# The Concord Algorithm for Synchronization of Networked Multimedia Streams

N. Shivakumar* C. J. Sreenan B. Narendran P. Agrawal

AT&T Bell Laboratories
Murray Hill, NJ 07974

## Abstract

*Synchronizing different data streams from multiple sources simultaneously at a receiver is one of the basic problems involved in multimedia distributed systems. This requirement stems from the nature of packet based networks which can introduce end-to-end delays that vary both within and across streams. In this paper, we present a new algorithm called Concord, which provides an integrated solution for these single and multiple stream synchronization problems. It is notable because it defines a single framework to deal with both problems, and operates under the influence of parameters which can be supplied by the application involved. In particular, these parameters are used to allow a trade-off between the packet loss rates, total end-to-end delay and skew for each of the streams. For applications like conferencing this is used to reduce delay by determining the minimum buffer delay/size required.*

## 1 Introduction

The use of packet based communication networks to transport digitized streams of audio and video is becoming increasingly common. A characteristic of these networks is that the total delay experienced by each packet is a function of variable delays due to physical media access and relay queueing, in addition to fixed propagation delays. The result is that the time difference between transmitting any two packets at the source is unlikely to be the same as that observed between their arrival at the destination. This is a problem for a stream of multimedia packets, because the presence of delay variations (known as *jitter*) can have an impact on the audio-visual quality as perceived by a human user. Delay is also a problem when using multiple related streams, because it introduces a temporal difference which is undesirable in situations where, for example, *lip-sync* is required. We describe these two problems as single and multiple stream synchronization respectively.

To solve these synchronization problems, one must introduce additional delay by buffering packets at or near the point of presentation. In the single stream case this is used to smooth out jitter prior to passing data samples to an output device. For multiple related streams it is used to compensate for inter-stream delay differences. There are two basic approaches to operating this synchronization buffer (sometimes called an elastic buffer) [5]:

- *Packet-Preserving:* In this scheme, the receiver waits until a packet $p$ arrives to play back the packet.
- *Time-Preserving:* If a packet $p$ does not arrive within a certain time bound, the packet is assumed to be lost, and subsequent packets are played back. If $p$ does arrive later, it is thrown away.

Our focus is on time-preserving applications and the additional requirements associated with the total end-to-end delay: the cumulative delay suffered by packets in the network and buffer. Clearly, if every packet is delayed in the buffer such that it suffers cumulatively (in the network and buffer) a delay equal to the maximum network delay, the receiver can reproduce a jitter-free playback. 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 average delay. This is important for real-time applications like conferencing and telephony, where large delays are detrimental to interactivity. For example, in packetized voice applications, figures in the 150 - 250 msec range are often quoted as being the maximum acceptable total end-to-end delay. It is possible, however, to

*Now at the Department of Computer Science, Stanford University.

take advantage of the built in redundancy of a multimedia stream to reduce the buffer delay at the expense of throwing away packets which arrive too late. This may not be the case with other data streams such as file transfers.

In this paper, we present the Concord algorithm, which 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 loss against the total end-to-end delay suffered by packets, by applying these parameters to probabilistic data describing packet delay distributions. Section 2 gives the details and also describes extensions to deal with the effects of sender/receiver clock drift. In Section 3 the solution for multiple stream synchronization extends this scheme by taking user input describing upper bounds on the acceptable difference in end-to-end delay (known as *skew*) for the related streams. This information is used, if necessary, to increase the selected buffer sizes to achieve synchronization between the streams. The scheme operates dynamically by updating its probability delay distributions and actual buffer sizes over time in response to the observed behavior of the network. Some early simulation work is presented in Section 4; followed by a review of related work and our conclusions in Sections 5 & 6 respectively.

## 2 Single Stream Solution

This section addresses the problem of single stream synchronization. After giving a formal definition of the problem, our algorithm is described in detail. The initial discussion ignores issues of imperfect clocks, an assumption which is removed and dealt with in Section 2.3.

### 2.1 Problem Formulation

As shown in Figure 1, we have a network $N$, over which sender $S$ wishes to send stream $M$ to receiver $R$, with the packets in $M$ being produced every $T_r$ seconds. All packets have sequence numbers. Stream $M$ is characterized by the *maximum acceptable delay* (MAD) it can suffer, and *maximum packet loss rate* (PLR) it can tolerate. For the network $N$ we construct a *packet delay distribution* (PDD): an estimate of the probable delays suffered by packets in the network over a time window. This PDD may draw on existing traffic conditions, history information or any negotiated service characteristics to derive estimates for minimum, maximum and/or mean delays and delay distributions. The PDD constructed from this information is approximate, but because its long-term accuracy is important, we assume that this can be improved dynamically as described in Section 2.2.5. 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 PLR and MAD are satisfied, if possible. That is, to find $B$ the minimum buffer size satisfying the following:

- For every packet $p$: $nd_p + bd_p$ is a constant $TED$, where $TED$ is the *total end-to-end delay*, $nd_p$ is the *network delay* suffered by packet $p$, and $bd_p$ is the induced *buffer delay* for $p$, a function of the buffer size.
- The chosen $TED$ is lesser than the $MAD$ of the stream.
- The chosen $TED$ does not lead to more than $PLR$ packets/sec being thrown away.

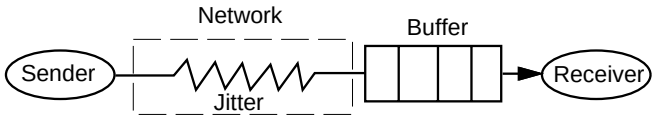


Figure 1: System elements

### 2.2 Algorithm Details

#### 2.2.1 Computing Total End-to-End Delay

In this section we describe how to calculate the end-to-end delay, using the type of PDD shown in Figure 2 to illustrate the procedure. This PDD shows a normal distribution of packets delays, which is typical of packet switched networks [2]. From this plot it is clear that all packets that suffer more than $TED$ in network delay will be declared lost in the time-preserving scheme. Hence $(1 - Cdf(TED))$ packets will be lost every second, where $Cdf$ is the *cumulative distribution function* on the $PDD$. There are a few possible directions at this point to choose the $TED$ of the packets.

1. *Minimize PLR:* Choose $TED$ so that $PLR$ is minimized while satisfying the $MAD$. $Cdf(t)$ is a monotonically increasing function, hence $(1 - Cdf(t))$ is monotonically decreasing. That is, the packet loss rate decreases with an increase in $t$.

Hence, the choice of $TED = MAD$, will have the minimum packet loss rate, since $MAD$ is the maximum permissible $TED$.

2. *Minimize TED:* Minimize $TED$ such that the $PLR$ is satisfied. This can be achieved by choosing the minimum $TED$ such that $(1-Cdf(TED))$ is $PLR$. Choosing any $t$ greater than this $TED$ will reduce the packet loss rate of the stream as seen above. Conversely, any $t$ lesser than $TED$ will lead to a greater packet loss rate than $PLR$. Hence the given choice of $TED$ will be the minimum $t$ satisfying the given $PLR$.

3. *Minimizing Hybrid-Cost:* Find a $TED$ such that some cost metric $cost[f(TED, plr)]$ is minimized, where $f(TED, plr) \rightarrow \Re$ with the constraints that $TED \leq MAD$, and $plr \leq PLR$, where $plr$ is $(1 - Cdf(TED))$.