Real-Time Midi data flow on Ethernet and the software architecture of MidiShare

Dominique Fober

Real-time Midi data flow on Ethernet and the software architecture of MidiShare.

Dominique Fober
GRAME Research Laboratory, 9, rue du Garet, BP 1185, 69202 LYON Cedex 01, France
Email: fober@rd.grame.fr

Abstract

We propose a way to transmit real-time musical data flow on Ethernet. The presented implementation is based on the software architecture of MidiShare. After some reminders about Ethernet and MidiShare, we shall present an overview of the intended solution. Then we shall examine the chosen protocols, the implementation and its performance.

1 Introduction

Using a network to transmit musical events results in transporting frequently small amounts of data. With Ethernet, the problem to solve is that it is not adapted to frequent transmission of small packets and furthermore in real-time. The two reasons for this are:

- the access time of the network is not deterministic,
- Ethernet is very efficient to transmit large packets occasionally, but its efficiency decreases rapidly when the size of the packets decreases and when their frequency increase.

There are not many projects using Ethernet in real-time. Those that exist mainly concern data transmission, voice transmission or mixed voice and data transmission particularly in an industrial environment, where transmission delay can be critical. The model we propose here is based on the IEEE 802.3 protocol which means that it can work efficiently on any network supporting the CSMA/CD protocol. Our method relies on the results of previous work concerning real-time data flow on Ethernet. Our goal with this implementation is to extend the communication scheme of musical applications, allowing them to share remote resources (hardware as well as software). Later this system will become the basis of our hierarchical real time inter-application communication system [Orlarey, 1991].

2 Backgrounds

2.1 Ethernet

Ethernet is based on the CSMA/CD protocol (Carrier Sense, Multiple Access / Collision Detection) as defined by the IEEE 802.3 standard [Millet, 1986]. This protocol allows free access to all the stations on the network and in respect of its rules, they can transmit a message without having to be invited.

2.1.1 Transmission mechanisms

The network is characterized by its flow and the transmission speed of its medium. The *time slot* represents the double of end to end course time on the network. A packet transmission on the bus can be detected by its corresponding state transitions, we speak then of the presence of a *carrier* . Every station on the network is capable of simultaneously reading and writing to the bus. Carrier detection is used to prevent further packet transmission on the network and hence reduces data collision.

Packet transmission: a station ready to transmit a packet will always begin by listening to the bus, if it detects a carrier, it delays its transmission until the bus is silent. When the bus becomes free, the station can then begin to transmit. The packet is sent as a binary string and propagates on the bus in the two directions from the connection point.

Collisions detection : if several stations are queued to transmit, when the carrier disappears, they will simultaneously begin to transmit and their packets will interfere on the bus, this is called a collision. Stations detect collision by comparing the value of the bit they write to the bus with the one they read from the bus, a difference indicates that their packet has been damaged. To be sure to detect all the collisions, the minimum duration of a message transmission needs to be equal or greater than the time slot.

Collisions solving: when several stations detect a collision, they first enforce it to be sure that all stations on the network will detect it, they then stop their transmission and calculate a random retry slot according to a recent history of collisions. Several algorithms permit the control of these retry delays so as to optimize network efficiency.

2.1.2 Bus states

A bus can take two states:

- a productive state: that is, when all the periods of data transmission are successful.
- an unproductive state: that is all the periods of competition between stations to acquire exclusive use of the bus.

2.1.3 Transmission periods

Packet transmission consists of two periods:

- a contention period: of duration equal to the time slot in which a collision can happen.
- a productive period: that succeeds to the previous one. In fact, after the contention period, all the stations on the network listen to the carrier. If no collision happened during the contention period, then the transmitting station is sure to be able to transmit its packet correctly.

2.2 MidiShare

MidiShare is a real-time multi-tasking Midi operating system specially devised for the development of musical applications [Orlarey and Lequay, 1989]. MidiShare is based on a client/server model. It is composed of four main components : an event manager, a time manager, a task manager and a communication manager. The event manager provides an uniform way to process and store midi events. The time manager and the task manager control the real-time

behavior of applications. The communication manager is in charge of both inter-applications and midi communications.

Communication is based on high level events instead of midi byte packets. These events support fully the Midi and the Midi File standard. Their semantic can be easily extended for the purpose of any client application.

The heart of MidiShare is a real time scheduler which allows events and function calls to be scheduled in the future. This powerful mechanism considerably simplifies the task of creating applications which need to operate with real-time. A proprietary scheduling algorithm ensures a very low constant time scheduling cost per event, even when the scheduler is heavily loaded [Orlarey, 1989].

3 Overview of the intended solution

The data flow on the network is made of millisecond dated events. These events are gathered into packets which are then transmitted on the network. Ideally, any station should be able to receive events as soon as they are sent and to render them with the same scheduling. Actually due to the CSMA/CD protocol, the network transmission always introduces a variable delay corresponding to the sum of the physical transmission time and the network acquisition time. Acquisition time is the sum of the waiting time for the network to become free and the possible contention periods in the case of collision. The physical transmission time is considered to be insignificant. We shall further refer to this total transmission delay as transmission time.

The network efficiency (defined as the ratio between productive periods and the sum of productive periods and contention periods) decreases when the traffic increases but above all, when the size of the transmitted packets decreases [Metcalfe and Boggs 1983]. Taking account of this fact, we introduce a second transmission delay, the grouping period which corresponds to the time during which we can reasonably differ the data transmission to avoid overloading the network and to increase the packets size. So the total transmission delay is made up of two successive periods: the grouping period followed by the transmission time. Events to be transmitted are accumulated during the grouping period before being sent on to the network.

Although Ethernet has a high probability of transmission success there is no protocol to guarantee that a packet will arrive at its destination and arrive undamaged. It is the applications responsibility to implement the necessary protocol to ensure their required security level. This distribution of the transmission control on the network allows an independent packet sizing protocols and in our case allows the use of a low time cost protocol which avoids overloading the network with unnecessary acknowledgments.

Each station on the network is known by all the others and can directly address them by using their Ethernet address, so that each connection can be independent from all the others. A station can also simultaneously send a message to all the others, by using a multicast address.

4 Protocols

We are using IEEE 802.2 Type 1 packets which include a protocol number. At reception time, the Ethernet driver reads this number and checks to see if any client has a reading function pending for it. If not, the driver discards the packet. We use 3 different protocol numbers to transmit small events, large events and stations management packets.

4.1 Station management

Every station on the network is connected to all the others. The difference with the standard use of the network (file transfer for example) is that a real-time data flow is endless and the corresponding connections needs to be maintained in time. We use a particular protocol to manage the inter-station connections, defined by 4 packets types:

- 'NEW' packet: sent to all stations by every new station on the network to notify its presence.
- 'MACH' packet: sent as reply by every station that received a 'NEW' packet for identification.
- 'QUIT' packet: sent to all stations by every station that leaves the network.
- 'CTRL' packet: regularly sent by every station to all the others. It permits control of the state of the connection between stations. If for one of the stations this packet disappears, the others will consider that the corresponding station is not available any more.

Ethernet addresses for all stations on the network are collected at identification time.

4.2 Small events transmission

We consider events smaller than 6000 bytes, i.e. those that fit into 5 packets, to be small events. Small event transmission uses only one packet type: small events packets. Each packet includes a unique serial number particular to each connection on the network. This number increases with each packet transmission on this connection and the receiver can detect transmission failures by comparing the current serial number to the last one received. In the current implementation, there is no recovery sequence for lost packets.

4.3 Large events transmission

For large event transmission, we need to ensures as much security as synchronization between the transmitter and the receiver. All packets belonging to a large event are transmitted one after another possibly up to the full network flow rate. A minimum protocol ensures that if the receiver is busy with other tasks at transmission time, it is able to enforce its own rate to receive and process the packets. The large events transmission protocol uses 3 packet types:

- large events packets
- acknowledgment packets
- transmission cancel packets

This protocol simply consists of:

- for the receiver: to send a acknowledgment for each received packet.

- for the transmitter: to send the next large event packet when the acknowledgment corresponding to the last packet is received. If it does not receive an acknowledgment after one retry the process is canceled (see fig. 1).

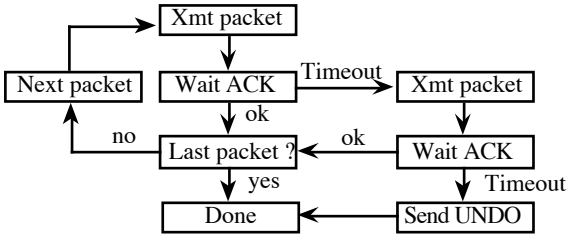

fig. 1: large events transmission protocol

4.4 Consensus on transmission delay

As previously defined, the transmission delay is the sum of the grouping factor and the transmission time, it has a fixed maximum value, given by the maximum acquisition time allowed. All the receivers on the network must take account of this value to accordingly adjust their event time rendering.

5 Implementation

5.1 Architecture

The general architecture of our implementation is as shown in figure 2. It includes a network driver and a remote controller for every station on the network:

- the network driver is in charge of the transmission and reception of packets, as well as their dispatch to the corresponding remote controllers, according to their Ethernet source address.
- the remote controller includes a time renderer for scheduling the received events, an input stream sampler to collect events ready to be sent, a packet assembler to gather them into packets and a trigger to launch the sampler. The trigger is set by the assembler at transmission time with the fixed value of the grouping period.

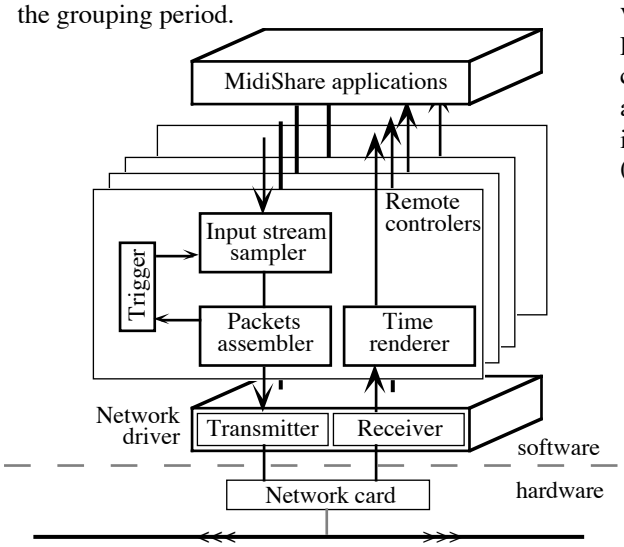

fig. 2: general architecture

This mechanism leads towards a time multiplexing of the transmission channel. If a collision occurs during a packet transmission and if the two stations involved try to transmit their next packet at the same fixed time after this collision, there is great probability that a new collision will occur. Setting the trigger after a successful channel acquisition forces the transmission dates to diverge from possible collision dates.

The dispatch of the events to the different MidiShare applications or to the physical Midi ports is handled by the MidiShare driver. This is done according to the current connections between the remote controllers and the MidiShare applications. In fact, to the network user , the remote controllers appear just like any other MidiShare application .

5.2 The different packets

All the packets have a common header consisting of the Ethernet address of the source and destination stations, and a protocol number. The common header includes all the necessary information to identify a station on the network.

- Events packets: small and large events packets differ only by their protocol number. They are made up of the common header followed by the real transmission date, the packet serial number, the packet scheduling date (use to measure acquisition times), its corresponding grouping period and then the data.
- Stations management packets: these have a particular protocol number and consist of a common header, a 4 bytes message type and corresponding data.

5.3 Time rendering: reference conversion

All the transmitted events are MidiShare events and include a date expressed in MidiShare time i.e. milliseconds. The applications in charge of transmitting and receiving these events are MidiShare applications and so, share the same clock as the events. A problem arises here due to the shift of the different clocks on the different stations and the lack of synchronization between them. As MidiShare allows us to schedule events in the future, we just need to convert the events dates from the transmitter time reference to the receiver and to delay them for a correct time rendering.

Let P be a packet transmitted on the network. P contains its real transmission date D_e, its duration L and events e_1 to e_n. The maximum acquisition time k is implicitly known by all the stations on the network (see fig. 3).

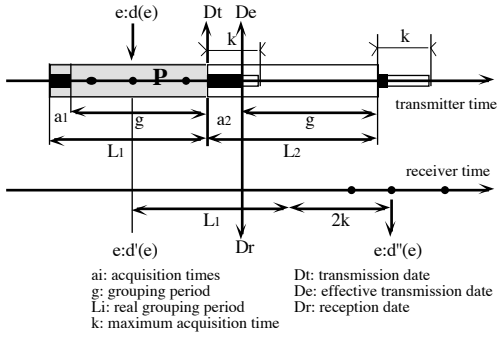

fig. 3: time rendering

Using the consensus on transmission delay, an event e will be scheduled in the receiver time reference for the date:

$$d@@(e) = d@(e) + L + 2k \quad \text{where} \quad d@(e) = d(e) + Dr - De$$

where D_r is the packet reception date expressed in the receiver reference and k is the maximum transmission delay.
Event dates are converted to the destination station reference time by adding the offset between the two station references. We then add the packet duration L to move this event into the present time. Finally, adding two maximum acquisition times makes up for the acquisition time delays and allows a correct time rendering of the transmitted events. The real grouping period is variable and depends on the acquisition time.

Another solution would be to regularly schedule each transmission at multiple dates of a fixed grouping period g. This has the advantage of reducing the transmission delay because we just need to add 1 maximum acquisition time (instead of 2) to make up for acquisition time delays, however this technique does not allow channel multiplexing and so reduces network efficiency.

6 Performances

Hutchinson and Merabti [1987] showed that if the network load is lower than a given threshold, acquirement time will always fit in a given period. Their results come from a network behavior simulation. For n stations transmitting a 512 bit packet every 100ms, they got a maximum acquisition time variation from 7,5 to 28ms for a network load variation of 10 to 90%. From 90 to 98%, this time increases up to 70ms and beyond 98%, it overruns 100ms.

6.1 Theoretical behavior

We shall further refer to the *reference data flow* as a 1000 note events per second data flow (each note corresponds to one key on and one key off).
The grouping period has a direct effect on the packets size. For a reference data flow this size varies from 510 to 1150 bytes when the grouping period varies from 30 ms to 70 ms. It also has a direct effect on the resulting network flow rate: in fact, to transmit a reference data flow, the real flow rate on the network will be:

$$\frac{(h+16g)\times 8}{g}$$

where h is the packet header size and g the grouping period. So, for a 10 mbps system, the theoretical count of stations that can simultaneously transmit a reference data flow is as shown in the table 1 below:

gr. period	5	10	20	30	50	70
n stations	57	66	71	74	75	76

table 1: count of possible stations according to the grouping factor

For our purpose, we choose the values of 50 ms for the grouping period and 30 ms for the maximum transmission delay so that events can be rendered with a 110 ms delay or 80 ms without time multiplexing.

6.2 Practical measures

The following results are incomplete, due to a lack of stations to make extensive benchmarks, but they clearly show the efficiency of our solution.

6.2.1 Network efficiency

To evaluate the real-time data transmissions influence on the other services of the network, we measured file transfer times with an increasing number of stations transmitting a reference data flow. We used 8 different stations: 2 of them were dedicated to file transfer only, another one to time measurement, 5 stations were dedicated to real-time transmissions. Files sizes varied from 100k to 1Mb. With up to 5 stations transmitting a reference data flow, transfer times remained constant.

6.2.2 Acquisition times

For our measurements we consistently used 7 stations. Each of them was transmitting and receiving one or more reference data flow. We have been able to load the network with up to 25 reference data flows. The maximum acquisition time always fits in 22ms and the median acquisition time was 3,5ms. We have not been able to increase the network load beyond this, because of the station saturation.

7 Conclusion

This model is our first step in creating a network system that takes account of time in musical data transmission. In comparison to previous works, our system adheres to both the constraints of simultaneity and time rendering without altering the CSMA/CD protocol. We hope that it will offer new prospects for musical application.

References

[Chalmtac, 1985] Chalmtac I. An Ethernet compatible protocol for real-time voice/data integration. Computer networks and ISDN systems, 1985, Vol 10, N°2, pp. 81-96.

[Hoffmann and Kersting, 1984] Hoffmann W., Kersting T. *Simulation von Ethernet unter Echtzeitbedigungen*. Regelungstechnische Praxis, 1984, Vol 26, N°11, pp. 486-491.

[Hutchinson and Merabti, 1987] Hutchinson D., Merabti M. *Ethernet for real-time applications*. IEE proceedings. Part E. Computers and digital techniques, 1987, Vol 134, N°1, pp.47-53.

[Maxemchuk, 1982] Maxemchuk N.F. *A variation on CSMA/CD that yelds movable TDM slots in integrated voice/data local networks*. Bell Syst. Tech. J. 1982, 61, (7), pp.1527-1550.

[Metcalfe and Boggs 1983] Metcalfe Robert M., Boggs David R. *Ethernet : Distributed Packet Switching for Local Computer Networks*. Communication of the ACM, Vol 26, N°1 January 1983, pp.90-95.

[Millet, 1986] Millet P. *Transmission et réseaux locaux, architecture I.E.E.E. 802*. Masson 1986

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

[Orlarey, 1990] Orlarey Y. *An Efficient Scheduling Algorithm for Real-Time Musical Systems*. Proceedings of the ICMC, 1990, ICMA, San Francisco.

[Orlarey, 1991] Orlarey Y. *Hierarchical Real Time Inter application Communications* Proceedings of the ICMC, 1991, ICMA, San Francisco.