0.1	1	1.00	1

TABLE III
RESULTS FOR TRACE T2

	ted (milliseconds)					Burst Loss Lengths		
Algorithm	Min	Mean	Max	Std	*alp* (%)	Min	Mean	Max
Fixed	200	200	200	0.0	4.3	1	2.46	12
Reactive	26	192	384	24.2	0.0	1	1.00	1
Concord	170	221	230	6.9	0.5	1	1.73	6

TABLE IV
RESULTS FOR TRACE T3

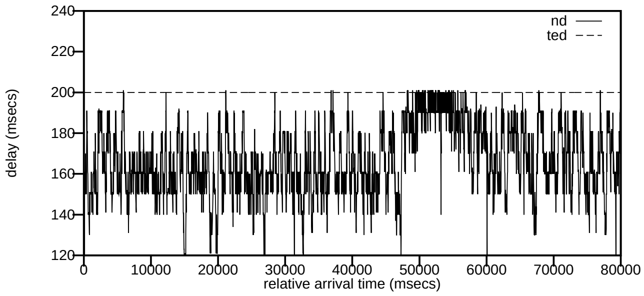

Fig. 3. Time plot segment for trace t1 with fixed algorithm (*ted* = 200 milliseconds)

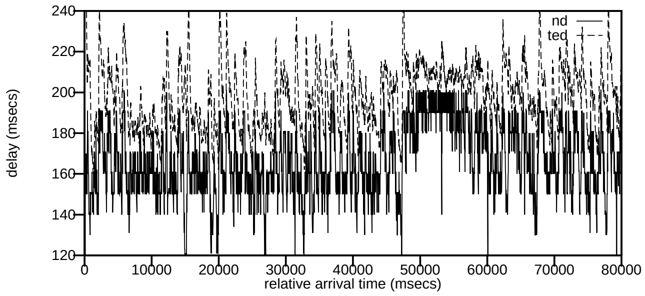

Fig. 4. Time plot segment for trace t1 with reactive algorithm

normal operation when the transient condition ends. It is designed to execute for every received packet and deals explicitly with audio, relying on a number of constant values which were derived after extensive examination of many audio traces. Note that *ted* was again initialized to 200 milliseconds for each experiment. Unlike the fixed algorithm, this scheme aims to minimize packet lateness by changing the buffer delay as necessary. The result is that *ted* tends to vary considerably, even over short time periods.

A.2 Concord

Concord uses available information on delay probabilities to set a value for *ted* such that application supplied values for the *mlp* and *mad* parameters are satisfied. In this section we analyze Concord's dynamic mode of operation, where the value of *ted* is occasionally recalculated. In essence, this requires the receiver to maintain a historical record of observed *nd* values in the form of a measured histogram. The performance of the algorithm is then expected to improve over time as this structure is built up, so the obvious optimization is to initialize it appropriately if reasonable information is available (e.g. using recent data from that source). The recorded information is consulted occasionally and used to revise the current *ted* if necessary to stay within the application QoS parameters. We use two controls over Concord's dynamic behavior. The first is a threshold factor which determines when *ted* is recalculated. The second controls the relevance of the histogram data by aging its contents over time with the aim of more

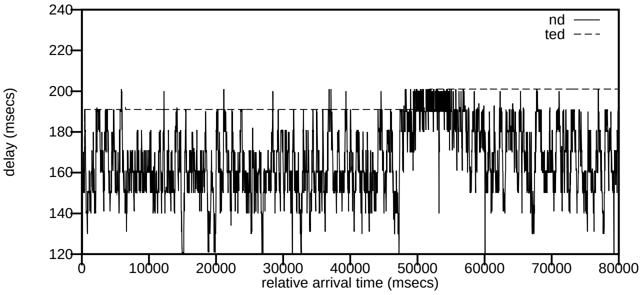

Fig. 5. Time plot segment for trace t1 with Concord algorithm ($mlp = 1\%$)

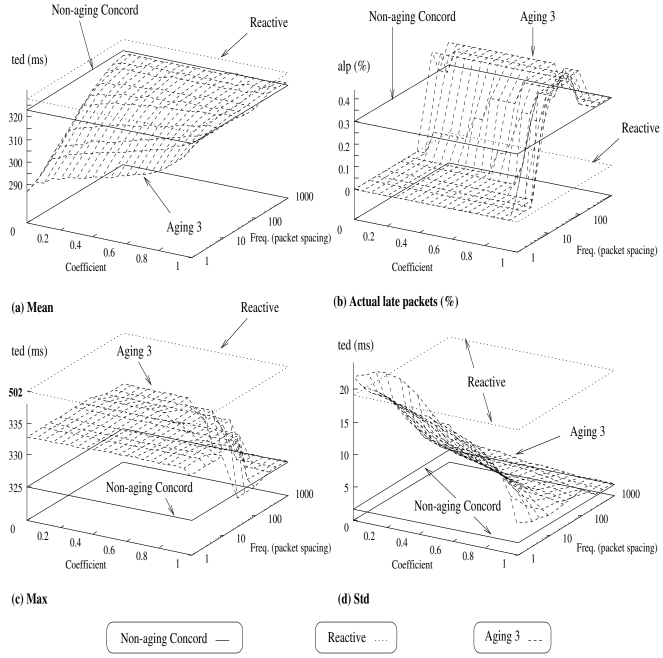

Fig. 6. Aging algorithm 3. (a) mean *ted*, (b) actual late packets (%), (c) maximum *ted*, (d) standard deviation of *ted*.

accurate behavior. The execution overhead can be reduced by choosing a threshold factor which results in a smaller number of *ted* recalculations, but of course this may result in decreased effectiveness. Aging increases the execution overheads, although this can be reduced by using a larger bin width. We use default settings for the following experiments: recalculate *ted* for each arriving packet, a histogram with one millisecond bin-width, and *no aging* of histogram contents.

A.3 Discussion

The Concord algorithm started with an empty histogram which it updated as each packet arrived. The results were obtained using a value of $mlp = 1\%$, thus easing comparison with the reactive algorithm. An immediate observation is that the maximum and standard deviation for *ted* are less than for the reactive algorithm, which in turn can allow receiver buffer requirements to be reduced. Thus Concord does indeed result in lower maximum delays and less variation in delays than the reactive algorithm. This is due to the fact that Concord uses a larger sample space when making delay adjustment decisions. Values for mean *ted* are similar in both algorithms. The nature of the reactive algorithm can lead to some samples receiving a very low delay, although this number is typically very small, e.g. $0.03\% < 100$ milliseconds for trace t1. This explains the significant difference in minimum delay values as shown.

A feature of Concord which can be a drawback is that multiple successive packets can be discarded. These burst loss quantities were not significant as shown in the current results. To gauge the effect of these losses on the audio quality we performed a crude subjective analysis by listening to a voice segment after Concord processing. The result was quite audible and compared well with the original voice segment. There were periods when the discarded packets caused a break up in continuity, but these were short and infrequent enough to not make listening uncomfortable. Naturally, the choice of a higher *mlp* value would allow Concord to discard a greater number of packets and this in turn would cause a reduction in audio quality. In comparison, using the reactive approach there are negligible discarded packets and hence the quality is basically unaffected, while using the fixed approach the quality varied wildly from near perfect (as in trace t1) to unintelligible (as in trace t2).

In summary, it is clear that Concord outperforms the reactive approach by producing lower values for maximum delay and delay variation. There are no conclusive differences between the average delay values of the two algorithms. The reactive algorithm relies on a relatively straightforward delay estimate closely related to that used in TCP, while Concord requires use of a history of measured delays along with a relatively simple prediction calculation. Thus the choice of algorithm is effectively (a) a trade off of memory for storing the histogram against the need to have reduced maximum delays, (b) an application's ability to tolerate a small increase in additional late packets when using the Concord algorithm. In light of this trade off we consider that Concord is most suitable for applications such as IP Telephony which have tight bounds on end-to-end latency, and may be willing to sacrifice a small increase in packets lost due to lateness. In the next section we further demonstrate that the addition of aging to Concord yields additional delay reductions, with a modest increase in the complexity of the prediction calculation.

V. Evaluating Effectiveness of Aging in Concord

The previous section evaluated the basic performance of Concord. This section examines the impact of the proposed aging techniques on the performance of Concord, using the same trace-driven simulation environment.

A. Aging Functions

We present simulation results for the different aging algorithms described in section III-A. For comparison, results using a reactive algorithm and the version of Concord

Algorithm	*ted* (Milliseconds) Min	Mean	Max	Std	*alp* (%)	Burst Loss Lengths Min	Mean	Max
Reactive	23	328	502	19.0	0.0	1	1.00	1
Concord	276	323	325	1.8	0.3	1	1.06	3

TABLE V
COMPARISON OF REACTIVE ALGORITHM AND CONCORD WITHOUT AGING.

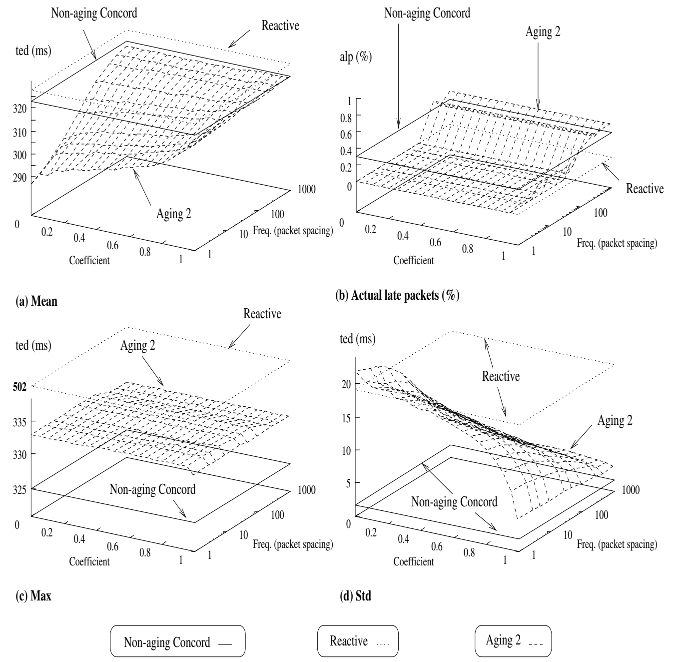

Fig. 7. Aging algorithm 2. (a) mean *ted*, (b) actual late packets (%), (c) maximum *ted*, (d) standard deviation of *ted*.

without aging are also presented. Table V compares the reactive algorithm and Concord without aging, in which mean *ted*, maximum *ted*, standard deviation of *ted*, and actual late packets (*alp*) are also plotted in figures 6-7. In figures 6-7, z-axis is *ted* or *alp*, while x-axis and y-axis are aging coefficient and aging frequency, respectively. Note that aging frequency in y-axis is based on logarithm scale. The *mlp* is set to 1% in Concord so that the mean values are almost identical in both reactive and non-aging Concord cases. The focus here is on the aging function and the comparison between aging and non-aging Concord algorithms. Note that aging coefficient and frequency have no effect for the reactive and non-aging Concord. Their values of *ted* and *alp* remain constant in figures 6-7. The experiments explore varying aging coefficient in x-axis and aging frequency in y-axis, where the aging coefficient is ranged from 0 to 0.99. The aging frequency is based on the number of packets received. The histogram is aged every f packets, where $1 \leq f \leq 1000$. A lower value of aging frequency implies a more frequent aging. The 3-dimensional presentation of figures 6-7 shows how variation in both aging coefficient and aging frequency affects *ted* and *alp* in Concord. Although there are three different algorithms, we show figures for results from aging algorithms 2 and 3 only. Algorithm 1 has the same general trends as those indicated in figures 6 (a)-(d). The following paragraphs discuss algorithm 3 first. Then rather than repeating the same discussion for all of them, we make a direct comparison between these three algorithms.

Maximum *ted* is directly related to the buffer requirements. Mean *ted* and *alp*, the actual late packets (%), are the parameters of QoS. Standard deviation demonstrates the delay variation (jitter). Minimum *ted* was omitted since it does not provide very useful information. The analysis in section III-A shows that the bigger the coefficient, the higher the *ted* (equations (7)-(9)). As expected, the mean *ted* decreases as the aging coefficient decreases in figure 6 (a). When the aging coefficient is smaller, the older samples have less of an impact. If the aging is more frequent (corresponding to a lower packet spacing frequency in the figure), the data stored is very current and reacts quickly to changes. When the frequency is greater than 100 packets, the change in *ted* is not very significant while varying the coefficient. However, changing in *ted* is visible while varying the frequency for a high coefficient value. This indicates that the aging function should choose a frequency less than a certain number in order to have the statistical data *retire* properly. On the other hand, even a high coefficient can age the old data if the frequency is chosen correctly. When the aging *coefficient* equals 0, it is the equivalent of *flush and refresh*. Similarly, the special case of *full aggregation* is when the aging *factor* equals 1. Figure 6 (a) also indicates that aging coefficient and frequency have no effect on the non-aging Concord and reactive algorithms.

Figure 6 (b) shows the packet lateness rate. As in fig. 6 (a), decreasing the frequency or coefficient generally reduces the lateness rate. However, there are exceptions in the high frequency and high coefficient area. When frequency is less than 50, the lateness rate is almost 0 independently of the coefficient value. This is because old data is aged frequently which results in that the chosen *ted* being close to the current delay. Please note that all of them are less than the *mlp* which is set to 1%.

Figure 6 (c) examines the maximum *ted* which is directly related to the buffer size required in the receiver. When aging frequently and more aggressive scaling for the old statistical data, *ted* changes frequently as the network delay changes. There is a higher probability that the chosen *ted* is close to the network delay, which might vary from 0 to ∞. This causes higher maximum values than in the non-aging or *less-aging* cases where *ted* relies more on historical data. Figure 6 (c) indicates that it reaches an upper bound quickly as the coefficient and frequency decrease from high values. A similar situation applies to figure 6 (d) which shows the variation of *ted*. The difference in (c) and (d) is that (d) shows the deviation of *ted* while (c) takes the maximum. Figure 6 (d) indicates that decreasing frequency and coefficient may cause high delay variation which might have an impact on the audio-visual quality as perceived by a human user. That is the price paid for lower *ted* and the small lateness rate shown in (a) and (b). A larger buffer may also be needed as shown in (c).

A.1 Comparison of Different Algorithms

Although the results of algorithm 1 are not shown here, *ted* in algorithm 2 is smaller than that in algorithm 1, especially for high frequency which means performing the aging less frequently. Algorithm 1 always scales down the historical data with a fixed factor (equation (1)), while the factor used to scale down the old data in algorithm 2 depends on the sum of the bin contents in the histogram (equation (10)), which leads to the same ratio of old data to newly arrived packet value (equation (11)). Therefore, in algorithm 2 the newly added bin contributes the same weight to the histogram through the lifetime of the stream. In algorithm 1, however, a new bin added after aging has less influence as shown in equation (3). Hence, algorithm 2 reacts to changes faster than algorithm 1. This is reflected in the small mean value in *ted*, and general small lateness rate. As discussed above, a quick reaction in changes of network delay results in bigger maximum *ted* and delay variation in algorithm 2.

Algorithm 3 takes the aging frequency into consideration as well. The factor used to scale down the old data increases while the aging frequency increases (equation (14)). Comparing it to algorithm 2, the impact is that it reacts to changes slowly in high aging frequency. As discussed above, small coefficient changes the statistical behavior no matter in high or low frequency. Therefore, the difference in algorithm 2 and 3 should be mainly in high-frequency and high-coefficient area.

Comparing figures 6-7, (a) and (d) have similar trends with some differences as discussed above. Generally speaking, (b) in the figures is similar also. For (c), the maximum *ted* has a sudden drop-off in the high-frequency and high-coefficient area in algorithms 3. This is not the case in algorithm 2. For all algorithms, if the aging is more frequent and the coefficient is smaller, the data stored is very current and it reacts quickly to changes. The maximum delay of this trace stream is 333 which is not very far from the mean value of 288. Figures show that the maximum *ted* of algorithm 2 reaches the maximum delay closely with any coefficient and frequency, while algorithm 3 does not reach it in the high-frequency and high-coefficient area since it does not react quickly enough in this area. Also, algorithm 3 reacts more slowly than algorithm 2 in this high-frequency and high-coefficient area and does not achieve the maximum delay in that area. Overall, there are tradeoffs between these algorithms. For the application with a short stream lifetime, algorithm 1, which is the simplest, could be the choice because the impact of different weight in each aging is not apparent for a stream with short lifetime. For the application with a long stream lifetime, algorithms 2 or 3 could be used. Comparing to algorithm 1, they still react to aging as time goes by. When aging does not happen very often, algorithm 2 is more suitable for applications which still need to react fast to changes in network delay. Algorithm 3, however, is the choice for those require small buffer even if aging takes place less frequently.

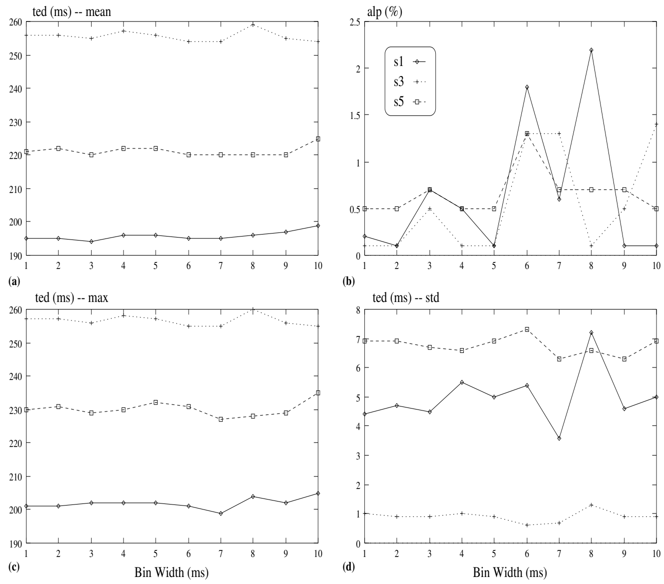

Fig. 8. Bin width results without aging. (a) mean *ted*, (b) actual late packets (%), (c) maximum *ted*, (d) standard deviation of *ted*. Results are shown with three different sources: S1, S3, and S5.

B. *Bin Width of Histogram*

In this experiment, we study the effect of different bin widths. The bin width is set to 1 millisecond initially. *mlp* is set to 1% also. We then repeat the same process with different bin widths. The *ted* chosen to represent the bin is the mean value of this bin as in equation (19). For example, if the bin width is 25 milliseconds, 12 milliseconds is chosen to represent the *teds* which fall in the range of 0-24 milliseconds. Due to the space limitations, figure 8 only shows the results without aging for the trace from `stanford.edu`. Although not shown here, other traces with or without aging have the same trends generally.

Figure 8 shows the results from three different streams which are labeled by S1, S2, and S3. Results show that the mean value of *ted* changes slightly while the bin width changes. The value may be increased or decreased without any constraints. The packet lateness rate, however, may increase very quickly when the bin width increases. Figure 8 (b) indicates that the packet lateness rate might be much larger than the original setting when the bin width is bigger than 10. However, it may still maintain the *alp* which is close to the initial setting when the bin width is less than 10. Suppose the original bin width is 1, and $0 \le a < c < b$, where a, b, and c are different network delays. Let's choose a new bin width w which equals $(b-a+1)$, and c is the mean value which represents the network delay for all delays in the range of $[a, b]$. For all chosen *teds* in the range of $[c, b]$, they are replaced by c now in the wider-bin case. If the network delay of one arrived packet is greater than c but less the original *ted*, it is not lost when using the original bin width but it is lost in the wider-bin case. The actual lateness rate depends on the network delay and the chosen *ted*. However, this is more common when the bin width is bigger. Therefore, we can see that the lateness rate is generally higher when the bin width is bigger, and when the bin width is bigger, the chosen *ted* may deviate more from the original value. This may cause higher standard deviation. However, if all *teds* are deviated to the same value, it results in a small standard deviation. Hence, figure 8 (d) shows that the standard deviation may be higher or lower while changing the bin width. Since we choose a mean value to represent the delay in the range of w, the mean and maximum *teds* change slightly while the bin width changes.

Overall, the experiments indicate that narrow bins carry higher accuracy than wider bins. For most streams, increasing the bin width slightly won't change mean *ted*, maximum *ted*, *alp* and *std* too much. In simulations, the maximum delay may be ranged from 200 to 300 milliseconds, which needs around 200 to 300 bins. When the bin width is increased from 1 to 2 milliseconds, the number of bins required is actually half of the original one. It is only one fourth when the bin width is increased to 4 milliseconds. This reduction in memory may be sufficient for most cases.

VI. Related Work

The area of delay adaptation for packet voice has been receiving attention for many years [6], [7]. Early experimental systems for packetized voice on a LAN used a fixed scheme for managing the playout buffer [8]. More recently, reactive approaches have become popular in the Internet, such as that employed in the *vat* audio tool (based on delay estimation from [9]) and the related improvements proposed in [5]. More general adaptation work has also appeared, both for application design [10], [11] and middleware support for cooperative QoS management [12]. Synchronization between multiple related streams has also been a popular research topic. Some solutions assume that reasonable upper bounds on the delay across streams are available [13], while others set a fixed upper delay bound for a stream, discarding data which arrives late and would cause a loss of synchronization [14].

More closely related to our current work are efforts to reduce delays for reactive approaches. The technique of [15] is based on a reactive algorithm which is modified to quickly reduce the buffer size after it had been increased to handle a delay spike. The decision is made after observing the jitter encountered by the most recently arrived packets. More recently, [16] also advocates a predictive-style approach to sizing the playout buffer, aiming to reduce the delays of the reactive approach from [5]. It operates by recording packet delay information with the aim of determining the average network delay under normal conditions, using that information to react more effectively to delay spikes. Rather than maintaining a statistical approximation of the delay distribution as in our proposal, they record actual delays for a fixed number of previously arrived packets, currently set to 10000. In [17], a new reactive algorithm is presented, which uses a simple least-mean-square adaptive predictor for network delays, demonstrating lower delays in comparison to conventional reactive approaches. Unlike our work these schemes do not explore allowing explicit application control over lateness or interaction.

For aging techniques, [18] presents a page replacement algorithm, called LRU-K, for database disk buffering by considering only the last K references to a page. Comparing to our approaches, LRU-K does not accumulate all historical data in histogram and employ techniques to gradually diminish the effect of old data. The exponential aging proposed in [19] is similar to our approaches. It however recommends a constant aging factor of 0.9 which is more like our aging algorithm 1 only. Several other agings have been proposed for different purposes also. The priority aging in [20], for instance, continually depresses a job's priority until the end of every decay cycle. The aging window in [21] is the amount of time a record is eligible for reintegration. Comparing to our approaches, most of them simply keep track of time then take some actions such as update, remove or reuse the data in certain time points.

VII. Conclusion

This paper presented the Concord algorithm for synchronizing networked multimedia streams. Concord is notable because it defines a solution for synchronization, that operates under the direct influence of application-supplied parameters for QoS control. In particular, these parameters are used to allow a trade-off between the packet lateness rates, total end-to-end delay and skew. Thus, an application can directly indicate an acceptable lost packet rate, rather than having the synchronization mechanism operate by always trying to minimize losses due to lateness.

We have described the details of the Concord algorithm and its extension. Issues of calculating the receiver buffer size and estimating network delays were discussed. In addition, features for dealing with imperfect clocks and dynamic network behavior were described. The evaluation is based on traces of audio and video traffic which had traversed the Internet. Concord employs a new predictive approach, which we evaluate and compare with other algorithms. We demonstrate that Concord can operate with lower maximum and variation in total end to end delay, which in turn can allow receiver buffer requirements to be reduced. In addition, we show it discards late packets at a predictable rate, as specified by the application along with a maximum acceptable delay bound. We also explore the use of aging techniques to improve the effectiveness of the historical information and hence the delay predictions. Three aging algorithms are proposed and compared using a simulation driven with the same Internet traffic traces. In comparison to the type of reactive approach typically used in the Internet, the results show that Concord algorithm with aging can produce significant reductions in buffering delay and jitter at the expense of packet lateness values of less than 1%.

REFERENCES

[1] N. Shivakumar, C. J. Sreenan, B. Narendran, and P. Agrawal, "The Concord algorithm for synchronization of networked multimedia streams," in *Proc. of IEEE International Conference on Multimedia Computing and Systems*, pp. 31–40, May 1995.

[2] C. J. Sreenan, B. Narendran, P. Agrawal, and N. Shivakumar, "Internet stream synchronization using Concord," in *Proc. of IS&T/SPIE Conference on Multimedia Computing and Networking*, pp. 352–359, 1996.

[3] P. Agrawal, J.-C. Chen, and C. J. Sreenan, "Use of statistical methods to reduce delays for media playback buffering," in *Proc. of IEEE International Conference on Multimedia Computing and Systems*, (Austin, TX), pp. 259–263, June 1998.

[4] D. Sanghi, A. K. Agrawala, O. Gudmundsson, and B. N. Jain, "Experimental assessment of end-to-end behavior on Internet," in *Proc. of IEEE INFOCOM*, pp. 867–874, 1993.

[5] R. Ramjee, J. Kurose, D. Towsley, and H. Schulzrinne, "Adaptive playout mechanisms for packetized audio applications in wide-area networks," in *Proc. of IEEE INFOCOM*, (Toronto, Ont., Canada), pp. 680–688, June 1994.

[6] G. Barberis and D. Pazzaglia, "Analysis and optimal design of a packet-voice receiver," *IEEE Transactions on Communications*, vol. 28, pp. 217–227, Feb. 1980.

[7] W. A. Montgomery, "Techniques for packet voice synchronization," *IEEE Journal on Selected Areas in Communications*, vol. 1, pp. 1022–1028, Dec. 1983.

[8] S. Ades, R. Want, and R. Calnan, "Protocols for real time voice communication on a packet local network," in *Proc. of IEEE International Conference on Communications (ICC)*, pp. 525–530, June 1986.

[9] V. Jacobson, "Congestion avoidance and control," in *Proc. of ACM SIGCOMM*, pp. 314–329, Aug. 1988.

[10] P. Moghe and A. Kalavade, "Terminal QoS of adaptive applications and its analytical computation," in *Proc. IFIP International Workshop on Quality of Service*, pp. 369–380, May 1997.

[11] P. Sisalem, "End-to-end quality of service control using adaptive applications," in *Proc. IFIP International Workshop on Quality of Service*, pp. 369–380, May 1997.

[12] C. J. Sreenan and P. P. Mishra, "Equus: a QoS manager for distributed application," in *Proc. IFIP International Conference on Distributed Platforms*, pp. 496–509, Mar. 1996. Published as "Distributed Platforms" by Chapman & Hall, 1996.

[13] J. Escobar, D. Deutsch, and C. Partridge, "Flow synchronization protocol," in *Proc. of IEEE GLOBECOM*, pp. 1381–1387, 1992.

[14] R. Yavatkar, "MCP: a protocol for coordination and temporal synchronization in multimedia collaborative applications," in *Proc. IEEE International Conference on Distributed Computing Systems*, pp. 606–613, June 1992.

[15] D. L. Stone and K. Jeffay, "Queue monitoring: a delay jitter management policy," in *Proc. of Intl Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, Nov. 1993.

[16] S. B. Moon, J. Kurose, and D. Towsley, "Packet audio playout delay adjustment: performance bounds and algorithms," *ACM/Springer Multimedia Systems Journal*, vol. 6, pp. 17–28, Jan. 1998.

[17] P. De Leon and C. J. Sreenan, "An adaptive predictor for media playout buffering," in *Proc. of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, (Phoenix, AZ), pp. 3097–3100, Mar. 1999.

[18] E. J. O'Neil, P. E. O'Neil, and G. Weikum, "An optimality proof of the LRU-K page replacement algorithm," *Journal of the ACM*, vol. 46, pp. 99–112, Jan. 1999.

[19] A. Brunstrom, S. T. Leutenegger, and R. Simha, "Experimental evaluation of dynamic data allocation strategies in a distributed database with changing workloads," in *Proc. of International Conference on Information and Knowledge Management*, (Baltimore, MD), pp. 395–402, Nov. 1995.

[20] D. H. J. Epema, "Decay-usage scheduling in multiprocessors," *ACM Transactions on Computer Systems*, vol. 16, pp. 367–415, Nov. 1998.

[21] L. B. Mummert, M. R. Ebling, and M. Satyanarayanan, "Exploiting weak connectivity for mobile file access," in *Proc. of ACM symposium on Operating Systems Principle*, (Copper Mountain, CO), pp. 143–155, Dec. 1995.

Cormac J. Sreenan is a Professor of Computer Science at University College Cork in Ireland. Prior to taking up his current position in August 1999, he was a Principal Technical Staff Member at AT&T Labs Research in Florham Park, NJ, USA, and a Member of Technical Staff at Bell Labs in Murray Hill, NJ, USA. He holds a Ph.D. in Computer Science from Cambridge University. His research interests include multimedia networking, mobile computing, and packet telephony.

Jyh-Cheng Chen (S'96-M'99) received the B.S. degree in information science from Tunghai University, Taichung, Taiwan, R.O.C. in 1990. In 1992, he received the M.S. degree in computer engineering from Syracuse University, Syracuse, NY. He received the Ph.D. degree in electrical engineering from the State University of New York at Buffalo in 1998. From 1995 to 1996, he worked as a software engineer at ASOMA-TCI, Inc., North Tonawanda, NY. During the summer of 1997, he worked in the Networked Computing Technology Department at AT&T Labs, Whippany, NJ and contributed to energy efficient MAC protocols for wireless ATM networks. Since August 1998, he has been a Research Scientist in the Applied Research at Telcordia Technologies (formerly Bellcore), Morristown, NJ. At Telcordia Technologies, his research is focused on third generation wireless networking, next generation networks, QoS for IP networks, Internet telephony, and multimedia communications. Dr. Chen is a member of ACM.

Prathima Agrawal (S'74-M'77-SM'86-F'89) is Executive Director of the Computer Networking Research Department and Assistant Vice President of the Internet Architecture Research Laboratory at Telcordia Technologies (formerly Bellcore), Morristown, NJ. She is also a Visiting Professor of Electrical and Computer Engineering, Wireless Networking Laboratory (WINLAB), Rutgers University, NJ. Prior to this she worked for 20 years in AT&T/Lucent Bell Laboratories in Murray Hill, NJ, where she was Head of the Networked Computing Research Department. Presently, she leads the ITSUMO joint research project between Telcordia and Toshiba Corp. ITSUMO is a third generation wireless access system for multimedia communication over end-to-end packet networks. Dr. Agrawal received her Ph.D. degree in Electrical Engineering from the University of Southern California. Her research interests are computer networks, mobile and wireless computing and communication systems and parallel processing. She has published over 150 papers and has received or applied for more than 40 U.S. patents. Dr. Agrawal is a Fellow of the IEEE and a member of the ACM. Presently, she chairs the IEEE Fellow Selection Committee.

B. Narendran has a B.Tech. in Computer Science from the Indian Institute of Technology, Madras, India, and a Ph.D. in Computer Science from the University of Wisconsin-Madison. He was a Member of Technical Staff at Bell Laboratories, Lucent Technologies from 1993 to 1997 where he worked on a variety of research issues in wireless and multimedia networking. He is currently a VP at Juno Online Services.