Clock Skew Compensation over a High Latency Network

Dominique Fober, Stéphane Letz, Yann Orlarey
GRAME Research Laboratory
9 rue du Garet, BP 1185, 69202 LYON Cedex 01, France
[fober, letz, orlarey]@grame.fr

Abstract

Exchange of time stamped events between different stations raises the problem of the clock frequencies difference as soon as one station try to compensate for the transmission delay and to render the events with a minimum time distortion. We propose a simple, efficient and low cost method to compensate for the clock frequencies difference. This method rely only on regular time stamped packets transmissions and may be used in many cases. It provides good performances to the receiver station in regard of the sender reference time even on a heavily loaded communication channel. It operates also very efficiently on a low latency local network.

1. Introduction

In the musical domain, real-time rendering of time ordered events transmitted over a high delay network is a tricky task as time ordering is part of the musical information itself and should be preserved with a maximum acuracy. Obviously, playing the transmitted events at reception time is not a solution as the transport latency variation will introduce an important time distortion which may reach several hundred of milliseconds on the Internet. Therefore, any suitable real-time transmission protocol should include mechanisms to compensate for the latency variation. Such mechanisms are based or equivalent to buffering technic [1, 2, 3, 4] which consists in delaying the events rendering by a fixed value, greater than the maximum latency variation expected. However, if the receiver and sender clock frequencies differ, this buffering technique will sooner or later reach its limit:

- if the receiver clock is running faster than the sender clock, the receiver will consume the events faster than produced, exhausting at term the provisions made to compensate for the transport latency,
- if it is running slower, the number of buffered events will increase with time, exhausting at term the receiver memory space.

For the rest of the paper, we'll refer to the clock frequencies difference as the "*clock skew*".

Clock synchronization has been subject of many works. As it is one of the most basic problem in distributed systems, many Clock Synchronization Algorithms (CSA) have been developed [5, 6, 7] to ensure that physically dispersed process will acquire a common notion of time using local physical clocks and message exchange over a communication network.

Beside distributed systems point of view, protocols such as NTP or SNTP [8, 9] have been designed, intended to synchronize clocks over the Internet. Partialy based on NTP, time synchronization has also been approached in the context of audio streams [10].

Our proposed method differs from the protocol approaches for the following reasons:

- it operates without requiring a master clock,
- it doesn't require any transaction to operate and therefore may be used with an existing protocol such as MWPP [11].

The clock skew detection is based on distributed systems algorithms, modified to take account of peer-to-peer connections and of the high transmission delay introduced by the Internet.

The rest of this paper is structured as follow: section 2 presents the clock skew detection algorithm, section 3 is dedicated to its implementation, section 4 deals with performances issues, and section 5 summarizes and outlines the future developments of this work.

2. Clock skew detection algorithm

2.1 Related technologies

In distributed systems, a common approach to establish a common time base consists in using fault-tolerant interactive convergence algorithms. These systems are generaly made up of several nodes, located on the same network. The principle of these algorithms is the following:

- at regular intervals called “resynchronization intervals”, all the nodes of the system exchange their clock values in order to determine a clock correction term. Communication between the nodes is generaly achieved using broadcast of time stamped messages over the network.
- for each received message, a node will compute a clock deviation as the difference between the message reception and transmission dates, minus the current evaluation of the transport latency.

At the end of the resynchronization interval, each node will have collected a set of clock deviations, which are then used to compute a clock correction term. Several methods exists, we reused the following:

- the Fault Tolerant Midpoint Algorithm (FTMA): operates on a sorted set of clock deviations, it assumes that some nodes of the system are sending faulty values: these faulty values are therefore discarded and the correction term is then computed as the arithmetic mean of the lower and higher remaining values.
- the Adaptative Exponential Fault Tolerant Midpoint Algorithm (AEFTMA): operates a smoothing of the correction terms produced by FTMA using exponential smoothing technics, where the weighting factor is dynamically computed according to the current correction term.

Compared to these works, our situation constitutes a special case from several points of view:

- only two nodes are involved in the skew detection process,
- they don't have to agree on a common time base: each node is independantly estimating its clock deviation compared to its peer node,
- as the latency variation is used to detect the clock skew, we cannot consider that there are faulty messages.

2.2 Peak latency filtering

We have made several transport latency measurements using differents network paths. It appears that this latency globally fits in a constant range, with more or less frequent peaks. The range width and the peaks amplitude may greatly vary depending on the Internet service provider and the network path.

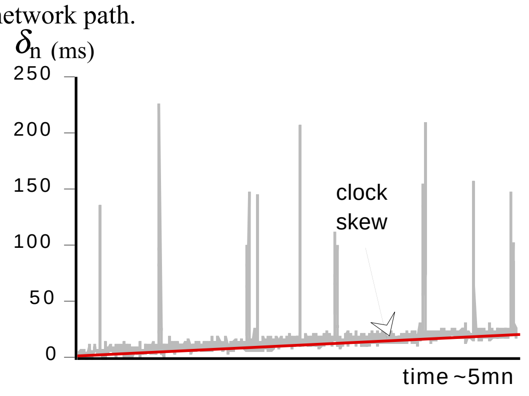

Figure 1: continuous latency variation measured over 5 mn

The figure 1 shows 5 minutes of continuous measurements done over a 42,6 kbps modem line connection via a free Internet service provider, at a time renowned for being a high traffic period. The measurement has been done using time stamped UDP packets sent every 200 milliseconds. The slow latency increase slope is due to the clock skew and detection of the skew may be viewed as measuring this slope.

The intended solution consists in peak filtering and in computing the convergence point of the remaining values.

Our algorithm, named Peak Tolerant Midpoint Average algorithm (PTMA) operates similarly to FTMA with the following differences:

- the sorted FTMA clock deviation vector made up of the *n* nodes deviations collection is transformed into a sorted latency variation vector made up over time,
- the discarded values count is not limited to the *k* Byzantine tolerance,
- the arithmetic mean is computed using all the remaining latency variations (and not only the lowest and highest).

Let *w* be the time window size of the latency variation collection and *k* be the number of values discarded per window. At the time *t*, the sorted latency variation vector is

$$\Delta_t = [\delta_{t-w} \ldots \delta_i \ldots \delta_t],\ \delta_i \leq \delta_{i+1}$$

and the average midpoint latency variation is then computed as:

$$LV_t = \frac{\sum_{t-w+k/2}^{t-k/2} \delta_i}{w-k}$$

Intuitively, the algorithm operation may be viewed as a drastic selection on the latency variations: it tends to discard all the lowest and highest variations; but also as a selection on the variation history as it retains only a discontinuous subset of the sliding temporal window. Applied to the measured latency variations, we obtain what we call the *skew profile*.

2.3 Skew profile smoothing

However, in a more chaotic transmission context, the skew profile obtained with PTMA is varying significantly around the real clock skew. Therefore we have extended PTMA to the Exponential Peak Tolerant Midpoint Average algorithm (EPTMA) by simply applying exponential smoothing to the PTMA output. Assume that LV_t is the *skew profile* value at time *t*, EPTMA computes the corresponding smoothed value by:

$$LV^{t}_{EPTMA} = \alpha . LV_t + (1 - \alpha) . LV^{t-1}_{EPTMA}$$

where the weight factor α is such as $0 \leq \alpha \leq 1$.

Choosing a small α value gives more weight to the passed values and minimizes the effect of any pulse while this effect is more persistent.

2.4 Initialization speeding-up

Acuracy of the results is highly dependant of the parameters used by the algorithm. In particular, choosing a large value for the temporal PTMA window size W and a small value for the retained values R will increase the long term acuracy but may also delay significantly the first results produced by the algorithm. Therefore, we improved PTMA in order to produce values earlier. Let *n* be the number of values currently pushed in the window: as long as *n* is lower than W, we dynamically compute the retained values *r* as:

$$r_n = R - \frac{R}{W^2}(n - W)^2$$

which produces a parabolic curve from 0 to R when n varies from 0 to W.

Always to speed up the initialization process, we choosed to use a weighting factor $\alpha = 1$ for the first value pushed into EPMTA.

3. Implementation

Detection of the clock skew requires only to receive packets time stamped with their transmission date at a sufficient rate to guaranty a good acuracy. Input values of the algorithm are latency variations which may be simply measured as follow:

- at first packet reception, a receiver host evaluates its initial *apparent clocks offset* (ACO_1) as the difference between the current local time and the packet time stamp.
- then for any following packet, the latency variation LV_n is:

  $$LV_n = ACO_n - ACO_1 \ (n > 1)$$

  where ACO_n is the apparent clock offset for the packet n.

Let $CORR_n$ be the correction term returned by EPTMA for the packet n, the corrected local time is then:

$$T_{CORR} = T - CORR_n$$

where T is the current local time.

4. Performances

Actually, the system behave like shown in figure 2: at initialization stage (time 0), the receiver and sender clock offset has a given value (150 for the example) which will continue to move up to the stable stage. Note that on the given example, the clock deviation is zoomed 10 times.

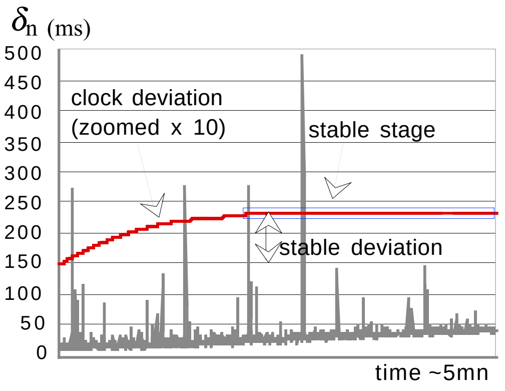

Figure 2: system behavior

At the stable stage time, the additionnal clock deviation becomes equal to:

$$K = k \pm \mathcal{E}(eptma)$$

where k is the stable constant deviation and $\mathcal{E}(eptma)$ represents the error due to the skew profile evaluation.

We choosed to characterize the algorithm performances with both k and $\mathcal{E}(eptma)$ values: the k value is important because it is to be added to the provisions made for the latency compensation and the error denotes the stability of the system.

The results presented below come from a simulation of the system on a single station, using however both latencies generated by an abstract model of a network path behavior and latencies collected from real network transmissions.

3.1 Internet measurements

For the measurments below, we made use of the following parameters:

• packets transmission rate:	200 ms
• PTMA window size:	200 values
• retained values:	20 values
• EPTMA weighting factor α:	0.01

First results have been collected using an abstract model of a network path behavior with a clock deviation of 2 for 10000 time units. Figure 3 illustrates an example of the transport conditions which may be qualified as bad.

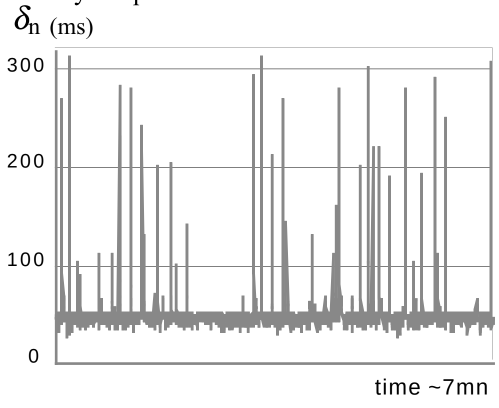

Figure 3: example of simulated latencies.

Results of 100 successive simulations over 5 minutes are presented by the table 1: it shows the maximum values reached, the average values and the standard deviations.

	k	$\mathcal{E}(eptma)$
max	20.6	–1.25
average	6.79	–0,67
std dev	5.06	0.20

Table 1: simulation of a busy network

	k	$\mathcal{E}(eptma)$
max	23.25	–0.65
average	8.11	–0.49
std dev	8.32	0.17

Table 2: real world latencies

We have also measured the performances using real latencies variations collected on the Internet. The estimated clock deviation was 1 for 10000 time units. Other parameters remain unchanged. Results are presented by the table 2.

It appears that in both cases, the system presents a good stability and that the remaining clock deviation may be considered as low in regard of the expected latency over the Internet.

Using real world data, we have ignored the worst transport conditions encountered because the latency variation has exceeded reasonable values for a correct time rendering (over several seconds). However, it is interesting to examine the system behavior in this exceptionnal context. Figure 4 presents the clock deviation for one of these worst cases. Note that the deviation deviation is still zoomed 10 times.

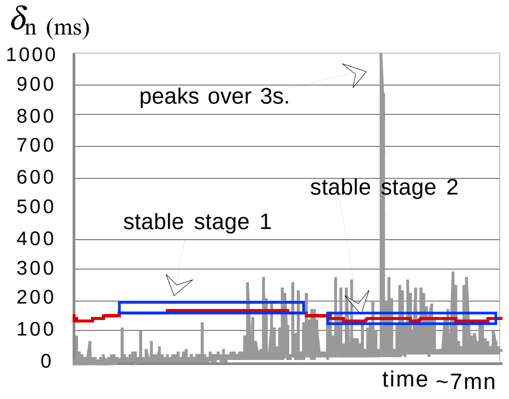

Figure 4: worst case latency

It appears that the system is self-adapting to medium or long term changes of the network behavior: it smoothly switch from one stable stage to another one. Note also that the overall error $\mathcal{E}(eptma)$ has always been lower than ±2.

3.2 Local network measurements

The algorithm operates also very efficiently on a low latency local network. Table 3 presents the corresponding results. Latency is characterized by a range of 1 ms delay with peaks up to 5 ms.

	k	$\mathcal{E}(eptma)$
max	0,7	−0.05
average	0,69	−0.01
std dev	0,02	0.02

Table 3: local network performances

The parameters used changed from previous experiments:

• packets transmission rate:	200 ms
• PTMA window size:	30
• retained values:	10
• EPTMA weighting factor α:	0.2

The receiver clock deviation was 2 for 10000 time units. The system shows a very good stability and a remaining clock deviation under one time unit.

4. Conclusion

We have proposed a new algorithm to compensate for the clock skew on a high delay network. It runs very efficiently on the Internet but also on a local network. Its presents many advantages compared to previous works:

- it doesn't require any transaction to operate and therefore it may be used independantly of the transport protocol,
- it doesn't rely on a master/slave scheme: each receiver is independantly evaluating its clock skew relatively to the sender,
- consequence of the previous point: as the compensation scheme is attached to a given real-time stream, a network of any complexity can operate correctly and in particular, a receiver may handle several incoming streams with different clock skews characteristics.

This algorithm has been implemented in the form of MidiShare [12] drivers and provides real-time communication over Internet or over a local network to the system client applications.

It may be probably more improved: in particular concerning the way to handle the initalization stage (the first seconds of a transmission) which is critical in regard of the future efficiency.

Acknowledgements

This research was partly supported by the Mil Productions Company (Villefranche/Saone - France). We would like to thank it for its support.

References

[1] Fober D. Real time, musical data flow on Ethernet and MidiShare software architecture. *Proceedings of the ICMC*, 1994, ICMA, San Francisco, pp. 447-450

[2] M. Goto, R. Neyama, Y. Muraoka. RMCP: Remote Music Control Protocol. *Proceedings of the ICMC*, 1997 ICMA San Francisco, pp.446-449

[3] J.P. Young, I. Fujinaga. Piano master classes via the Internet. *Proceedings of the ICMC*, 1999 ICMA San Francisco, pp.135-137

[4] D. Fober, Y. Orlarey, S. Letz. Real Time Musical Events Streaming over Internet. *Proceedings of the International Conference on WEB Delivering of Music*, 2001, pages 147-154

[5] T.K. Srikanth, S. Toueg. Optimal Clock Synchronization. *Journal of the ACM*, vol. 34, pp. 626–645, July 1987.

[6] M.M. de Azevedo and D.M. Blough, "Fault-Tolerant Clock Synchronization for Distributed Systems with High Message Delay Variation, *Fault-Tolerant Parallel and Distributed Systems,* D. Pradhan and D. Avresky, eds., pp. 268–277, IEEE CS Press, 1995.

[7] R. Ostrovsky, B. Patt-Shamir. Optimal and efficient clock synchronization under drifting clocks. *Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing,* 1999, pp. 3 - 12

[8] D. L. Mills. Network Time Protocol (NTP) version 3. IETF, RFC 1305, 1992

[9] D. L. Mills. Simple Network Time Protocol (SNTP) version 4. IETF, RFC 2030, 1996

[10] E. Brandt R.B. Dannenberg. Time in Distributed Real-Time Systems. *Proceedings of the ICMC*, 1999 ICMA San Francisco, pp.523-526

[11] J. Lazzaro, J. Wawrzynek. The MIDI Wire Protocol Packetization (MWPP). Internet-Draft. IETF, 2002, http://www.ietf.org/internet-drafts/draft-ietf-avt-mwpp-midi-rtp-01.txt (work in progress)

[12] Y. Orlarey, H. Lequay. MidiShare : a Real Time multi-tasks software module for Midi applications - *Proceedings of the ICMC 1989*, ICMA San Francisco, 1989, pp.234-237