DISTRIBUTED DELAY JITTER CONTROL IN PACKET-SWITCHING INTERNETWORKS†‡

Domenico Ferrari

The Tenet Group
University of California
and International Computer Science Institute
Berkeley, California, U.S.A.

Address:
Computer Science Division
EECS Department
University of California
Berkeley, CA 94720
ph. (510)642-3806
fax (510)643-7684

---

† Portions of this paper were presented at the Second International Workshop on Network and Operating System Support for Digital Audio and Video, Heidelberg, Germany, November 18-19, 1991, and published by Springer Verlag in the Proceedings of that workshop.

‡ This research was supported by the National Science Foundation and the Defense Advanced Research Projects Agency (DARPA) under Cooperative Agreement NCR-8919038 with the Corporation for National Research Initiatives, by AT&T Bell Laboratories, Hitachi, Ltd., Hitachi America, Ltd., Pacific Bell, the University of California under a MICRO grant, and the International Computer Science Institute. The views and conclusions contained in this document are those of the author, and should not be interpreted as representing official policies, either expressed or implied, of the U.S. Government or any of the sponsoring organizations.

Abstract

Delay jitter is the variation of the delays with which packets traveling on a network connection reach their destination. For good quality of reception, continuous-media (video, audio, image) streams require that jitter be kept below a sufficiently small upper bound. This paper proposes a distributed mechanism for controlling delay jitter in a packet-switching network. The mechanism can be applied to an internetwork that satisfies the conditions detailed in the paper, and can coexist with other schemes (including the absence of any scheme) for jitter control within the same network, the same node, and even the same real-time channel. The mechanism can guarantee small jitter bounds even when the clocks of the host systems and the gateways along a channel's route are only loosely synchronized; furthermore, it makes the distribution of buffer space requirements more uniform over a channel's route, and reduces by a non-negligible amount the total buffer space needed by a channel. The paper argues that, if these advantages are sufficient to justify the higher costs of the distributed jitter control mechanism with respect to a non-distributed one, it would be useful to offer to the network's users a jitter control service based on the mechanism proposed here.

Keywords: delay jitter, guaranteed performance, jitter control, packet-switching internetworks, real-time communication.

1. Introduction

Digital audio, video, and image-sequence transmission, both of the interactive and of the non-interactive kind, need a high degree of regularity in the delivery of successive stream-frames[1] to the destination. Ideally,[2] the end-to-end connections that carry those continuous-media streams should delay each element (e.g., a frame) of a stream by the same amount of time; in other terms, each connection should behave as a perfect delay line. This should normally be the case even when the elements of a stream are not transmitted at a constant rate (for example, in image browsing applications, where the user needs to be able to do pausing, fast and slow forwarding, fast and slow rewinding, and so on).

---

[1] We call *stream-frames* the data elements (video frames, audio samples, image frames) that constitute a continuous-media stream.

[2] We ignore here the case of stored streams that are shipped in batches and are to be played at the destination upon reception, as well as all the other cases in which input patterns are not to be faithfully reproduced at the output of the network. These either must be regarded as file transfers (i.e., not as continuous-media transmissions) from a storage device to a remote one, or require a receiver that will time their presentation following a given (e.g., periodic) time law. Some form of jitter control may be needed even in these cases.

An ideal constant-delay network could be characterized also as a *zero-jitter* one, where jitter (more precisely, *delay jitter*) is any variation of stream-frame delays. A more realistic requirement for continuous-media transmission is that jitter have a small upper bound. Given an upper bound for delays, which is necessary for interactive continuous media and often desirable even in non-interactive cases, assigning a jitter bound is equivalent to specifying also a lower delay bound fairly close to the upper bound. A jitter bound alone will not guarantee the faithful reproduction of the input pattern at the output. To achieve this result, such a bound will have to be accompanied by a delay bound guarantee.

The computers and the (packet-switching) networks that connect users involved in continuous-media communications may introduce large amounts of jitter, due to the fluctuations of their loads. End-to-end delay bounds can be guaranteed by suitable design of the various components of a connection, including the host computers at the two ends; for instance, a scheme for establishing real-time connections with deterministic or statistical delay bounds, as well as throughput and loss probability bounds, in a packet-switching network is described in (Ferrari and Verma, 1990a). Is it possible to reduce (below a small upper bound) end-to-end delay jitter as well?

The answer is affirmative, as long as (*i*) a bound on delay can be guaranteed, (*ii*) a sufficiently large amount of buffer space is available at the destination or close to it, and (*iii*) the clocks of the source and the destination are kept in synchrony with each other (Partridge, 1991). This jitter control scheme is very simple: the source timestamps each stream-frame, and the destination, knowing the timestamp and the ideal constant delay (which must be greater than or equal to the delay bound guaranteed by the network), buffers each stream-frame until the time it has to be delivered to the receiving user.

The scheme can be implemented at the application level or at a lower level (e.g., at the transport or even the network layer in the communication protocol hierarchy); it is easy to see that the existence of a jitter control mechanism at a lower level, while not automatically providing low jitter to the application, greatly facilitates the achievement of this objective. Essentially, a bounded-jitter packet stream is not a necessary, but may be regarded as a sufficient, condition for obtaining a bounded-jitter sequence of stream-frames, as long as stream-frame reassembly, higher-level protocol processing, and stream-frame delivery to the application do not introduce appreciable amounts of jitter.

An alternative approach to jitter control has been presented in (Verma *et al.*, 1991). While the approach described above (see for example (Partridge, 1991)) operates primarily at the destination, the one proposed in (Verma *et al.*, 1991) is distributed; thus, it can only be implemented by the network layer, not by the upper protocol layers or by the application. In the kinds of networks in which it can be used, the distributed approach has been found to require a substantially smaller total amount of buffer space, and to yield a more uniform buffer allocation along the path of a bounded-jitter connection. In this paper, we introduce a distributed scheme that can be applied to much more complicated networks (in fact, to internetworks), analyze its robustness with respect to clock drift, and argue that, if a network already implements distributed jitter control for its internal purposes, it could, at almost no extra cost, offer a jitter control service to its users; indeed, if the receiving systems can be designed to inject only negligible amounts of jitter into the incoming streams, this service would provide the end-to-end jitter bounds needed by continuous-media streams.

A more detailed discussion of the two approaches (distributed and non-distributed) to jitter control can be found in Section 7. That discussion is preceded by a description of the network environment in Section 2, a presentation of our scheme in Section 3, and an analysis of the effects of loose clock synchronization on jitter control in Section 4. Section 5 summarizes the scheme proposed in (Verma *et al.*, 1991), and discusses its possible uses in conjunction with the new scheme or alone in networks that do not obey the assumptions on which it is based. The use of our distributed scheme in an internetworking testbed currently under development is examined in some detail in Section 6.

The rest of this section summarizes the approach to real-time communication presented in (Ferrari and Verma, 1990a), which, like the jitter control scheme in (Verma *et al.*, 1991), applies to what we will in the sequel call a *simple network*. Such a (packet-switching) network is characterized by hosts and switches interconnected by physical links in an arbitrary topology. The switches operate according to the store-and-forward principle, the links introduce only propagation delays, and the output queues of a switch (one per output link) are managed by a multiple-class version of the Earliest Due Date (EDD) (Liu and Layland, 1973) discipline.

A path from source to destination in such a network may be depicted as in Figure 1. Nodes 2, 3,... N-1 represent the switches traversed by the path. Node 1 and node N implement the protocol layers below the transport layer on the host systems. For these systems to offer end-to-end delay guarantees, they must be scheduled by a policy (such as Multi-Class EDD) that gives higher (and preemptive, when necessary) priority to the real-time activities of the upper protocol layers, as well as to the output queues of the "resident network node" reserved for real-time packets.

The basic abstraction the scheme in (Ferrari and Verma, 1990a) is capable of implementing is the *real-time channel*. A real-time channel is a simplex connection between a sender and a receiver[3] characterized by performance guarantees. The performance

---

[3] The notion of real-time channel can be extended to the case of multiple receivers. For the sake of sim-

guarantees a client may request include a delay bound D for each packet; on a *deterministic channel*, packets are guaranteed to arrive at the destination with a delay not larger than D.

Besides specifying performance requirements such as D, a client will have to provide a simple description of the maximum traffic on the channel to be established. One of the parameters this description consists of in the approach we are summarizing here is $x_{\min}$, the minimum time between successive packets at the entrance to the channel.[4] This will also automatically specify the client's maximum bandwidth requirements.

The creation of a real-time channel may be requested by a client running on the host system that should become the channel's source. An establishment message carrying the performance requirements and the traffic specifications given by the client leaves the source and, while building the connection, tests the ability of each switch to handle the new channel without violating the bounds promised to the other clients; it also collects information, to be used by the destination, on the loading of each switch. If the message reaches the destination without having been rejected, the destination runs the final tests; if also these tests are successful, the destination subdivides the client-given bounds into per-switch bounds, and sends them to the switches with the channel acceptance message. Thus, switch n receives, for example, the value of its local delay bound $d_{i,n}$ for the new channel i, and will use it to compute the local deadline of each packet transmitted on that channel.

2. The Network

In this section, we describe the environment for which our jitter control scheme has been designed; in particular, the class of networks (to be more precise, internetworks) that can guarantee delay jitter bounds, as well as delay bounds, using the approach to be presented in Sections 3 and 4 (while the method described in Section 5 applies only to a simple network). The scheme described in (Ferrari and Verma, 1990a) for simple networks can be extended to a class of internetworks (Ferrari, 1991), which is the one we consider here. The rest of this section gives a brief description of the characteristics of that class.

---

plicity, we will not consider such an extension in this paper.

[4] The client will actually provide the value of the minimum time between successive stream-frames; this will be translated into the minimum interpacket time by the channel establishment protocol.

We consider an internetwork in which it is possible to create real-time channels. We assume that the internetwork offers channels with deterministic (i.e., absolute) delay bounds, and we wish to devise a way to obtain guaranteed jitter bounds for the same channels when the clients need such guarantees.

The path of a channel from the sending host to the receiving host will generally traverse a number of different networks, interconnected by gateways.[5] We consider a hierarchical model of the internetwork in which the level 1 nodes are the gateways. The level 1 path of a channel is a path consisting of gateways interconnected by *logical* links; each one of these links may represent a path through the network of which the two nodes at its extremes are gateways. This path may be very complicated; for example, the network in question might in turn be an internetwork, whose gateways constitute a level 2 network, and so on recursively.

In this hierarchical model, a path through a network may be abstracted by transforming it into a simple logical link at the immediately higher level (note that in this paper higher levels are labeled with lower numbers). This process of abstraction ends with the level 0 path, which is a single logical link connecting the sender and the receiver. This is the logical link whose delay and delay jitter are to be bounded. What are the minimum necessary properties of the underlying internetwork that allow the bounds selected by the client to be guaranteed, if possible?[6] What must we assume about the various networks constituting the hierarchy?

If all the constituent networks were capable of guaranteeing delay and jitter bounds, the internetwork could provide similar guarantees at the end-to-end level. However, such a restrictive condition is by no means necessary. All we need (see (Ferrari, 1991)) is the assumption that the delay on each level 1 logical link be bounded or at least boundable (with a sufficiently small bound), and that those bounds be either known *a priori* or obtainable by a recursive application (at the various levels in the hierarchy) of an extension of the scheme described in (Ferrari and Verma, 1990a).

We also assume that level 1 nodes are store-and-forward, and schedule packets for transmission on each of their output links in such a way that, if the client’s request for the channel is accepted, it is possible to guarantee the end-to-end delay bound specified by the client.[7] Many scheduling algorithms satisfy the latter assumption (see (Ferrari,

---

[5] We use this term in the generic sense of "a computer system interconnecting two distinct networks at the network or higher layer;" thus our "gateways" may be routers as well as gateways proper.

[6] If it is not possible to provide the requested guarantees, the client’s request will be rejected.

[7] Note that no assumptions are made about gateways or routers at lower levels in the hierarchy. The only constraint on them is that they allow level 1 delays to be boundable.

1991)); for example, round-robin and its extensions and modifications (Fair Queueing (Demers *et al.*, 1989), Hierarchical Round-Robin (Kalmanek *et al.*, 1990)), deadline-based disciplines (Earliest Due Date or EDD, Multi-Class EDD (Ferrari and Verma, 1990a)), FCFS, and others.

If these assumptions, which are summarized in Figure 2, are satisfied, the internetwork can provide guaranteed delay bounds (Ferrari, 1991), and we assume that the internetworks in the class we are considering indeed do provide them.

For convenience, we break the delay bound $dl_{i,l}$ of a level 1 logical link l for channel i into the sum of two terms, a fixed delay $df_{i,l}$ and a maximum variable delay $dv_{i,l}$, where $df_{i,l}$ is the delay encountered by the smallest possible packet travelling on the logical link under zero-load conditions, and $dv_{i,l}$ is an upper bound to the variable components of packet delays along the same link. Thus, $df_{i,l}$ includes propagation, switching, and transmission delays for the minimum-size packet, while $dv_{i,l}$ primarily consists of queueing delays and the additional transmission delays that a maximum-size packet would incur. The actual delay of a packet on that channel along that level 1 logical link will be between $df_{i,l}$ and $dl_{i,l}$.

As in a simple network, each level 1 node traversed by channel i has a delay bound $d_{i,n}$ assigned to it by the destination at channel establishment time. This bound is the maximum residence time of a packet in node n. The minimum residence time is the packet's processing time in the node (which includes also its transmission time on the output link); we call $dp_{i,n}$ the smallest of these times, corresponding to the idle-node residence time of the smallest packet the channel will carry; by $dq_{i,n}$ we denote the maximum queueing time in the node, which can only be reached by the smallest packet, since for all packets the sum of processing time and queueing time will be at most $d_{i,n}$. Thus, we have $d_{i,n} = dp_{i,n} + dq_{i,n}$.

3. Jitter Control with Perfectly Synchronized Clocks

In an internetwork of the class defined in Section 2, let us assume that the clocks of all level 1 nodes are in perfect synchrony with each other. That is, simultaneous readings of these clocks are always identical. A reasonably good approximation to this ideal situation can be achieved by having the gateways run a clock synchronization protocol (e.g., NTP (Mills, 1989)), or listen to the times broadcast by a satellite and adjust their clocks accordingly. In the level 1 path of a real-time channel, all components of the end-to-end delay bound are known after the establishment of the channel. Each logical link l of channel i has a total delay bound equal to $df_{i,l} + dv_{i,l}$, and each node n has a local bound

$d_{i,n}$ assigned to it by the destination (see Section 1 and (Ferrari and Verma, 1990a)). Thus, it is easy to calculate the maximum delay between a node and the next, and to compensate for the jitter due to the variable parts of the delay, which may make the actual delay shorter (sometimes much shorter) than the maximum.

The mechanism is very simple: it requires timestamping each packet's header by the source host, reading the clock at a packet's arrival at each level 1 node, and keeping the packet until the *canonical* (i.e., zero jitter) arrival time at that node before giving it to the node's scheduler. Each node is informed, when a real-time channel is created, about the maximum delay that separates it from the source for the packets on that channel; this is the *canonical delay* $D^*_{i,n}$ for channel i at node n. Assuming that the clock is read in each node immediately at packet arrival time, the maximum delay with respect to the canonical arrival time at the previous node (node $n-1$) equals $df_{i,l} + dv_{i,l} + d_{i,n-1}$.

Given the time of departure from the source, which is carried by the header, the node calculates the canonical arrival time and compares it with the actual one (which is guaranteed not to exceed the canonical time). The packet will be kept in storage in the node for a time interval equal to the difference between the canonical and the actual arrival times. When that interval expires, the packet becomes *schedulable*; the jitter introduced by the scheduler in node n will be corrected exactly by the same mechanism in node $n+1$. The only jitter that cannot be corrected in this way is therefore the one introduced by the last node on the path. This will have to be corrected by the receiving host, which will need to know the maximum delay assigned to the last node and keep the packet until that bound is reached before giving it to the application. If this is done, a zero-jitter (perfect delay line) transmission is obtained; if it is not done, the only end-to-end jitter is that generated by the last node, since those produced by all previous nodes and (logical) links are perfectly compensated for by the simple mechanism we have just illustrated.

What is the maximum value of jitter when no jitter control is used, and how much is that value reduced by the above scheme? To answer these questions, we refer to the channel whose path is shown in Figure 1.[8] We therefore omit the channel index i in the symbols we will use, and assign to each link the same number as the node immediately downstream.

---

[8] The nodes in the figure are level 1 nodes, which run an internetwork protocol. Node 1 represents the network layer and lower layers on the source computer, and shares that computer with source S, which represents the application and the protocols at and above the transport layer.

Without jitter control, the link-node pair numbered n contributes to the total maximum jitter the quantity

$$j_n = dl_n + d_n - dp_n - df_n = dv_n + dq_n, \tag{1}$$

Equation (1) can be explained by observing that the maximum delay through the link-node pair is $dl_n + d_n$, whereas the minimum delay is $dp_n + df_n$.

Thus, the maximum jitter for the packets on the channel is

$$J_{\max} = \sum_{n=1}^{N} j_n = \sum_{n=1}^{N} (dv_n + dq_n). \tag{2}$$

From the above discussion, we have that the maximum jitter permitted by the jitter control mechanism described in this section is[9]

$$J'_{\max} = d_N - dp_N = dq_N. \tag{3}$$

Note that the maximum jitter at the input to node n is

$$j'_{in,n} = dv_n + dq_{n-1}, \tag{4}$$

while at the output of the same node it is

$$j'_{out,n} = dq_n. \tag{5}$$

These values must be compared with

$$j_{in,n} = dv_n + \sum_{m=1}^{n-1} j_m \tag{6}$$

and

$$j_{out,n} = \sum_{m=1}^{n} j_m, \tag{7}$$

respectively. The value of $j'_{in,n}$ equals the maximum amount of time a packet may have to wait in the node before becoming schedulable. Thus, the maximum residence time of a packet in node n with distributed jitter control is

$$r'_n = dv_n + dq_{n-1} + d_n, \tag{8}$$

and only

$$r_n = d_n \tag{9}$$

without distributed jitter control; in the last node, however, if jitter is to be absorbed by

---

[9] We use primed symbols to refer to cases in which our distributed jitter control mechanism is present.

buffering (a network-layer implementation of the non-distributed approach to jitter control described in Section 1), the maximum residence time must equal the total maximum jitter instead of d_N as dictated by (9):

$$r_N = \sum_{n=1}^{N} j_n \tag{10}$$

To calculate the number of packet-sized buffers needed by each channel in node n to avoid packet losses due to buffer overflows, we observe that the worst case corresponds to a stream of packets entering the channel at node 1 with interpacket distances equal to $x_{\min}$. The latest time a packet from the stream can leave node n equals the sum of its departure time from the source, the canonical delay at node n, and the local delay bound in the same node. Similarly, the earliest time a packet can arrive at node n is given by its departure time from the source plus the canonical delay at node n minus the maximum jitter at the input to the same node. If we number the packets in the stream 1, 2, ..., and we assume that node n has allocated B_n buffers to the channel, the packet numbered 1 will have to have left the node before the arrival of the packet numbered $B_n + 1$ even in