Internet stream synchronization using Concord

C. J. Sreenan, B. Narendran, P. Agrawal

AT&T Bell Laboratories
Murray Hill, NJ 07974

N. Shivakumar

Dept. of Computer Science
Stanford University, CA 94305

ABSTRACT

Using packet networks to transport multimedia introduces delay variations within and across streams, necessitating synchronization at the receiver. This requires stream data to be buffered prior to presentation, which also increases its total end to end delay. *Concord* recognizes that applications may wish to influence the underlying synchronization policy in terms of its effect on quality of service (QOS). It provides a single framework for synchronization within and across streams and employs an application specified tradeoff between packet losses, delay and inter-stream skew. This paper presents a new predictive approach for synchronization and a selection of results from an extensive evaluation of Concord for use in the Internet. A trace driven simulator is used, allowing a direct comparison with alternative approaches. A conclusion is 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.

Keywords: multimedia, synchronization, Internet, quality of service

1 INTRODUCTION

There is considerable interest in developing and using multimedia applications and services which operate in geographically distributed computing systems. Typical examples include conferencing, distance learning and movies on demand. These services rely on the transport of packetized streams of continuous media, such as audio and video, over the Internet or ATM-based networks. In these networks the total delay experienced by each packet is a function of variable delays due to physical media access, relay queueing and choice of route, 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 multi-stream synchronization respectively.

Use of a network which offers quality of service (QOS) contracts can be used to mitigate these problems at the cost of reserving network resources. A more general solution is to introduce additional delay by buffering packets at or near the point of presentation. The questions of how to configure and operate such a buffer have received attention in the past but have led to independent solutions which do not explicitly consider the desired QOS requirements of the application involved. This provided motivation for our new stream synchronization algorithm known as *Concord*, which we introduced in a previous paper.[8] The Concord approach 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. This paper gives a summary of the features of Concord and presents a detailed evaluation of its performance. The results are produced using a set of captured traces for audio and video streams having traversed the Internet. The research contribution of this paper is towards a better understanding of the inter-relationship between QOS-controlled synchronization mechanisms and network behavior.

2 ALGORITHM BASICS

2.1 Approach

There are two basic approaches to operating a synchronization buffer (sometimes called an elastic buffer). In the *time preserving* approach, if a packet i does not arrive within a certain time bound, the packet is assumed to be lost, and subsequent packets are played back. If i does arrive later, it is said to be *late* and is thrown away. In the *packet preserving* approach, the receiver waits until a packet i arrives to play back the packet. 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 mean 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 millisecond (msec) range are often quoted as being the maximum acceptable total end-to-end delay. It is possible, however, to take 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 loss rate). This may not be the case with other data streams such as file transfers.

Concord performs a trade-off 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 packet delay distributions. For multi-stream synchronization we extend 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.

2.2 Operation

We define a stream as a sequence of packets, produced according to a period marked with a sequence number. A stream is characterized by the *maximum acceptable delay* (mad) it can suffer, and *maximum late packets* (mlp) it can tolerate. For each stream 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 delay bounds and distributions. The

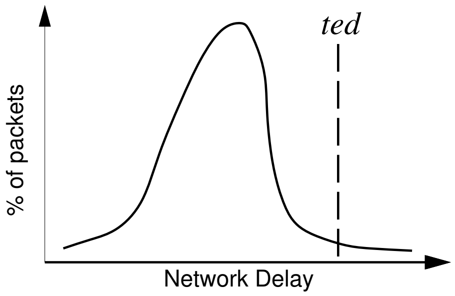

Figure 1: Example packet delay distribution

PDD constructed from this information is approximate, but over time it is updated dynamically so that it closely tracks the actual performance. The speed of getting this process established is accelerated by having initial PDD information which is reasonably accurate. 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* and *mad* are satisfied, if possible. That is, to find b the minimum buffer size satisfying the following:

- For every packet i: $nd_i + bd_i$ is a constant *ted*, where *ted* is the *total end-to-end delay*, nd_i is the *network delay* suffered by packet i, and bd_i is the induced *buffer delay* for i, 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 *mlp* percent of packets being thrown away.

We show how to calculate the end-to-end delay, using the type of PDD shown in Figure 1 to illustrate the procedure. This PDD shows a normal distribution of packets delays, which is typical of packet switched networks.[1] From this plot it is clear that all packets that suffer more than *ted* in network delay will be declared late in the time-preserving scheme. Hence $(1 - Cdf(ted))$ packets will be declared late, where *Cdf* is the *cumulative distribution function* on the PDD. There are a few possible directions at this point to influence the behavior of Concord. In essence one can choose a value of *ted* so that either *mlp* or *ted* are minimized (best visualized by moving the *ted* line to the right or left respectively in the figure).

We have extended this basic technique for the multi-stream case by assigning the ted_s for each stream to equal ted_{sys} as follows:

- As before, calculate minimum acceptable delay $t_s = Cdf_s^{-1}(1 - mlp_s)$ for each stream s = 1 ... n.
- Choose ted_{sys} to be the maximum of the computed t_s, s = 1 ... n, hence satisfying all *mlp* requirements.
- If $ted_{sys} > mad_s$ for any s = 1 ... n, there is no feasible solution for these stream parameters.
- Choose b_s, the buffer size for stream s to be $(ted_{sys} - min_s)/p_s$, where min_s is the estimated minimum network delay for that stream source and p_s is its period.

This represents the situation where perfect synchronization is required. Concord also allows an application to specify that it can tolerate some skew between streams using the *maximum relative skew* (*mrs*) parameter. This is used to relax the feasibility constraint by allowing streams to be within *mrs* of the derived ted_{sys}.

3 EVALUATION

The aim of the present work is to evaluate Concord for operation in the Internet. A simulation of the algorithm's behavior 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 Bell Labs in Murray Hill, New Jersey (`research.att.com`) from a host in one of several locations in the USA and one location in the UK. In this paper we focus on the results using traces from `stanford.edu` (13 hops) to illustrate the general conclusions. Audio streams are 64 Kbps with packets generated every 20 msec; video is 128 Kbps with packets generated every 200 msec (i.e. 5 frames per second). These values are typical of the conferencing tools currently used on the MBONE. A set of traces, each lasting ten minutes, were collected from each source during the morning, afternoon and evening of several weekdays in July/August 1995. Each single stream was transmitted on its own, while multi-streams were transmitted as one of an audio/video pair commencing simultaneously.

Packets were transmitted using the UDP protocol and contained 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 introduced in the previous section, we define *alp* as the actual late packets (%) as measured. The next section analyzes Concord for single stream synchronization, while Section 3.2 deals with the multi-stream case.

3.1 Single stream case

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 to 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[2,7] 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 from packet to packet. The following section illustrates the use of fixed and reactive approaches to synchronize a set of audio streams, while Section 3.1.2 examines and compares the Concord scheme. Tables 1 & 2 & 3 show the results using each algorithm for three traces. Note that a consequence of our definition of *nd* is that it is acceptable to compare the performance of different algorithms for any individual stream, but not to compare absolute *nd* for different unrelated streams. This is not true in the multi-stream case.

3.1.1 Fixed and reactive approaches

The results for the fixed approach were produced using a *ted* value of 200 msec. 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 2 illustrates *nd* in a time plot for one of the traces. Packets which appear above the *ted* line are discarded for being late. Figures 3 & 4 show the corresponding segment for reactive and predictive algorithms respectively.

	ted (msec)					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
predictive	160	195	201	4.4	0.2	1	1.17	3

Table 1: Results for trace t1

	ted (msec)					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
predictive	247	256	257	1.0	0.1	1	1.00	1

Table 2: Results for trace t2

	ted (msec)					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
predictive	170	221	230	6.9	0.5	1	1.73	6

Table 3: Results for trace t3

For comparison we show the results from a reactive algorithm. The algorithm we choose was introduced after observing that sudden variations in Internet transit delay are common, and that existing algorithms tend to react too slowly.[4] 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 normal operation when the exceptional 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 msec for each experiment. Unlike the fixed algorithm, this scheme aims to minimise packet lateness by changing the buffer delay as necessary. The result is that *ted* tends to vary considerably, even over short time periods.

3.1.2 Predictive approach: Concord

As described in Section 2, 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

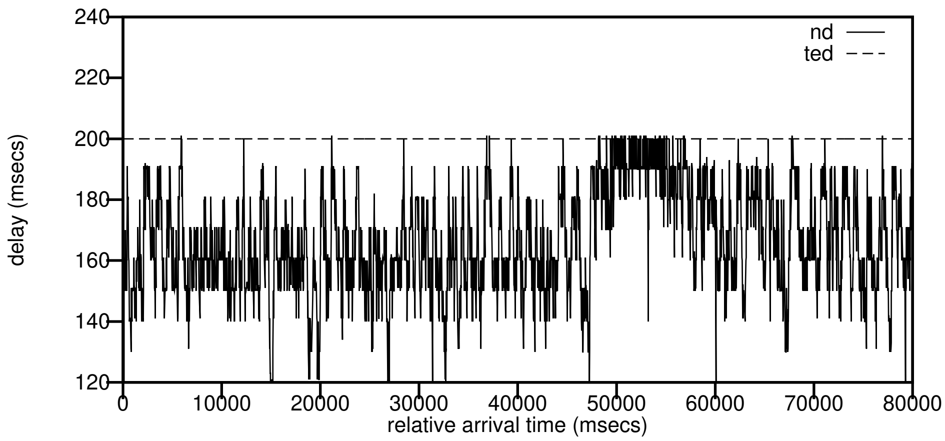

Figure 2: Time plot segment for t1 with fixed algorithm ($ted = 200$ msec)

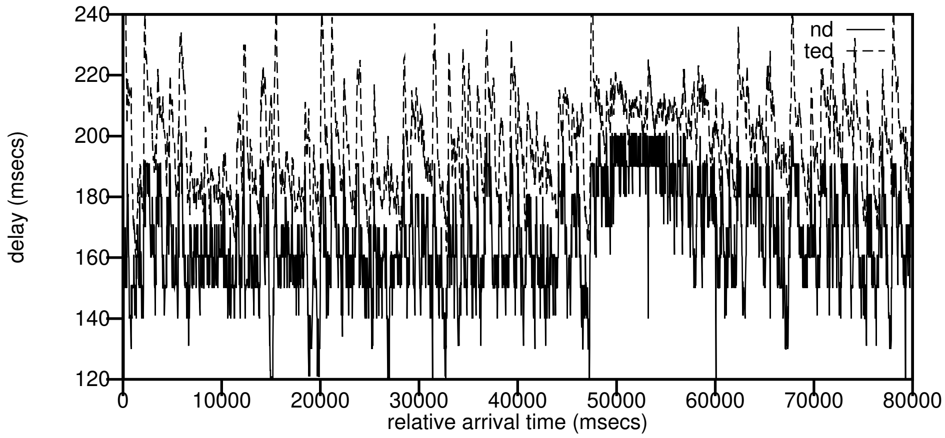

Figure 3: Time plot segment for t1 with reactive algorithm

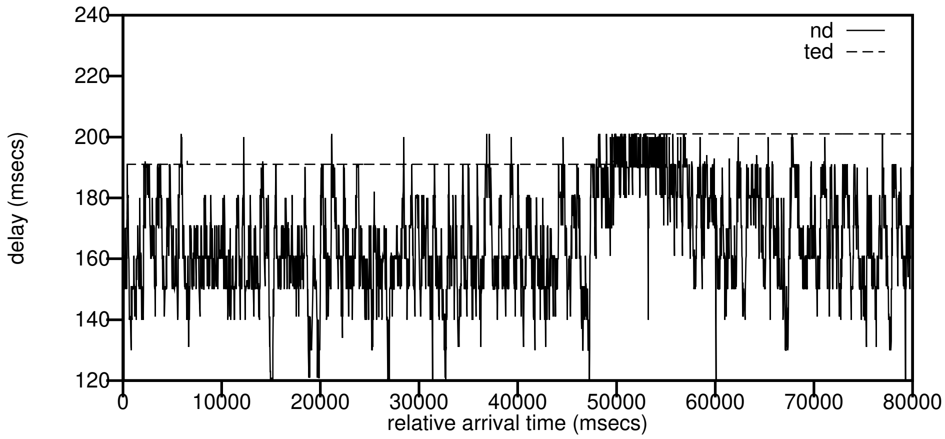

Figure 4: Time plot segment for t1 with predictive algorithm ($mlp = 1\%$)

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 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 size. We use default settings for the following experiments: recalculate *ted* for each arriving packet, a histogram with one millisecond bins, and no aging of histogram contents.

The 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. This results from Concord's use of a larger sample space when making delay adjustment decisions. Values for mean *ted* are similar. 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 msec 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 losses were not generally significant as shown in the current results.

3.2 Multi-stream case

An attractive feature of Concord is that the basic algorithm extends quite naturally to deal with multi-stream synchronization. In this case a group of related streams are transmitted simultaneously from the sender to the receiver, and the QOS parameters are extended to allow the maximum relative skew (mrs) between streams to be specified. A value of zero for this parameter is a demand for perfect synchronization. Positive values indicate that an amount of relative difference between streams is acceptable, e.g. a value of 150 msec might be used to achieve lip-sync. In essence, the mrs value is used to constrain the *ted* of the streams within the group.

As described in section 2.2, the mechanism operates by calculating ted_{sys} and ensuring that all ted_i values are within mrs of that value, by increasing them if necessary. In addition to specifying a skew limit, the relationship between streams can also be defined. The default relationship is one in which all streams are equal, and the ted_{sys} is determined by the maximum *ted* within the group. We call this the *leader* stream. Of course, when operating Concord in dynamic mode, this means that the leader may change as ted_i values are recalculated, similar in principle to Rothermel's protocol.[6] It also means that as conditions change the system may become infeasible, i.e. not possible to satisfy mrs and each stream's mad and mlp constraints. At this point we envisage that the application should be consulted and asked to relax some of the parameters. This situation does not occur if the application chooses to specify that streams have a master-slave relationship. In that case, the application specifies that one stream is the master, and slaves are kept within $+/-mrs$ of it, violating their mlp and mad parameters if necessary, i.e. these are treated as preferences rather than constraints.

4 RELATED WORK

Several of the early experimental systems for packetized voice used a time-preserving scheme for the receiver buffer. More recent work has produced a method for calculating accurate delay estimates in the absence of global clocks,[1] and compared policies for changing the buffer size in packet-preserving operation.[4]

Some multi-stream solutions assume that reasonable upper bounds on the delay across streams are available,[3] while others set a fixed upper delay bound for a stream, discarding data which arrives late and would cause a loss of synchronization.[9] The multi-stream problem has also been defined in terms of various forms of data frames, relying on the underlying transport protocol to constrain their jitter and delay.[5]

5 CONCLUSION

This paper presents a detailed evaluation of the Concord algorithm for synchronizing multimedia streams. Concord is notable because it defines a common solution framework for both single and multi-stream synchronization, while operating under the direct influence of application supplied parameters for QOS control. 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. A conclusion is 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, it is shown to discard late packets at a predictable rate, as specified by the application along with a maximum acceptable delay bound. Future work will include studying the dynamic control parameters to improve the efficiency and operation of the algorithm.

6 REFERENCES

[1] F. Alvarez-Cuevas, M. Bertran, F. Oller and J. M. Selga. "Voice synchronization in packet switching networks," IEEE Network, pages 20–25, September 1993.

[2] J. C. Bolot. "End-to-end packet delay and loss behavior in the Internet," *Proceedings of ACM SIGCOMM '93,* pages 289–298, September 1993.

[3] J. Escobar, D. Deutsch and C. Partridge. "Flow synchronization protocol," *Proceedings of IEEE GlobeCom '92,* pages 1381–1387, 1992.

[4] R. Ramjee, J. Kurose, D. Towsley and H. Schulzrinne. "Adaptive playout mechanisms for packetized audio applications in wide-area networks," *Proceedings of IEEE INFOCOM '94,* pages 680–688, June 1994.

[5] K. Ravindran and V. Bansal. "Delay compensation protocols for synchronization of multimedia data streams," IEEE Transactions on Knowledge and Data Engineering, 5(4):574–589, August 1993.

[6] K. Rothermel and T. Helbig. "An adaptive stream synchronization protocol," *Proceedings of the 5th International Workshop on Network and Operating System Support for Digital Audio and Video,* pages 189–201, April 1995.

[7] D. Sanghi, A. K. Agrawala, O. Gudmundsson and B. N. Jain. "Experimental assessment of end-to-end behavior on Internet," *Proceedings of IEEE INFOCOM '93,* pages 867–874, 1993.

[8] N. Shivakumar, C. J. Sreenan, B. Narendran and P. Agrawal. "The Concord algorithm for synchronization of networked multimedia streams," *Proceedings of the 2nd International Conference on Multimedia Computing and Systems (ICMCS '95),* pages 31–40, May 1995.

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