# Real Time Musical Events Streaming over Internet

Dominique Fober Yann Orlarey Stephane Letz
*Grame - Computer Music Research Laboratory*
*9, rue du Garet BP 1185*
*69202 LYON CEDEX 01*
*[fober, orlarey, letz]@grame.fr*

## Abstract

*We present a new protocol to transmit time ordered events in real-time over Internet and to operate a correct time rendering on the receiver side. This protocol provides solutions to compensate for the network latency, to optimize the bandwidth use and to take account of the clock drift of the different stations involved in a transmission. It is particularly suitable to transmit musical events such as MIDI events. The implementation is based on the User Datagram Protocol (UDP) however, the proposed solution is independant of the underlying network layers.*

## 1. Introduction

Real-time streaming of digital continuous media such as audio or video raises several problems identified at the beginning of the 90s in term of requirements for real-time communication services [1]. They include delay, throughput and reliability requirements. As multimedia streams are generally large bandwith consumers, a particular attention has been given to throughput and protocols such as the Internet Stream Protocol (ST2) [2] or the Resource ReSerVation Protocol (RSVP) [3] [4] have been developed to support the efficient delivery of data streams to single or multiple destinations in applications that require guaranteed quality of service. This support is notably achieved with a resource reservation policy all along each node on the data path. Although these protocols do not affect routing, it involves the support of routers as the resources allocation should be maintained during all the session. Resource reservation has also been addressed at lower level protocol layer by ATM (specified in 1991 by CCITT I.361) and integration of real-time services in an IP-ATM network architecture [5] has been approached using ST2 or RSVP.

On the contrary, RTP, a Transport Protocol for Real-Time Applications [6], has been designed beside the quality of service concerns: it does not address resource reservation and does not guarantee quality of service for real-time services. But the data transport is augmented by a control protocol (RTCP) to allow monitoring of the data delivery in a manner scalable to large multicast networks, notably using regular transmission of RTCP reports carrying quality feedback. As RTP and RTCP are designed to be independant of the underlying transport and network layers, it may provide an efficient real-time streaming solution in association with a resource reservation protocol such as ST2.

Provided that their size is limited, events real-time transmission differs from audio or video streams because it does not require any particular bandwith. Transmission of MIDI formatted datas for example, can be achieved in far less time than necessary to render them: a Bach Two-Part Invention in C Major (BWV 772) MIDI file, which represents 6646 bytes and about 1 minute and 30 seconds of music, may be send in about 2 seconds using UDP on a 28.8 kbps line. Therefore, raw files transmission is fullfilling the requirements of most applications, possibly associated with buffering technics in case of large files, in order to start rendering before transfer completion.

However, this transmission way is not suitable when the datas are generated in real-time: in this case, further conventions between the sender and the receiver are required. As for audio streams, any protocol design will then hit the problem of transmission delay and network latency but in a particular manner as it should provide a way to render the events time ordering at receiver side.

Concerning the throughput requirement, although it may not be a problem, a special attention should be given to packets optimisation as a too frequent events transmission may jam the network with half filled packets: this may be the case if one tries to send a full MIDI data flow (or greater), which represents one 3 bytes MIDI packet at least every millisecond. In this case, the underlying transport layers overhead represents more than 91% of the packet content which does not constitute a

satisfactory solution from efficiency point of view.

Finally, one problem not addressed by the previous works concerns the sender and receiver time synchronisation. In fact, we are only interested in clock skew which represents the frequency difference between the clocks of two stations on the network. Unless a correction mechanism is provided, the clock skew may appear on a station as a constantly increasing network latency (or decreasing, depending on which clock is faster) and may prevent it from a correct time rendering.

There are few publications dealing with real-time streaming in the particular context of time ordered events. Young and Fujinaga have presented a work applied to piano master classes via the Internet [7]. It is based on OTUDP (Open Transport UDP), a Max object that transmits data using UDP [8], and focus mainly on packet losses. Time synchronisation is not an issue of the proposed solution. Another work applied to real-time MIDI streaming over Ethernet [8] includes bandwith optimization technics which are adopted in the present work.

The rest of the paper is structured as follow: section 2 presents the solutions provided to compensate for the network latency and optimize packets transmission. Section 3 focus on the mechanisms intended to handle clock skew. Section 4 presents the protocol implementation and experimental results. Section 5 summarizes and outlines the future developments of this work.

### 1.1. Terminology

Some terms used in this paper are defined as follow:

*time rendering*: the time rendering process ensures that time ordered events will be rendered at receiver side with the same time order, including offset between consecutive events.

*transport latency*: it represents the transport time from one protocol level to the same protocol level on remote host. The level examined is the presented protocol level, which means that the transport latency includes the possible host software latency as well as the network latency.

*rendering delay*: let $d_a$ be the date of an event occurrence on host A and $d_b$ the rendering date of this event on host B. Assuming that $d_a$ and $d_b$ are expressed in a reference time common to A and B, the rendering delay $D$ is defined as $d_b - d_a$. It includes the transport latency but also the delay introduced for the time rendering.

*clocks offset*: the offset of two clocks is the time difference between them.

*clock skew*: the skew represents the frequency difference between two clocks.

*apparent clocks offset*: the clocks offset increased by the current transport latency at measurement time.

## 2. Transport and time rendering

The mechanisms to transmit and to render events are based on a previous work on real-time MIDI streaming over Ethernet [8]. They relie on a *grouping period* and a *maximum latency allowed* to operate a correct time rendering of the transmitted events.

### 2.1. Grouping period

The grouping period is intended to minimize the packets frequency and to optimize the ratio between the transport layers overhead and the data to be transmitted. It represents the period during which events are accumulated before being sent on the network. It is also the lower bound of the packets transmission rate. Figure 1 presents the transmitted data count per packet according to the events rate and a grouping period varying from 10 to 200 ms. The events rate is expressed in number of events per second. It is assumed that each event is coded using 5 bytes. Figure 2 presents the corresponding protocols overhead ratio, including the underlying layers overhead. It is assumed that this overhead is 44 bytes per packet (IP, UDP and protocol headers) .

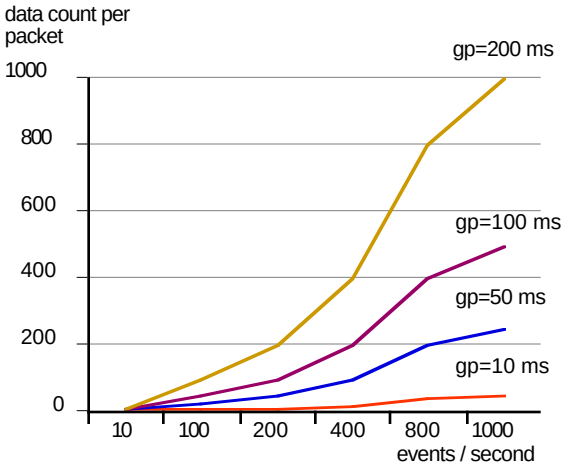


**Figure 1: Data count per packet according to the grouping period (gp) and the events rate.**

The grouping period affects the sender behavior and is part of the rendering delay. It has the additionnal effect to minimize the packet loss probability as shown in [9] and packets delivery inversion due to different routes from

source to destination. The grouping period value should be fixed according to the network transmission context and to the expected events rate. In particular, it may be greatly minimized if the protocol is intended to operate on a Local Array Network (LAN).

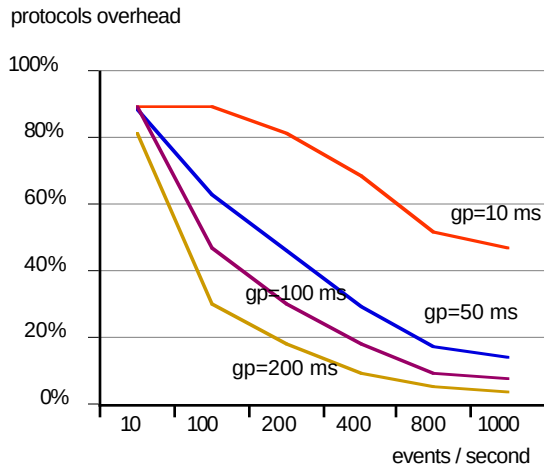


**Figure 2: Protocols overhead according to the grouping period and the events rate.**

## 2.2. Time rendering

As the next section is dealing with clock skew, we'll ignore it to explain the base mechanism of time rendering. This mechanism relies on a *maximum latency allowed* to ensure a correct time rendering of the transmitted events. It acts similarly to buffering technics: events rendering is delayed according to the current transport latency, which is equivalent to accumulate these events during a period equal to the maximum latency allowed.

The *maximum latency allowed* parameter affects the receiver behavior and is part of the rendering delay.

**2.2.1. Latency variation evaluation.** In fact, we are interested in the latency variation rather than the transport latency itself. First the *apparent clocks offset* is fixed at protocol initialization. We assume that each packet is time stamped with its transmission date at protocol level. Let $A_n$ be the time stamp of the $n$th packet sent by a host A, and $B_n$ the reception date of this packet on host B at the same protocol level, the apparent clocks offset is viewed on host B as:

$$\Theta = B_o - A_o \quad (1)$$

which includes the transport latency.

For any further packet transmission, the transport latency variation from host A to B is then:

$$\delta_n = B_n - A_n - \Theta \quad (2)$$

**2.2.2. Events time stamping.** At transmitter side, each event is time-stamped as an offset to its transport packet time stamp. At receiver side, this offset is first added to the packet reception date to produce the *local event time stamp*. Let $B_n$ be the $n$th packet reception date and $o_n^i$, the $i$th event time offset within the $B_n$ packet, the local event time stamp $e_n^i$ is then:

$$e_n^i = B_n + o_n^i \quad (3)$$

To compensate for the transport latency, the rendering date $r_n^i$ of the event $e_n^i$ is then computed as:

$$r_n^i = e_n^i + L_{max} - \delta_n \quad (4)$$

where $L_{max}$ is maximum latency allowed.

Due to (1, 2, 3), $r_n^i$ may be also expressed as follow:

$$r_n^i = B_o + A_n - A_o + L_{max} + o_n^i \quad (5)$$

which is independant of the latency variation $\delta_n$.

However, to operate a correct time rendering, we must ensure that the rendering date $r_n^i$ is greater or equal to the packet reception date $B_n$. Derived from (2), it can be expressed as:

$$B_o + A_n - A_o + L_{max} + o_n^i \geq A_n + \Theta + \delta_n \quad (6)$$

and from (1), it is equivalent to:

$$L_{max} + o_n^i \geq \delta_n \quad (7)$$

Assuming that $o_n^i$ is null for the first event in a packet, correctness of the events time rendering assumes that the latency variation never exceed the *maximum latency allowed* parameter.

## 3. Clock skew detection

Clock skew may be viewed at receiver side as a constantly increasing latency: let R be the ratio of two clocks frequency on hosts A and B. We consider that the transport latency is null. Then, according to (2), we should have:

$$B_n = A_n + \Theta \quad (8)$$

but according to the clocks skew, the reception date of the $n$th packet on B can be expressed as:

$$B_n = \frac{(A_n + \Theta)}{R} \quad (9)$$

which means that the latency variation is viewed on host B as:

$$\delta_n = (A_n + \Theta)\frac{1 - R}{R} \quad (10)$$

If the ratio R between the clocks A and B is equal to 1, then the latency variation $\delta_n$ is actually null, but if R is smaller than 1 (e.g. the B clock is running faster than

the A clock), then $\delta_n$ increases with $A_n$ and may reach more or less fast, the maximum latency allowed, and prevents the protocol from correct events time rendering.

### 3.1. Related technologies

Clock synchronization has been subject of many works. Among them, the Network Time Protocol (NTP) [10] is intended to synchronize clocks over the Internet. It may be run beside our protocol however, such an architecture would come up against some NTP limitations in regard to our requirements: in particular, acuracy achieved by NTP is directly dependant on the time taken to achieve it and periods of many hours and dozens of measurements are required to resolve oscillator skew and maintain local time to the order of a millisecond.

As clock synchronization is one of the most basic problems in distributed systems, many Clock Synchronization Algorithms (CSA) have been developed [11, 12, 13] to ensure that physically dispersed process will acquire a common notion of time using local physical clocks and message exchange over a communication network. Although fewer, research has also been conducted on clock rate synchronization [14, 15]. All these works generally apply to multi-nodes systems. A common approach consists of using fault-tolerant clock synchronization algorithms and frequently, via interactive convergence algorithms [16, 17] such as the sliding window algorithm (SWA), the fault-tolerant midpoint algorithm (FTMA), the adaptative exponential fault-tolerant midpoint algorithm (AEFTMA) and the multistep interactive convergence algorithm (m-ICV). Our solution for clock skew detection is based on these convergence averaging algorithms.

### 3.2. Skew detection algorithm

Compared to the previous works on clock and rate synchronization, 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.

**3.2.1. Peak latency filtering.** We made several transport latency variation 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 3 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. Table 1 shows the corresponding traceroute output.

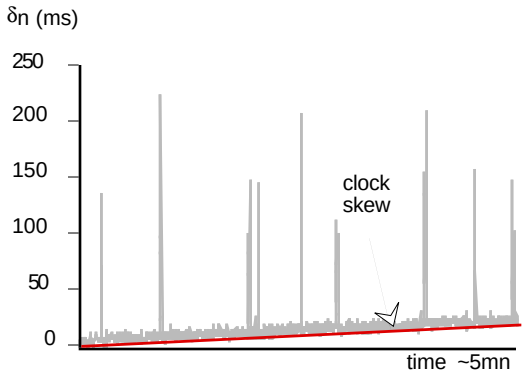


**Figure 3: Continuous latency variation measured over 5 mn**

The continuous measurement has been done using time stamped UDP packets sent every 200 milliseconds. The slow latency increase slope corresponds to the measured clocks deviation on peer hosts.

**Table 1: Traceroute output**

| | |
|---|---|
| 1 | lyon2-2-58-254.dial.proxad.net |
| 2 | paris11-2-a1.routers.proxad.net |
| 3 | telehouse-6.routers.proxad.net |
| 4 | teleglobe.net |
| 5 | if-0-0.core1.paris.teleglobe.net |
| 6 | No host name found. |
| 7 | if-1-7.core1.newyork.teleglobe.net |
| 8 | ix-1-8.core1.newyork.teleglobe.net |
| 9 | jfk-core-02.inet.qwest.net |
| 10 | jfk-core-01.inet.qwest.net |
| 11 | chi-core-01.inet.qwest.net |
| 12 | chi-edge-01.inet.qwest.net |
| 13 | ar-chicago-cern.ch |
| 14 | cernh9-pos500.cern.ch |
| 15 | in2p3-fddi.in2p3.fr |
| 16 | lyon-inter.in2p3.fr |
| 17 | lyon-tif.in2p3.fr |
| 18 | grame-cisco.in2p3.fr |
| 19 | bach.grame.fr |

The intended solution consists in peak filtering and in computing the convergence point of the remaining values. We applied a FTMA derived algorithm to the measured latency variation in order to obtain what we call the *skew profile*.

Applied to clock synchronization, the fault-tolerant midpoint algorithm (FTMA) relies on the hypothesis that at most $k$ clocks are faulty at any resynchronization interval. At least $n = 3k + 1$ nodes are required in the system to tolerate $k$ Byzantine faults [21]. To find the clock correction term, FTMA discards the $k$ lowest and

highest clock deviations and computes the arithmetic mean of the lowest and highest remaining clock deviations [16].

Our algorithm, named peak-tolerant midpoint algorithm (PTMA) operates similarly 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 $\overline{\Delta_t} = [\delta_{t-w} \ldots \delta_i \ldots \delta_t], \delta_i \leq \delta_{i+1}$

and the midpoint latency variation is then computed as:

$$LV_t = \frac{\sum_{i=t-w+k/2}^{i=t-k/2} \delta i}{w-k} \quad (11)$$

Intuitively, the algorithm operation may be viewed as a selection on the latency variations: it discards the lowest and the highset variations, but also as a selection on the variation history as it may retain only a discontinuous subset of the sliding temporal window. Figure 4 presents the output of the algorithm applied to the previous latency variation measurement. The outlined *skew profile* represents the PTMA output.

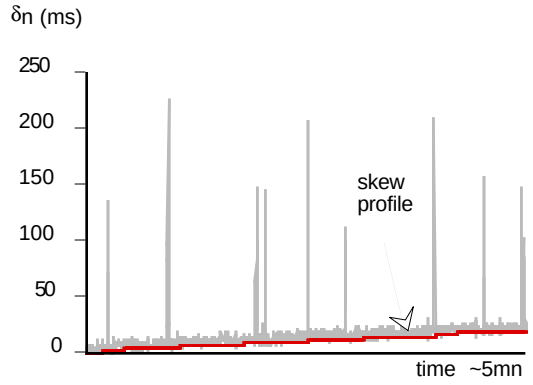


**Figure 4: PTME output**

The window size $w$ was 50 and only 10 values per window were retained. As the measurement rate was 200 ms, it means that only a discontinuous subset of 500 ms was retained over 10 seconds of measurements.

Note that the skew profile represents the clock skew but may also include possible long term changes in the transport latency, which also suits our requirements.

**3.2.2. Skew Profile Smoothing.** However, with the previous example, the transmission conditions was rather good compared to other measurements (using alternate Internet service providers and therefore other network paths) which showed more chaotic transmission conditions, with peak latencies over 3 sec and a constant range width varying up to 200 ms during periods larger than the PTMA window size.

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:

$$LV_{EPTMA}^{t} = \alpha . LV_t + (1-\alpha) . LV_{EPTMA}^{t-1} \quad (12)$$

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

Choosing a small $\alpha$ value gives more weight to the passed values and minimizes the effect of any pulse while this effect is more persistent. Assume that the latency has a constant value $L$. At time $t$, it changes by stage to a value $L'$. Let $R$ be the ratio between $L$ and $L'$. At $t+\theta$, the ratio between the computed smoothed value and $L'$ is expressed by:

$$1 - \frac{(1-\alpha)^{\theta+1}(R-1)}{R} \quad (13)$$

Figure 5 shows the response over time for $R=1.3$ and different values of the weight factor $\alpha$.

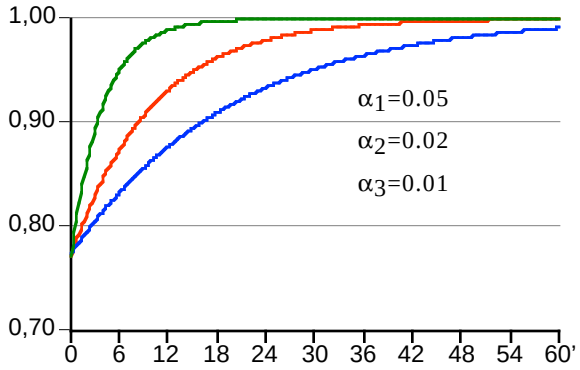


**Figure 5: Response to latency variation over time**

It appears that even with the smallest weight factor, the correction error may be considered as negligible after one minute, in regard of the time rendering requirements.

Figure 6 shows a *smoothed skew profile* made up of the worst transport latency measured. The corresponding weight factor $\alpha$ is 0.01.

The last part of the diagram shows important disturbance in the skew profile. Although this disturbance is partially reported to the smoothed values, the global shape of the skew detection is correct and the distortion introduced in time rendering is greatly minimized.

The *smoothed skew profile* resolution is the millisecond, which explains the step shape of the curve.

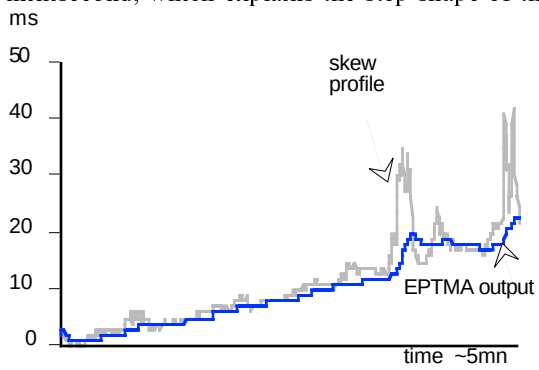


**Figure 6: EPMTA output compared to PTMA**

**3.2.3. Clock deviation correction.** As shown in the sub-section 2.2, events time rendering is based on the *apparent clocks offset* measured at protocol initialization. One host B, connected to a host A, maintains two dates: the initialization local time $B_o$ and the corresponding apparent remote time $A_o$. To compensate for the clock skew, the time rendering $r_n^i$ of an event $e_n^i$ expressed in (5) has now to be corrected as follow:

$$r_n^i = B_o + LV_{EPTMA}^n + A_n - A_o + L_{\max} + o_i \quad (14)$$

where $LV_{EPTMA}^n$ is the smoothed value of the skew profile computed at date *n*.

This is equivalent to add any change in the EPTMA output to the local initialization date $B_o$ in order to maintain the apparent clock offset constant in the peer reference time. This technic to compensate for the clock skew avoids to compute the clocks ratio and to convert the dates from one reference time to another. Moreover, the skew compensation mechanism can operate independantly of the time rendering mechanism, which greatly facilitates the protocol implementation.

## 4. Protocol implementation

The current protocol implementation transmits MidiShare events [18, 19, 20] which are high level structured events, time stamped with a millisecond resolution. Their typology includes MIDI events as well as MIDIFILE events. The underlying transport layer is the User Datagram Protocol (UDP). Two variations on the basic protocol requirements are presented, designed to optimizes LAN and WAN implementations.

### 4.1. Basic packets format

The common messages header is shown in Figure 7.

Following is a description of its fields:

- ID: a 16 bits protocol identificator intended to discriminate non-conformant packets.
- Version: a 8 bits version number, currently one (1).
- Type: a 8 bits message type identificator intended to discriminate different packet types.

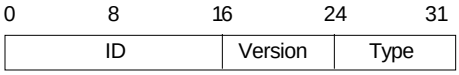


**Figure 7: Common messages header.**

The protocol relies on two packet types: *Events Packets* intended to transmit events and *ID Packets*, which purpose is both to allow a station identification on the network and to ensure the clock skew detection continuity when no events packet is to be transmitted. Both packet types are time stamped with a millisecond resolution.

**4.1.1. Events packet.** Events packet format is shown in Figure 8.

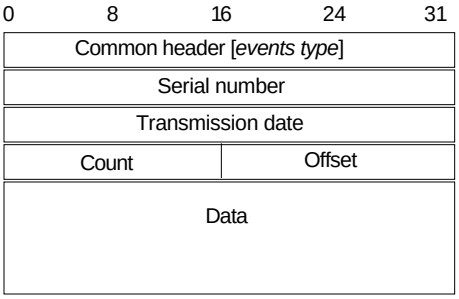


**Figure 8: Events packet format**

Following is a description of its fields:

- Common header: as described before
- Serial number: a 32 bits unique serial number, incremented at each packet transmission and intended to detect packet losses or duplicates.
- Transmission date: a 32 bits millisecond date expressed in the sender reference time.
- Count: a 16 bits data count which represents the Data chunk length.
- Offset: a 16 bits value which represents the first event start offset within the Data chunk. The offset field is commonly set to 0 but may be greater if a large event don't fit in a single packet. It allows data recovery in case of packet losses.
- Data: a variable length field which contains the transmitted events data. The only constraint on the data format concerns the events time stamping: as shown before and to allow a correct time rendering, each event should be time stamped as an offset to its packet transmission date. This time stamp is currently expressed with a millisecond resolution and 2 bytes are enough to cover the possible offset range.

The current protocol definition doesn't provide any recovery mechanism in case of packet losses.

**4.1.2. ID packet.** ID packet format is shown in Figure 9.

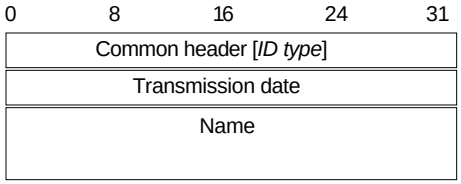


**Figure 9: ID packet format.**

As for *events packets*, the Transmission date field is a 32 bits millisecond date expressed in the sender reference time. The Name field is a variable length field containing the symbolic name of the sender.

Current use of ID packets slightly differs with the specific LAN and WAN implementations. However, its common purpose is to ensure the clock skew detection continuity.

### 4.2. LAN and WAN specific operations

The LAN and WAN implementations differ on their way to make a connection and to maintain the clock skew detection continuity. They also use additional optional packets to provide a clean connection management. Lastly and for LAN, the *grouping period* and the *maximum latency allowed* parameters are defined by default to minimize the rendering delay.

**4.2.1. Connection process.** The LAN implementation provides automatic connection process: at protocol startup, an *ID Packet* is broadcasted and then refreshed every 200 ms. Any host on the network receiving an *ID Packet* is supposed to allocate the necessary resources (if not already done) to handle the corresponding connection. When this packet disappears, the corresponding host is considered as unreachable and the connection is to be destroyed. The WAN implementation uses a TCP socket for the connection management.

**4.2.2. Skew detection continuity.** To maintain the skew detection over time, one host has to receive time stamped packets at a frequent rate. The regular broadcast of *ID Packet* on LAN is providing this continuity over time. On WAN and in order to minimize packets transmission, *ID Packet* are sent only when there is no *Events Packet* to be transmitted.

**4.2.3. Optional packets.** Two additional packets are defined to provide a clean connection management:

- a *Connection Refused Packet*: only used on WAN and via TCP, it contains the reason of the deny.
- a *Bye Packet*: it contains statistic information on the session to be closed, including the packet losses count, the maximum latency parameter value and the count of maximum latency overruns. On LAN, it is the only way to properly close a connection.

**4.2.4. Protocol parameters.** Experiments of the protocol on LAN and WAN made use of the following values:

| *param. name* | *LAN value* | *WAN value* |
|---|---|---|
| grouping period | 10 ms | 200 ms |
| maximum latency | 10 ms | 1500 ms |

The corresponding rendering delay is then 20 ms on LAN and at least 1700 ms on WAN.

The EPTMA parameters was also different for LAN and WAN experiments:

| *param. name* | *LAN value* | *WAN value* |
|---|---|---|
| window size | 10 | 50 |
| retained values | 8 | 10 |
| weight factor | 0.8 | 0.01 |

## 5. Conclusion

We have proposed a simple protocol to transmit and render time ordered events in real-time over Internet. The intended solution has many advantages: no transaction required to operate, independance of peer hosts, asymetric evaluation of the clock skew, ligthness and easyness of the implementation. However, many possible improvements have not been yet explored: in particular concerning the clock skew detection, adaptative algorithms based on PTMA and EPTMA may provide better results. Support of multicast address may also optimize the packets traffic in case of client / server uses.

The source code of the current implementation is freely available under the Library General Public License (LGPL). It is part of the MidiShare source code and can be reached at the following address:

http://www.grame.fr/MidiShare/SCPP/

## 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] D.Ferrari. Client requirements for real-time communication services. *RFC 1193 (Status: informational)* Nov-01-1990.

[2] L. Delgrossi, L. Berger. Internet Stream Protocol Version 2 (ST2) Protocol Specification - Version ST2+.*RFC 1819 (Status: experimental)* August 1995.

[3] R. Braden, Ed., L. Zhang, S. Berson, S. Herzog, S. Jamin. Resource ReSerVation Protocol (RSVP) *RFC 2205 (Status: proposed standard)* September 1997.

[4] S. Herzog. RSVP Extensions for Policy Control. *RFC 2750 (Status: proposed standard)* January 2000.

[5] M. Borden, E. Crawley, B. Davie, S. Batsell. Integration of Real-time Services in an IP-ATM Network Architecture. RFC 1821 August 1995.

[6] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. Audio-Video Transport Working Group. *RFC 1889 (Status: informational)* January 1996.

[7] J.P. Young, I. Fujinaga. Piano master classes via the Internet. *Proceedings of the International Computer Music Conference 1999.* ICMA San Francisco, 1999, pp.135-137

[8] M. Wright. Implementation and performances issues with OpenSound Control. *Proceedings of the International Computer Music Conference 1998*, ICMA San Francisco, 1998, pp. 224-227

[8] D. Fober. Real-Time Midi data flow on Ethernet and the software architecture of MidiShare - *Proceedings of the International Computer Music Conference 1994*, ICMA San Francisco, 1994, pp. 447-450

[9] J.C. Bolot. End-to-end packet delay and loss behavior in the internet. *Conference proceedings on Communications architectures, protocols and applications.* ACM, 1993, pp.289-298

[10] D.L. Mills. Network Time Protocol (Version 3) Specification, Implementation. *RFC 1305 (Status: draft standard)* March 1992.

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

[12] B. Patt-Shamir, S. Rajsbaum. A theory of clock synchronization (extended abstract). *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, 1994, pp. 810 - 819

[13] 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

[14] K. Schossmaier. An interval-based framework for clock rate synchronization. *Proceedings of the sixteenth annual ACM symposium on Principles of distributed computing*, 1997, pp. 169 - 178

[15] K. Schossmaier, B. Weiss. An Algorithm for Fault-Tolerant Clock State and Rate Synchronization. *Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems*, 1998

[16] M.M. de Azevedo and D.M. Blough, "Fault-Tolerant Clock Synchronization for Distributed Systems with High Message Delay Variation," *1994 IEEE Workshop Fault-Tolerant Parallel and Distributed Systems. (Appears in Fault-Tolerant Parallel and Distributed Systems, D. Pradhan and D. Avresky, eds., pp. 268–277, IEEE CS Press, 1995.*) Computing, vol. 27, pp. 1–14, May 1995.

[17] M.M. de Azevedo, D.M. Blough. Multistep Interactive Convergence: An Efficient Approach to the Fault-Tolerant Clock Synchronization of Large Multicomputers. *IEEE Transactions on Parallel and Distributed Systems 9(12)*, 1998, pp. 1195-1212

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

[19] D.Fober, Y. Orlarey, S. Letz. Recent developments of MidiShare - *Proceedings of the International Computer Music Conference 1996,* ICMA San Francisco, 1996, pp.40-42

[20] D.Fober, Y. Orlarey, S. Letz. MidiShare joins the Open Source Softwares - *Proceedings of the International Computer Music Conference 1999* ICMA San Francisco, 1999, pp.311-313

[21] L. Lamport, R. Shostak, M. Pease. The Byzantine Generals Problem, *ACM Transactions on Programming Languages and Systems*, Vol. 4, No. 3, July 1982, pp.382-401.