Jitter Control in QoS Networks*

Yishay Mansour
Dept. of Computer Science
Tel-Aviv University
and
AT&T Research Labs
mansour@research.att.com

Boaz Patt-Shamir†
Dept. of Electrical Engineering
Tel-Aviv University
and
Northeastern University
boaz@eng.tau.ac.il

December 17, 1998

Abstract

We study jitter control in networks with guaranteed quality of service (QoS) from the *competitive analysis* point of view: We propose on-line algorithms that control jitter and compare their performance to the best possible (by an off-line algorithm) for any given arrival sequence. For *delay jitter*, where the goal is to minimize the difference between delay times of different packets, we show that a simple on-line algorithm using buffer of B slots guarantees the same delay-jitter as the best off-line algorithm using buffer space $B/2$. We prove that the guarantees made by our on-line algorithm hold even for simple distributed implementations, where the total buffer space is distributed along the path of the connection, provided that the input stream satisfies a certain simple property. For *rate jitter*, where the goal is to minimize the difference between inter-arrival times, we develop an on-line algorithm using a buffer of size $2B + h$ for any $h \geq 1$, and compare its jitter to the jitter of an optimal off-line algorithm using buffer size B. We prove that our algorithm guarantees that the difference is bounded by a term proportional to B/h.

*Research supported in part by a grant from the Israel Ministry of Science. A preliminary version appeared in the proceedings of the "39th Annual Simposium on Foundation of Computer Science (FOCS)", pages 50–59, 1998.

†Supported in part by a special grant from the Government of Israel.

1 Introduction

The need for networks with guaranteed quality of service (QoS) is widely recognized today (see, e.g., [8, 11]). Unlike today's "best effort" networks such as the Internet, where the user has no guarantee on the performance it may expect from the network, QoS networks guarantee the end-user application a certain level of performance. For example, ATM networks support guaranteed QoS in various parameters, including end-to-end delay and delay jitter (called Cell Transfer Delay and Cell Delay Variation, respectively [5, 12]).

Jitter measures the variability of delay of packets in the given stream, which is an important property for many applications (for example, streaming real-time applications). Ideally, packets should be delivered in a perfectly periodic fashion; however, even if the source generates an evenly spaced stream, unavoidable jitter is introduced by the network due to the variable queuing and propagation delays, and packets arrive at the destination with a wide range of inter-arrival times. The jitter increases at switches along the path of a connection due to many factors, such as conflicts with other packets wishing to use the same links, and non-deterministic propagation delay in the data-link layer.

Jitter is quantified in two ways. One measure, called *delay jitter*, bounds the maximum difference in the total delay of different packets (assuming, without loss of generality, that the abstract source is perfectly periodic). This approach is useful in contexts such as interactive communication (e.g., voice and video tele-conferencing), where a guarantee on the delay jitter can be translated to the maximum buffer size needed at the destination. The second measure, called *rate jitter*, bounds the difference in packet delivery rates at various times. More precisely, rate jitter measures the difference between the minimal and maximal inter-arrival times (inter-arrival time between packets is the reciprocal of rate). Rate jitter is a useful measure for many real-time applications, such as a video broadcast over the net: a slight deviation of rate translates to only a small deterioration in the perceived quality.

Another important reason for keeping the jitter under control comes from the network management itself, even if there are no applications requiring jitter guarantees. For example, it is well known that traffic bursts tend to build in the network [8, 15]. Jitter control provides a means for regulating the traffic inside the network so that the behavior of internal traffic is more easily manageable. A more subtle argument in favor of jitter control (given by [17]) proceeds as follows. When a QoS network admits a connection, a type of "contract" is agreed upon between the network and the user application: the user is committed to keeping its traffic within certain bounds (such as peak bandwidth, maximal burst size etc.), and the network is committed to providing certain service guarantees (such as maximal delay, loss rate etc.). Since the network itself consists of a collection of links and switches, its guarantees must depend on the guarantees made by its components. The guarantees made by a link or a switch, in turn, are contingent on some bounds on the locally incoming traffic. As mentioned above, unless some action is taken by the network, the characteristics of the connection may in fact get worse for switches further down the path, and thus they can only commit to lower QoS. Jitter control can be useful in allowing the network to ensure that the traffic incoming into a switch is "nicer," and get better guarantees from the switch.

Jitter control implementation is usually modeled as follows [17, 8]. Traffic incoming into the switch is input into a *jitter-regulator*, which re-shapes the traffic by holding packets in an internal buffer. When a packet is released from the jitter-regulator, it is passed to the *link scheduler*, which schedules packet transmission on the output link. In this work we focus on studying jitter-regulators.

Nature of our results. Before we state concrete results, we would like to explain the insight we seek. Prior to our work, performance of jitter control algorithm was measured either by worst-case behavior, or under statistical assumptions. Thus the properties of the algorithms were either *deterministic* (given deterministic worst-case assumptions on the input stream), or *probabilistic* (given stochastic assumptions on the input stream). In this work, we prove *relativistic* guarantees: we compare the performance of the algorithm in question to the performance of the *best possible* algorithm, which we treat as an adversary we compete against. The adversary algorithm is not assumed to be constrained by the on-line nature of the problem: it is assumed to produce the best possible output for the given input, *even if the best output may be computable only in hindsight* (hence the adversary algorithm is sometimes called the *off-line* algorithm). Algorithms whose performance can be bounded with respect to the performance of an off-line adversary are called *competitive* [10, 7, 1]. We argue that proving that an algorithm is competitive is meaningful, and sometimes superior, to proving deterministic or stochastic guarantees: first, deterministic or stochastic guarantees say nothing about the case where the underlying assumptions do not hold for some reason (even worse, the underlying assumptions—in particular, tractable stochastic assumptions—are notoriously hard to justify). On the other hand, a competitive algorithm does not assume anything about the input, and therefore its guarantees are more robust in this sense. Secondly, worst-case guarantees usually do not say much about individual cases: for example, an algorithm may be called deterministically optimal even if it performs always as bad as the worst case; competitive algorithms, by contrast, are guaranteed to do relatively well on each and every instance. Thirdly, if we add an assumption about the input sequence, the relativistic guarantee would immediately translate to a specific deterministic guarantee.

We remark that unlike conventional competitive analysis, in most cases we shall compare the performance of our on-line algorithms to the performance of an (optimal, off-line) adversary which is restricted to use *less buffer space*. For example, we prove statements such as "an algorithm Z using space B produces jitter which never more than the jitter produced by an optimal algorithm, for the given arrival sequence, using space $B/2$." One possible interpretation for this result is that algorithm Z always uses at least half of its buffer space optimally—as if it knew the future in advance.

Our Results. We consider both delay- and rate-jitter. For delay-jitter, we give a very simple on-line algorithm, and prove that the delay-jitter in its output is no more than the delay-jitter produced by an optimal (off-line) algorithm using half the space. We give a lower bound on delay-jitter showing that doubling the space is necessary. We also consider a distributed implementation of our algorithm, where the total space of $2B$ is distributed along a path. We prove that the distributed algorithm guarantees the same delay-jitter of a centralized, off-line algorithm using space B, provided that an additional condition on the beginning of the sequence is met. To complete the picture, we also describe an efficient optimal off-line algorithm. For all our delay-jitter algorithms, we assume that the average inter-arrival time of the input stream (denoted X_{a}) is given ahead of time.

One way to view the relativistic gurantee of our algorithm is the following. Assume that the specific arrival sequence is such that using a buffer of size B one can reduce the jitter comletely (i.e. zero jitter). In such a case, our online algorithm, using space $2B$ would also output a completely periodic sequence (i.e. zero jitter).

For rate jitter, we assume that the on-line algorithm receives, in addition to X_{a}, two parameters denoted $I_{\max}$ and $I_{\min}$, which are a lower and an upper bound on the desired time between consecutive packets in the output stream. The on-line algorithm we present uses a buffer of size $2B + h$ where $h \geq 1$ is a parameter, and B is such that an off-line algorithm using buffer space B can release the packets with inter-departure times in the interval $[I_{\min}, I_{\max}]$ (but the optimal jitter may be much lower). The algorithm guarantees that the rate-jitter of the released sequence is at most the best off-line jitter plus an additive term of $2(B + 2)(I_{\max} - I_{\min})/h$. We also show how can the algorithm adapt to unknown X_{a}. Finally, we prove that on-line algorithms using less than $2B$ buffer space are doomed to have trivial rate-jitter guarantees with respect to an off-line algorithm using space B.

Related Work. QoS has been the subject of extensive research in the current decade, starting with the seminal work of Ferrari [2] (see [16] for a comprehensive survey). A number of algorithms has been proposed for jitter control. Partridge [9] proposed to time-stamp each message at the source, and fully reconstruct the stream at the destination based on a bound on the maximal end-to-end delay. Verma et al. [13] proposed the *jitter-EDD* algorithm, where a jitter controller at a switch computes for each packet its *eligibility time*, before which the packet is not submitted for to the link scheduler. The idea is to set the eligibility time to the difference between maximum delay for the previous link and the actual delay for the packet: this way the traffic is completely reconstructed at each jitter node. Note that jitter-EDD requires nodes to have synchronized clocks. The *Leave-in-Time* algorithm [3] replaces the synchronized clocks requirement of jitter-EDD with virtual clocks [19]. Golestani [4] proposed the *Stop-and-Go* algorithm, which can be described as follows. Time is divided to frames; all packets arriving in one frame are released in the following frame. This allows for high flexibility in re-shaping the traffic. *Hierarchical Round-Robin* (HRR), proposed in [6], guarantees that in each time frame, each connection has some predetermined slots in which it can send packets. A comparative study of rate-control algorithms can be found in [18]. A new jitter control algorithm was proposed in [14].

Paper Organization. In Section 2 we give the basic definitions and notations. In Section 3 we study delay jitter for a single switch. In Section 4 we extend the results of Section 3 to a distributed implementation. In section 5 we study rate jitter.

2 Model

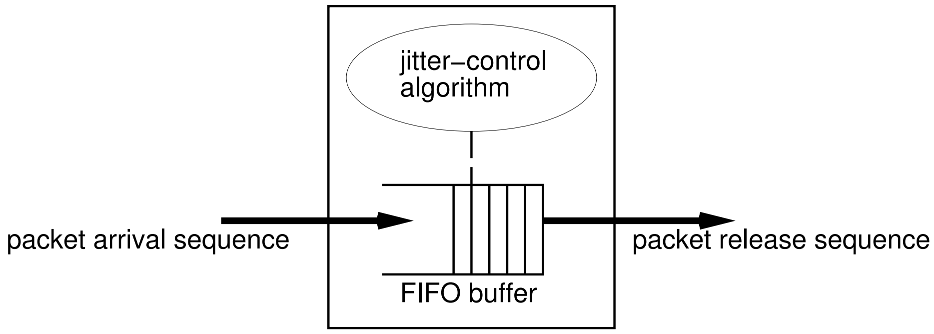

Figure 1: *Abstract node model. The jitter control algorithm controls packet release from the buffer, based on the arrival sequence.*

We consider the following abstract communication model for a node in the network (see Fig. 1). We are given a sequence of packets denoted $0, 1, 2, \ldots, n$, where each packet k arrives at time $\mathit{arrival}(k)$. Packets are assumed to have equal size. Each packet is stored in the buffer upon arrival, and is released some time (perhaps immediately) after its arrival. Packets are released in FIFO order. The time of *packet release* (also called *packet departure* or *packet send*) is governed by a *jitter control algorithm*. Given an algorithm A and an arrival time sequence, we denote by $\mathit{send}_A(k)$ the time in which packet k is released by A.

We consider jitter control algorithms which use bounded-size buffer space. We shall assume that each buffer slot is capable of storing exactly one packet. All packets must be delivered, and hence the buffer size limitation can be formalized as follows. The release time sequence generated by algorithm A using a buffer of size B must satisfy the following condition for all $0 \le k \le n$:

$$\mathit{arrival}(k) \le \mathit{send}_A(k) \le \mathit{arrival}(k+B) \ , \tag{1}$$

where we define $\mathit{arrival}(k) = \infty$ for $k > n$.[1] The lower bound expresses the fact that a packet cannot be sent before it arrives, and the upper bound states that when packet $k+B$ arrives, packet k must be released due to the FIFOness and the limited size of the buffer. We call a sequence of departure times *B-feasible* for a given sequence of arrival times if it satisfies Eq. (1), i.e., it can be attained by an algorithm using buffer space B. An algorithm is called *on-line* if its action at time t is a function of the packet arrivals and releases which occur before or at t; an algorithm is called *off-line* if its action may depend on future events too.

A *times sequence* is a non-decreasing sequence of real numbers. We now turn to define properties of times sequences, which are our main interest in this paper. Given a times sequence $\sigma = \{t_i\}_{i=0}^n$, we define its average, minimum, and maximum inter-arrival times as follows.

- The *average inter-arrival time of* σ is $X_{\mathrm{a}}^{\sigma} = \frac{t_n - t_0}{n}$.
- The *minimum inter-arrival time of* σ is $X_{\min}^{\sigma} = \min\{t_{i+1} - t_i \mid 0 \le k < n\}$.
- The *maximum inter-arrival time of* σ is $X_{\max}^{\sigma} = \max\{t_{i+1} - t_i \mid 0 \le k < n\}$.

We shall omit the σ superscript when the context is clear. The *average rate* of σ is simply $1/X_{\mathrm{a}}^{\sigma}$.

We shall talk about the *jitter* of σ. We distinguish between two different kinds of jitter. The *delay jitter*, intuitively, measures how far off is the difference of delivery times of different

[1] Note that our definition allows for 0-length intervals where more than B packets are in the system. This formal difficulty can be overcome by assuming explicitly that each event (packet arrival or release) occurs in a different time point. For clarity of exposition, we prefer this simplified model, although our results hold in both models.

packets from the ideal time difference in a perfectly periodic sequence, where packets are spaced exactly X_{a} time units apart. Formally, given a times sequence $\sigma = \{t_i\}_{i=0}^{n}$, we define the *delay jitter of* σ to be

$$J^{\sigma} = \max_{0 \le i,k \le n} \{|t_i - t_k - (i-k)X_{\mathrm{a}}|\} \ .$$

We shall also be concerned with the *rate jitter* of σ, which can be described intuitively as the maximal difference between inter-arrival times, which is equivalent to the difference between rates at different times. Formally, we define the rate jitter of σ to be

$$\max_{0 \le i,j < n} \{|(t_{i+1} - t_i) - (t_{j+1} - t_j)|\} \ .$$

The following simple property shows the relationship between delay and rate jitter.

Lemma 2.1 *Let σ be a times sequence.*

(1) The delay jitter of σ equals 0 if and only if the rate jitter of σ equals 0.

(2) If the delay jitter of σ is J, then the rate jitter of σ is at most $2J$.

(3) For all $\epsilon > 0$, and M, there exists a sequence $\sigma_{\epsilon,M}$ with rate jitter at most ϵ and delay jitter at least M.

Proof: Suppose that $\sigma = \{t_i\}_{i=0}^{n}$.

1. The delay jitter of σ is 0 iff for all $0 \le i \le n$ we have $t_i = t_0 + i \cdot X_{\mathrm{a}}$, which is true iff the rate jitter of σ is 0.

2. If the delay jitter of σ is J, then for all $0 \le i, j < n$ we have $|t_{i+1} - t_i| \le J$ and $|t_{j+1} - t_j| \le J$, and by the triangle inequality we have that the rate jitter of σ is at most $2J$.

3. Let $\epsilon' = \min(\epsilon, 2X_{\mathrm{a}})$. Choose an even number $n > 4M/\epsilon'$, and let $\sigma = \{t_i\}_{i=0}^{n}$ be defined inductively as follows. For $0 < i \le n/2$, define $t_i = t_{i-1} + X_{\mathrm{a}} + \frac{\epsilon'}{2}$, and for $n/2 < i \le n$, define $t_i = t_{i-1} + X_{\mathrm{a}} - \frac{\epsilon'}{2}$. Clearly, the resulting σ is a times sequence with average inter-arrival rate X_{a} and rate jitter at most $\epsilon' \le \epsilon$. However, we have that $t_{n/2} - t_0 = \frac{n}{2}(X_{\mathrm{a}} + \frac{\epsilon'}{2})$, and hence the delay jitter is at least $\frac{n\epsilon'}{4} > M$ by choice of n.

∎

Our means for analyzing the performance of jitter control algorithms is *competitive analysis* [1]. In our context, we shall measure the (delay or rate) jitter of the sequence produced by an on-line algorithm against the best jitter attainable for that sequence. As expected, finding the release times which minimize jitter may require knowledge of the complete arrival sequence in advance, i.e., it can be computed only by an off-line algorithm. Our results are expressed in terms of the performance of our on-line algorithms using buffer space B_{on} as compared to the best jitter attainable by an off-line algorithm using space B_{off}, where usually $B_{\text{off}} < B_{\text{on}}$. We are interested in two parameters of the algorithms: the jitter (guaranteed by our on-line algorithms as a function of the best possible off-line guarantee) and the buffer size (used by the on-line algorithm, as a function of the buffer size used by an optimal off-line algorithm).

3 Delay-Jitter Control

In this section we analyze the best achievable delay-jitter. We first present an efficient off-line algorithm which attains the best possible delay jitter using a given buffer with space B. We then proceed to the main result of this section, which is an on-line delay-jitter control algorithm which attains the best jitter guarantee that can be attained by any (off-line) algorithm which uses half the buffer space. Finally, we present a lower bound which shows that any on-line algorithm whose jitter guarantees are a function of the jitter guarantees of an off-line algorithm, must have at least twice the space used by the off-line algorithm.

3.1 Off-line Delay-Jitter Control

We start with the off-line case. Suppose we are given the complete sequence $\{\mathit{arrival}(k)\}_{k=0}^{n}$ of packet arrival times. We wish to find a sequence of release times $\{\mathit{send}_{\text{off}}(k)\}_{k=0}^{n}$ which minimizes the delay jitter, using no more than B buffer space. The off-line algorithm is defined as follows.

Algorithm A: off-line delay-jitter control.

1. For each $0 \le k \le n$, define the interval

$$E_k = [\mathit{arrival}(k) - kX_{\text{a}} \ , \ \mathit{arrival}(k+B) - kX_{\text{a}}] \ ,$$

   where we define $\mathit{arrival}(k) = \infty$ for $k > n$.

2. Find a minimal interval M which intersects all intervals E_k.

3. For each packet k, let $P_k = \min(E_k \cap M)$, and define $send_{\text{off}}(k) = P_k + kX_{\text{a}}$.

Theorem 3.1 *The sequence* $\{send_{\text{off}}(k)\}_{k=0}^{n}$ *is a non-deceasing, B-feasible sequence with minimal delay jitter.*

Proof: It is straightforward to see from the definitions that $send_{\text{off}}(k) = P_k + kX_{\text{a}} \in [arrival(k), arrival(k+B)]$ and hence the resulting sequence is B-feasible. Proving FIFOness is done as follows. By definitions, it is sufficient to prove that $P_k \leq P_{k+1} + X_{\text{a}}$. To see this, first note that by definition,

$$\begin{aligned}
\min(E_k) &= arrival(k) - kX_{\text{a}} \\
&\leq arrival(k+1) - kX_{\text{a}} \\
&= \min(E_{k+1}) + X_{\text{a}}
\end{aligned} \tag{2}$$

We distinguish between two cases now. If $\min(M) \geq \min(E_k)$ then $P_k = \min(M)$ and by Eq. (2) we have that $P_{k+1} \geq P_k$, and we are done. The second case is that $\min(M) < \min(E_k)$, and then $P_k = \min(E_k)$. In this case Eq. (2) implies that

$$\begin{aligned}
P_{k+1} &= \min(E_{k+1} \cap M) \\
&\geq \min(E_{k+1}) \\
&\geq \min(E_k) - X_{\text{a}} \\
&= P_k - X_{\text{a}} \ ,
\end{aligned}$$

and the proof of correctness is complete. The optimality of the solution follows immediately from the minimality of M. ∎

3.2 On-line Delay-Jitter Control Algorithm

We now turn to our main result for delay-jitter control: an on-line algorithm using $2B$ buffer space, which guarantees delay-jitter bounded by the best jitter achievable by an off-line algorithm using B space. The algorithm is simple: first the buffer is loaded with B packets, and when the $(B+1)$-st packet arrives, the algorithm releases the first buffered packet. From this time on, the algorithm tries to release packet k after kX_{a} time. Formally, the algorithm is defined as follows.

Algorithm B: on-line delay-jitter control. Define $\mathit{send}^*_{\mathrm{on}}(k) = \mathit{arrival}(B) + kX_{\mathrm{a}}$ for all $0 \le k \le n$. The release sequence is defined by

$$\mathit{send}_{\mathrm{on}}(k) = \begin{cases} \mathit{send}^*_{\mathrm{on}}(k), & \text{if } \mathit{arrival}(k) \le \mathit{send}^*_{\mathrm{on}}(k) \le \mathit{arrival}(k+2B) \\ \mathit{arrival}(k), & \text{if } \mathit{send}^*_{\mathrm{on}}(k) < \mathit{arrival}(k) \\ \mathit{arrival}(k+2B), & \text{if } \mathit{send}^*_{\mathrm{on}}(k) > \mathit{arrival}(k+2B) \end{cases}$$

Clearly, Algorithm B is an on-line algorithm. We prove its jitter-control property.

Theorem 3.2 *If for a given arrival sequence, an off-line algorithm using space B can attain delay jitter J, then the release sequence generated by Algorithm B has delay-jitter at most J using no more than $2B$ buffer space.*

Proof: Obviously, the buffer space used by Algorithm B is at most $2B$. The bound on the delay-jitter follows from Lemma 3.4 and Lemma 3.6 proved below. ∎

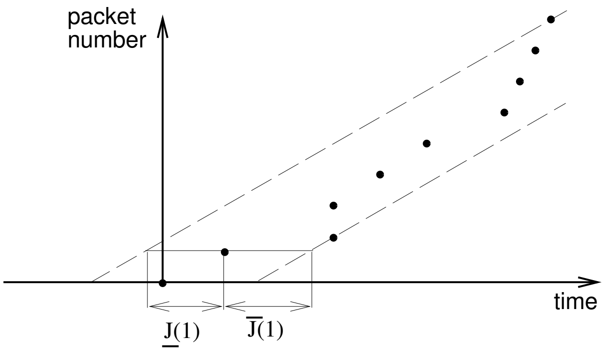

Figure 2: *An example for oriented jitter bounds. A point at coordinates (x, y) denotes that packet y is released at time x. The slope of the dashed lines is $1/X_{\mathrm{a}}$.*

The following definition is useful in the analysis (see Fig. 2).

Definition 3.1 *Let $\sigma = \{t_k\}_{k=0}^n$ be a time sequence. The* oriented jitter bounds for packet k *are*

$$\overline{J^\sigma}(k) \stackrel{\mathrm{def}}{=} \max_{0 \le i \le n} \{t_i - t_k - (i-k)X^\sigma_{\mathrm{a}}\}$$
$$\underline{J^\sigma}(k) \stackrel{\mathrm{def}}{=} \max_{0 \le i \le n} \{t_k - t_i - (k-i)X^\sigma_{\mathrm{a}}\}$$

Intuitvely, $\overline{J^\sigma}(k)$ says by how much packet k is late compared to the earliest packet, and $\underline{J^\sigma}(k)$ says by how much k is premature comapred to the latest packet. We have the following immediate properties for oriented jitter bounds.

Lemma 3.3 *Let $\sigma = \{t_k\}_{k=0}^n$ be a time sequence with average inter-arrival times X_a and delay jitter J. Then*

(1) For all k, $\overline{J}(k) \geq 0$ and $\underline{J}(k) \geq 0$.

(2) For all k, $\underline{J}(k) + \overline{J}(k) = J$.

(3) There exist k and k' such that $\overline{J}(k) = 0$ and $\underline{J}(k') = 0$.

Proof:

1. Follows by choosing $i = k$ in Definition 3.1.

2. Let i_0, j_0 be such that $J = |t_{i_0} - t_{j_0} - (i_0 - j_0)X_\mathrm{a}|$. Rearranging, we have that $J = |(t_{i_0} - i_0 X_\mathrm{a}) - (t_{j_0} - j_0 X_\mathrm{a})|$. Assume w.l.o.g. that $t_{i_0} - i_0 X_\mathrm{a} > t_{j_0} - j_0 X_\mathrm{a}$. From the definition it follows that $t_{i_0} - i_0 X_\mathrm{a} = \max_i \{t_i - iX_\mathrm{a}\}$ and $t_{j_0} - j_0 X_\mathrm{a} = \min_j \{t_j - jX_\mathrm{a}\}$. Therefore,

$$\begin{aligned} \overline{J}(k) &= \max_{0 \leq i \leq n} \{t_i - t_k - (i-k)X_\mathrm{a}\} \\ &= (t_{i_0} - t_k) - (i_0 - k)X_\mathrm{a}) \end{aligned} \tag{3}$$

and

$$\begin{aligned} \underline{J}(k) &= \max_{0 \leq i \leq n} \{t_k - t_i - (k-i)X_\mathrm{a}\} \\ &= (t_k - t_{j_0}) - (k - j_0)X_\mathrm{a} \end{aligned} \tag{4}$$

Summing Eqs. (3,4), the result follows.

3. Follows by choosing $k = i_0$ and $k = j_0$ in Eqs. (3,4), respectively.

▮

The following lemma shows that the deviation of the actual release time generated by Algorithm B from the ideal 0-jitter sequence of $\{send^*_\mathrm{on}(k)\}_k$ is bounded. Somewhat surprisingly, it is bounded by the oriented jitter bounds of two specific packets in *any* B-feasible sequence.

Lemma 3.4 *Let $\sigma = \{send_\mathrm{off}(k)\}_{k=0}^n$ be any B-feasible sequence for a given arrival sequence. Then for all $0 \leq k \leq n$, we have $-\overline{J^\sigma}(0) \leq send^*_\mathrm{on}(k) - send_\mathrm{on}(k) \leq \underline{J^\sigma}(B)$.*

Proof: We proceed by case analysis. If $send_\mathrm{on}(k) = send^*_\mathrm{on}(k)$ then we are done by Lemma 3.3 (1). If $send_\mathrm{on}(k) > send^*_\mathrm{on}(k)$, then by the specification of Algorithm B, we have that

$send_{\mathrm{on}}(k) = arrival(k)$. In this case the lemma is proved by the following inequality.

$$
\begin{aligned}
send_{\mathrm{on}}(k) &= arrival(k) \\
&\le send_{\mathrm{off}}(k) \\
&\le send_{\mathrm{off}}(0) + kX_{\mathrm{a}} + \overline{J^\sigma}(0) && \text{by definition of } \overline{J^\sigma}(0) \\
&\le send^*_{\mathrm{on}}(0) + kX_{\mathrm{a}} + \overline{J^\sigma}(0) && \text{since } send^*_{\mathrm{on}}(0) \ge send_{\mathrm{off}}(0) \\
&= send^*_{\mathrm{on}}(k) + \overline{J^\sigma}(0)
\end{aligned}
$$

The last case to consider is $send_{\mathrm{on}}(k) < send^*_{\mathrm{on}}(k)$. In this case, by the specification of Algorithm B, we have that $send_{\mathrm{on}}(k) = arrival(k + 2B)$. The lemma in this case is proved by the following inequality.

$$
\begin{aligned}
send_{\mathrm{on}}(k) &= arrival(k + 2B) \\
&\ge send_{\mathrm{off}}(k + B) && \text{by } B\text{-feasibility of off-line} \\
&\ge send_{\mathrm{off}}(B) + kX_{\mathrm{a}} - \underline{J^\sigma}(B) && \text{by definition of } \underline{J^\sigma}(B) \\
&\ge send^*_{\mathrm{on}}(0) + kX_{\mathrm{a}} - \underline{J^\sigma}(B) && \text{since } send^*_{\mathrm{on}}(0) \le send_{\mathrm{off}}(B) \\
&= send^*_{\mathrm{on}}(k) - \underline{J^\sigma}(B)
\end{aligned}
$$

∎

The reader may note that since Lemma 3.3 (1,2) implies that $\overline{J^\sigma}(0), \underline{J^\sigma}(0) \le J^\sigma$, Lemma 3.4 can be used to easily derive a bound of $2J^\sigma$ on the delay-jitter attained by Algorithm B. Proving the promised bound of J^σ requires a more refined analysis of the oriented jitter bounds. To facilitate it, we now introduce the following concept.

Definition 3.2 *Let $\sigma = \{t_k\}_{k=0}^n$ be a times sequence. Let $t \ge t_k$. The times sequence σ* perturbed at k to t *is*

$$
\sigma(k\!:\!t) \stackrel{\mathrm{def}}{=} \begin{cases} t_0, t_1, \ldots, t_{k-1}, \max(t, t_k), \max(t, t_{k+1}), \max(t, t_{k+2}), \ldots & \textit{if } t \ge t_k \\ \min(t, t_0), \min(t, t_1), \ldots, \min(t, t_k), t_{k+1}, t_{k+2}, \ldots & \textit{if } t \ge t_k \end{cases}
$$

Intuitively, $\sigma(k : t)$ is the sequence obtained by assigning release time t to packet k, and changing the times of other packets to preserve the FIFO order (see Fig. 3 for an example): if packet k is to be released earlier than t_k, then some packets before k may be moved as well; and if packet k is to be released later than t_k, then some packets after k may be moved.

The following properties for perturbed sequences are a direct consequence of the definition.

Lemma 3.5 *Let $\sigma = \{t_k\}_{k=0}^n$ be a times sequence, let k be any packet, and let t be any time point.*

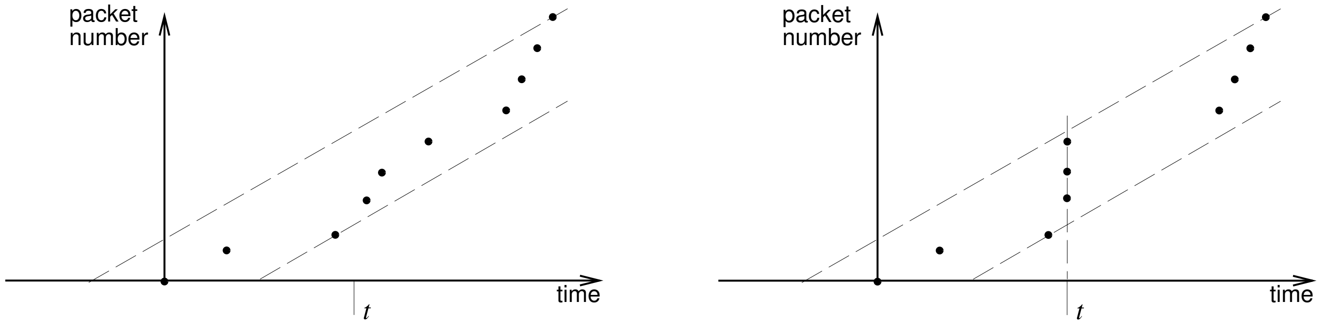

Figure 3: *An example of perturbation. Left: A sequence* σ. *Right:* $\sigma(5:t)$. *Note that in* $\sigma(5:t)$ *packets* $3, 4, 5$ *were moved with respect to* σ.

- *If* $t_k < t < t_k + \overline{J^\sigma}(k)$, *then*

  (A1) $J^{\sigma(k:t)} = J^\sigma$,

  (A2) $\overline{J^{\sigma(k:t)}}(k) < \overline{J^\sigma}(k)$, *and*

  (A3) $\underline{J^{\sigma(k:t)}}(i) \leq \underline{J^\sigma}(i)$ *for all* $i > k$.

- *If* $t_k - \overline{J^\sigma}(k) < t < t_k$, *then*

  (B1) $J^{\sigma(k:t)} = J^\sigma$,

  (B2) $\underline{J^{\sigma(k:t)}}(k) < \underline{J^\sigma}(k)$, *and*

  (B3) $\overline{J^{\sigma(k:t)}}(i) \leq \overline{J^\sigma}(i)$ *for all* $i < k$.

Proof: The simplest way to verify these claims is geometrical: Consider Figure 3, which corresponds to the case of $t_k - \overline{J^\sigma} < t < t_k$. Assertion (B1) says that if point k is not moved left to the left diagonal line, then all points remain between the two diagonal lines, and that there are points which lie on the diagonal lines. Assertion (B2) states that the horizontal distance between point k and left diagonal line strictly decreases, and Assertion (B3) states that for points below point k, the horizontal distance to the left diagonal line does not increase. The case of $t_k - \overline{J^\sigma}(k) < t < t_k$ is analogous. ∎

To prove Theorem 3.2, we prove an interesting property of oriented jitter bounds in optimal sequences. Intuitively, the lemma below says the following. Fix an arrival sequence, and consider all optimal release sequences using B buffer space. Fix any two packets at most B apart. Then it cannot be the case that in all optimal release sequences both the first packet is too early and the second packet is too late. Formally, we have the following.

Lemma 3.6 *Let J be the minimal delay jitter for a given arrival sequence using space B, and let $0 \le i \le j \le n$ be packets such that $j \le i + B$. Let σ be an optimal release sequence attaining jitter J for the given arrival sequence, in which $\overline{J^\sigma}(i) + \underline{J^\sigma}(j)$ is minimal among all optimal sequences. Then $\overline{J^\sigma}(i) + \underline{J^\sigma}(j) \le J$.*

Note that Lemma 3.6 with $i = 0$ and $j = B$, combined with Lemma 3.4, completes the proof of Theorem 3.2. We shall use the general statement in Section 4.

Proof: First, note that if either $\overline{J^\sigma}(i) = 0$ or $\underline{J^\sigma}(j) = 0$, then we are done since by Lemma 3.3 (1,2) we have that $\overline{J^\sigma}(i), \underline{J^\sigma}(j) \le J$. So assume from now on that for all sequences using space B and attaining delay jitter J we have that $\overline{J^\sigma}(i) > 0$ and $\underline{J^\sigma}(j) > 0$. We claim that in this case, $t_i = t_j$, i.e., packet i and packet j are released together (and hence all packets $i, \ldots, j$ are released together). We prove this claim by contradiction: suppose that $t_i < t_j$. Then it must be the case that either (i) $t_i < \mathit{arrival}(j)$ or (ii) $t_j > \mathit{arrival}(j)$, or both (i) and (ii) hold. If case (i) holds, let $0 < t < \min\left\{t_i + \overline{J^\sigma}(i), \mathit{arrival}(j)\right\}$ and consider the perturbed sequence $\sigma(i\!:\!t)$ in which packet i is released at time t. By choice of t, we have that $t_i < t < t_i + \overline{J^\sigma}(i)$. The perturbed sequence $\sigma(i\!:\!t)$ has the following properties.

(1) $\sigma(i\!:\!t)$ is B-feasible, since it may differ from σ at most by packets $i+1, \ldots, j-1$. These packets are held a little longer in $\sigma(i\!:\!t)$, but they are released at time $t < \mathit{arrival}(j) \le \mathit{arrival}(i + B)$.

(2) $J^{\sigma(i:t)} = J$ by Lemma 3.5 (A1).

(3) $\overline{J^{\sigma(i:t)}}(i) < \overline{J^\sigma}(i)$ by Lemma 3.5 (A2).

(4) $\underline{J^{\sigma(i:t)}}(j) \le \underline{J^\sigma}(j)$ by Lemma 3.5 (A3).

The claim now follows for case (i), since Properties (1,2) imply that $\sigma(i\!:\!t)$ is a sequence using B buffer space which attains jitter J, but Properties (3,4) contradict the assumed minimality of $\overline{J^\sigma}(i) + \underline{J^\sigma}(j)$. A similar argument shows that if case (ii) holds, then for $0 < t < \max\left\{t_j - \underline{J^\sigma}(j), \mathit{arrival}(j)\right\}$, the perturbed sequence $\sigma(j\!:\!t)$ contradicts the minimality of $\overline{J^\sigma}(i) + \underline{J^\sigma}(j)$.

Having proved that $t_i = t_j$, we can now bound $\overline{J^\sigma}(i) + \underline{J^\sigma}(j)$. First, note that since by definition there exists a packet k_1 such that $t_{k_1} = t_j + (k_1 - j)X_{\mathrm{a}} - \underline{J^\sigma}(j)$, and since by definition $t_{k_1} \ge t_i + (k_1 - i)X_{\mathrm{a}} - \underline{J^\sigma}(i)$, we get from the fact that $t_i = t_j$ that

$$\underline{J^\sigma}(i) \ge (j - i) \cdot X_{\mathrm{a}} + \underline{J^\sigma}(j) \ . \tag{5}$$

Similarly, we have that

$$\overline{J^\sigma}(j) \ge (j - i) \cdot X_{\mathrm{a}} + \overline{J^\sigma}(i) \ . \tag{6}$$

Adding Equations (5,6), we get that

$$\underline{J^\sigma}(i) + \overline{J^\sigma}(j) \;\geq\; \overline{J^\sigma}(i) + \underline{J^\sigma}(j) + 2(j-i)X_{\mathrm{a}} \;.$$

Since $\overline{J^\sigma}(i) + \underline{J^\sigma}(i) = \overline{J^\sigma}(j) + \underline{J^\sigma}(j) = J$, we conclude that

$$\overline{J^\sigma}(i) + \underline{J^\sigma}(j) \;\leq\; J - (j-i)X_{\mathrm{a}} \;\leq\; J \;,$$

as required. ∎

3.3 A Lower Bound for On-line Delay-Jitter Control Algorithms

We close this section with a lower bound for on-line delay-jitter control algorithms. The following theorem says that any on-line algorithm using less than $2B$ buffer space pays heavily in terms of delay jitter when compared to an off-line algorithm using space B.

Theorem 3.7 *Let $1 \leq \ell < B$. There exist arrival sequences for which an off-line algorithm using space B gets jitter 0, and any on-line algorithm using $2B - \ell$ buffer space gets delay-jitter at least ℓX_{a}. Moreover, there exist arrival sequences for which an off-line algorithm using space B gets 0-jitter, and no on-line algorithm using less than B buffer space can guarantee any finite delay jitter.*

Proof: Consider the following scenario. At time 0, packets $0, \ldots, B-1$ arrive, and at time $B \cdot X_{\mathrm{a}}$, packets $B, \ldots, 2B$ arrive. First, note that there is an off-line algorithm attaining 0 jitter by releasing each packet k at time $k \cdot X_{\mathrm{a}}$. Consider now any on-line algorithm Z. We first claim that Z cannot release packet 0 before packet B arrives: otherwise, packet B may arrive arbitrarily far in the future, making the delay jitter of the on-line algorithm arbitrarily large. Hence, at time $B \cdot X_{\mathrm{a}}$, when $B+1$ new packets arrive, algorithm Z still stores the first B packets, and since it has buffer space $2B - \ell$ by assumption, it is forced to release at least $\ell + 1$ packets immediately. Since the delays of packets 0 and ℓ are equal, it follows from the definition of delay-jitter that the delay-jitter of the release sequence is at least ℓX_{a}.

For the case of an on-line algorithm with less than B space, consider the scenario where a batch of B packet arrive together at time 0, and then a batch of B more packets arrive at time T for some very large T. Since the on-line algorithm has to release packet 0 at time 0, we have that its delay jitter is at least $T/(B-1)$, which can be arbitrarily large. ∎

4 Distributed Delay-Jitter Control

In Section 3 we have considered a single delay-jitter regulator. In this section we prove an interesting property of composing many delay-jitter regulators employing our Algorithm B. Specifically, we consider a path of m links connecting nodes $v_0, v_1, \ldots, v_m$, where v_0 is the source and v_m is the destination. We make the simplifying assumption that the propagation delay in each link is deterministic. We denote the event of the arrival of packet k at node j by $\mathit{arrival}(k,j)$, and the release of packet k from node v_j by $\mathit{send}(k,j)$. The input stream, generated by the source, is $\{\mathit{send}(k,0)\}_k$ (or $\{\mathit{arrival}(k,1)\}_k$), and the output stream is $\{\mathit{send}(k,m)\}_k$. Each node has $2B/m$ buffer space, and for simplicity we assume that m divides B. The distributed algorithm is the following.

Algorithm BD: distributed on-line delay-jitter control. For each $1 \le j \le m$, node v_j employs Algorithm B with buffer space $2B/m$. Specifically, node j sets $\mathit{send}^*_{\mathrm{on}}(k,j) = \mathit{arrival}(B/m,j) + kX_{\mathrm{a}}$, and it releases packet k as close as possible to $\mathit{send}^*_{\mathrm{on}}(k,j)$ subject to $2B/m$-feasibility (see Algorithm B).

We prove that the jitter control capability of Algorithm BD is the same as the jitter control capability of a *centralized* jitter control algorithm with B total buffer space, under a certain condition for the beginning of the sequence (to be explained shortly). Put differently, one does not lose jitter control capability by dividing the buffer space along the path. The precise result is given in the theorem below.

Theorem 4.1 *Suppose that for a given arrival sequence $\sigma = \{\mathrm{arrival}(k)\}_{k=0}^n$, there exists a centralized off-line algorithm attaining jitter J using space B, with packet* 0 *released before time* $\mathrm{arrival}(B/m)$*. Then if σ is the release sequence of node v_0, the release sequence $\{\mathrm{send}_{\mathrm{on}}(k,m)\}_k$ generated by Algorithm BD at node v_m has delay jitter at most J.*

Intuitively, the additional condition is that there is a way to release the first packet relatively early by a centralized optimal algorithm. This condition suffices to compensate for the distributed nature of Algorithm BD. The condition is also necessary for the algorithm to work: if B packets are input into the system at the start of the algorithm, then an off-line algorithm can still wait arbitrarily long before starting to release packets, while Algorithm BD is bound to start releasing packets even if only $\frac{2B}{m}+1$ packets arrive.

The proof is essentially adapting the proofs of Algorithm B in Section 3 to the distributed setting. We highlight the distinguishing points.

Let the propagation delay over link (v_{j-1}, v_j) be d_j, and denote $D = \sum_{j=2}^m d_j$, the total delay of links on the path.

The first lemma below bounds the desired release times of all packets at one node in terms of the desired release times in upstream nodes.

Lemma 4.2 *For all nodes $1 \le j \le i \le m$ and all packets k,*

$$\mathit{send}^*_{\mathrm{on}}(k,j) + \sum_{\ell=j+1}^{i} d_\ell \;\le\; \mathit{send}^*_{\mathrm{on}}(k,i) \;\le\; \mathit{send}^*_{\mathrm{on}}(k + \frac{B}{m}(i-j), j) + \sum_{\ell=j+1}^{i} d_\ell \;.$$

Proof: Consider the lower bound first. By the algorithm, we have that for all ℓ, $\mathit{send}^*_{\mathrm{on}}(0,\ell) = \mathit{send}_{\mathrm{on}}(0,\ell)$. Since for all $\ell > 1$, $\mathit{send}^*_{\mathrm{on}}(0,\ell) \ge \mathit{send}_{\mathrm{on}}(0,\ell-1) + d_\ell$, we obtain by induction on $i-j$ that $\mathit{send}^*_{\mathrm{on}}(k,i) \ge \mathit{send}^*_{\mathrm{on}}(k,j) + \sum_{\ell=j+1}^{i} d_\ell$, proving the lower bound.

We now prove the upper bound. First, we claim that for all $1 \le i \le m$,

$$\mathit{send}_{\mathrm{on}}(k,\ell) \le \mathit{send}^*_{\mathrm{on}}(k,\ell) \qquad \text{for } 0 \le k \le \frac{B}{m} \;. \tag{7}$$

Eq. (7) follows from the fact that by the specification of Algorithm B, a node starts releasing packets only if all first $\frac{B}{m}+1$ packets are in its buffer, and therefore none of the first $\frac{B}{m}+1$ packets is released too late in any node. We now prove the upper bound by induction on $i-j$. The base case, $i=j$, is trivial. For the inductive step, fix j and consider $i+1$. We have

$$\begin{aligned}
\mathit{send}^*_{\mathrm{on}}(k,i+1) &= \mathit{send}_{\mathrm{on}}(\tfrac{B}{m},i) + d_{i+1} + kX_{\mathrm{a}} && \text{by algorithm}\\
&\le \mathit{send}^*_{\mathrm{on}}(0,i) + d_{i+1} + (k+\tfrac{B}{m})X_{\mathrm{a}} && \text{by (7)}\\
&\le \mathit{send}^*_{\mathrm{on}}(0,j) + X_{\mathrm{a}}\frac{B}{m}(i-j) + \sum_{\ell=j+1}^{i} d_\ell + d_{i+1} + (k+\frac{B}{m})X_{\mathrm{a}} && \text{by induction}\\
&= \mathit{send}^*_{\mathrm{on}}(k,j) + X_{\mathrm{a}}\frac{B}{m}(i+1-j) + \sum_{\ell=j+1}^{i+1} d_\ell && \text{rearranging}\\
&= \mathit{send}^*_{\mathrm{on}}(k + \frac{B}{m}(i+1-j), j) + \sum_{\ell=j+1}^{i+1} d_\ell
\end{aligned}$$

∎

For the case of underflow, we argue that if a packet is "late" in the output node v_n, then it was late in all nodes on its way.

Lemma 4.3 *If $\mathit{send}_{\mathrm{on}}(k,m) > \mathit{send}^*_{\mathrm{on}}(k,m)$, then $\mathit{send}_{\mathrm{on}}(k,m) = \mathit{arrival}(k,1) + D$.*

Proof: First, we show that for any node v_j, if $\mathit{send}_{\mathrm{on}}(k,j) > \mathit{send}^*_{\mathrm{on}}(k,j)$, then $\mathit{send}_{\mathrm{on}}(k,j-1) > \mathit{send}^*_{\mathrm{on}}(k,j-1)$. This is true since by the specification of Algorithm B, at time $\mathit{send}^*_{\mathrm{on}}(k,j)$ the buffer at node j is empty, and hence node v_{j-1} has not sent packet k by time

$send^*_{\text{on}}(k,j)-d_j$. Since $send^*_{\text{on}}(k,j-1) \le send^*_{\text{on}}(k,j)-d_j$, this implies that $send_{\text{on}}(k,j-1) > send^*_{\text{on}}(k,j-1)$. Therefore, for all nodes v_j, we have that $send_{\text{on}}(k,j) = arrival(k,j)$, and by summation we obtain that $send_{\text{on}}(k,m) = send_{\text{on}}(k,0) + D$. ∎

For the case of overflow, we show the analogous property: if there is an overflow in the output node, then it is the result of a "chain reaction" of overflows in all nodes.

Lemma 4.4 *If $send_{\text{on}}(k,m) < send^*_{\text{on}}(k,m)$, then $send_{\text{on}}(k,m) = arrival(k+2B,1) + D$.*

Proof: We prove that if $send_{\text{on}}(k,i) < send^*_{\text{on}}(k,i)$ then $send_{\text{on}}(k+2\frac{B}{m}, i-1) < send^*_{\text{on}}(k+2\frac{B}{m}, i-1)$.

$$\begin{array}{rcll} send_{\text{on}}(k+\frac{2B}{m}, i-1) & \le & send_{\text{on}}(k,i) - d_i & \text{by the bound on the buffer size} \\ & < & send^*_{\text{on}}(k,i) - d_i & \text{by our assumption} \\ & \le & send^*_{\text{on}}(k+\frac{B}{m}, i-1) & \text{by Lemma 4.2} \\ & \le & send^*_{\text{on}}(k+\frac{2B}{m}, i-1) & \end{array}$$

In other words, if packet k is overflowing at node m, then packet $k+\frac{2B(m-i)}{m}$ is overflowing in node i, for each $1 \le i \le m$. Hence for each i, we have $send_{\text{on}}(k+\frac{2B(m-i)}{m}, i) = send_{\text{on}}(k+\frac{2B(m-(i-1))}{m}, i-1) + d_i$. The lemma follows. ∎

The lemmas above are used in the proof of the following variant of Lemma 3.4.

Lemma 4.5 *Let $\sigma = \{send_{\text{off}}(k)\}_{k=0}^{m}$ be any B-feasible sequence for a given arrival sequence such that $send_{\text{off}}(0) \le arrival(B/m, 1)$. Then for all $0 \le k \le m$, we have $-\overline{J^\sigma}(0) \le send^*_{\text{on}}(k,m) - send_{\text{on}}(k,m) \le \underline{J^\sigma}(B/m)$.*

Proof: If $send^*_{\text{on}}(k,m) = send_{\text{on}}(k,m)$ we are done by Lemma 3.3 (1). If $send_{\text{on}}(k,m) > send^*_{\text{on}}(k,m)$, then

$$\begin{array}{rcll} send_{\text{on}}(k,m) & = & send_{\text{on}}(k,0) + D & \text{by Lemma 4.3} \\ & \le & send_{\text{off}}(k) + D & \text{since } send_{\text{on}}(k,0) = arrival(k,1) \\ & \le & send_{\text{off}}(0) + kX_{\text{a}} + \overline{J^\sigma}(0) + D & \\ & \le & send^*_{\text{on}}(0,1) + kX_{\text{a}} + \overline{J^\sigma}(0) + D & \text{since } send_{\text{off}}(0) \le arrival(\frac{B}{m},1) \\ & \le & send^*_{\text{on}}(0,m) + kX_{\text{a}} + \overline{J^\sigma}(0) & \text{by Lemma 4.2} \\ & = & send^*_{\text{on}}(k,m) + \overline{J^\sigma}(0) & \end{array}$$

If $send_{\text{on}}(k,m) < send^*_{\text{on}}(k,m)$, then

$$\begin{aligned}
send_{\text{on}}(k,m) &= arrival(k+2B,1)+D && \text{by Lemma 4.4}\\
&\geq send_{\text{off}}(k+B)+D && \text{since off-line has } B \text{ space}\\
&\geq send_{\text{off}}(\tfrac{B}{m})+(k+(B-\tfrac{B}{m}))X_{\text{a}}-\underline{J}(\tfrac{B}{m})+D &&\\
&\geq arrival(\tfrac{B}{m},1)+(k+(B-\tfrac{B}{m}))X_{\text{a}}-\underline{J}(\tfrac{B}{m})+D && \text{By algorithm BD}\\
&\geq send^*_{\text{on}}(0,1)+(k+(B-\tfrac{B}{m}))X_{\text{a}}-\underline{J}(\tfrac{B}{m})+D &&\\
&= send^*_{\text{on}}(k+(B-\tfrac{B}{m}),1)-\underline{J}(\tfrac{B}{m})+D &&\\
&\geq send^*_{\text{on}}(k,m)-\underline{J}(\tfrac{B}{m}) && \text{by Lemma 4.2}
\end{aligned}$$

▮

Theorem 4.1 follows from Lemma 4.5, when combined with Lemma 3.6 (which is independent of the on-line algorithm), with $i = 0$ and $j = B/m$.

5 Rate-Jitter Control

In this section we consider the problem of minimizing the rate-jitter, i.e., how to keep the rate at which packets are released within the tightest possible bounds. We shall use the equivalent concept of minimizing the difference between inter-departure times. We present an on-line algorithm for rate-jitter control using space $2B + h$ and compare it to an off-line algorithm using space B and guaranteeing jitter J. Our algorithm guarantees rate jitter at most $J + c \cdot B/h$, for some constant c. We also show how to obtain rate jitter which is a multiplicative factor from optimal, with a simple modification of the algorithm. The algorithm can work without knowledge of the exact average inter-arrival time: in this case, jitter guarantees will come into effect after an initial period in which packets may be released too slowly. We also show that without doubling the space, no guarantees in terms of the optimal rate-jitter can be made. As an aside, we remark that off-line rate-jitter control can be solved optimally using linear-programming technique.

5.1 On-line Rate-Jitter Control Algorithm

We now turn to describe the main result for this section: an on-line algorithm for rate-jitter control. The algorithm is specified with the following parameters:

- B, the buffer size of an off-line algorithm, i.e. $B_{\text{off}} = B$.
- $h \geq 1$, space parameter for the on-line algorithm, such that $B_{\text{on}} = 2B + h$.

- $I_{\min}, I_{\max}$: bounds on the minimum and maximum inter-departure time of an off-line algorithm.
- X_{a}, the average inter-departure time in the input (and also the output) sequence.

The parameters $I_{\min}$ and $I_{\max}$ can be thought of as requirements: these should be the worst rate jitter bounds the application is willing to tolerate. The goal of a rate-jitter control algorithm is to minimize the rate jitter, subject to the assumption that space B is sufficient (for an off-line algorithm) to bound the inter-departure times in the range $[I_{\min}, I_{\max}]$. A trivial choice for $I_{\min}$ and $I_{\max}$ is $X_{\min}$ and $X_{\max}$, which are the minimal and maximal inter arrival times in the input sequence. However, using tighter $I_{\min}$ and $I_{\max}$, one may get a much stronger guarantee. The jitter guarantees will be expressed in terms of $B, h, I_{\max}, I_{\min}, X_{\mathrm{a}}$ and J, the best rate jitter for the given arrival sequence attainable by an off-line algorithm using space B.

Note that for an on-line algorithm, even achieving rate jitter $I_{\max} - I_{\min}$ may be non-trivial. These are bounds on the performance of an *off-line* algorithm, whose precise specification may depend on events arbitrarily far in the future.

The basic idea in our algorithm is that the next release time is a monotonically decreasing function of the current number of packets in the buffer. In other words, the more packets there are in the buffer, the lower the inter-departure time between the packets (and thus the higher the release rate).

Algorithm C: on-line rate-jitter control. The algorithm uses $B_{\mathrm{on}} \stackrel{\mathrm{def}}{=} 2B + h$ buffer space. With each possible number $0 \le j \le 2B + h$ of packets in the buffer, we associate an inter-departure time denoted $\mathrm{IDT}(j)$, defined as follows. Let $\delta \stackrel{\mathrm{def}}{=} (I_{\max} - I_{\min})/h$.

$$\mathrm{IDT}(j) = \begin{cases} I_{\max}, & \text{if } 0 \le j \le B \\ I_{\max} - (j - B)\delta, & \text{if } B \le j \le B + h \\ I_{\min}, & \text{if } B + h \le j \le 2B \end{cases}$$

Note that $\mathrm{IDT}(j)$ is a monotonically decreasing function in j. The algorithm starts with a *buffer loading* stage, in which packets are only accumulated (and not released) until the first time that the number j of packets in the buffer satisfies $\mathrm{IDT}(j) \le X_{\mathrm{a}}$. Let $S = \min\{j \mid \mathrm{IDT}(j) \le X_{\mathrm{a}}\}$, and let T^* denote the first time in which the number of packets in the buffer reaches S. At time T^*, the loading stage is over: the first packet is released and the following rule governs the remainder of the execution of the algorithm. A variable `last_departure` is maintained, whose value is the time at which the last packet was sent. If at time t, we have $t \le$ `last_departure` $+ \mathrm{IDT}(j)$, where j is the number of packets currently in the buffer, then we deliver a packet and update `last_departure`.

The rate-jitter bound of Algorithm C is given in the following theorem.

Theorem 5.1 *Let J be the best rate-jitter attainable (for an off-line algorithm) using buffer space B for a given arrival sequence. Then the maximal rate-jitter in the release sequence generated by Algorithm C is at most $J + (I_{\max} - I_{\min})\frac{2B+4}{h}$, and never more than $I_{\max} - I_{\min}$.*

The idea in the proof of Theorem 5.1 is that the number of packets in the buffer is never more than $B+2$ buffer slots away from the slots which correspond to rates generated by an optimal off-line algorithm. We now formally analyze Algorithm C. Fix an optimal execution of the off-line algorithm. Let us denote the maximum and minimum inter-departure times of the off-line execution by $Y_{\max}$ and $Y_{\min}$, respectively. (Hence the jitter attained by the off-line algorithm is $Y_{\max} - Y_{\min}$.) With these quantities, we also define the following terms.

$$
\begin{aligned}
U &= \min\{j \mid \mathrm{IDT}(j) \le Y_{\min}\} \\
I_U &= \mathrm{IDT}(U) \\
L &= \max\{j \mid \mathrm{IDT}(j) \ge Y_{\max}\} \\
I_L &= \mathrm{IDT}(L)
\end{aligned}
$$

Note that $L \le S \le U$. We shall also use the following shorthand notation. Let $B_{\mathrm{on}}(t)$ and $B_{\mathrm{off}}(t)$ denote the number of packets stored in time t in the buffers of the Algorithm C and of the off-line algorithm, respectively, and let $\mathrm{diff}(t) = B_{\mathrm{on}}(t) - B_{\mathrm{off}}(t)$, i.e., how many packets does the Algorithm C has more than the off-line algorithm at time t. We use extensively the following trivial property of the difference.

Lemma 5.2 *For all t, $-B \le \mathrm{diff}(t) \le B_{\mathrm{on}}(t)$.*

Proof: Immediate from the fact that $0 \le B_{\mathrm{off}}(t) \le B$. ▮

Let $S_{\mathrm{on}}(t_1, t_2)$ denote the number of packets sent by Algorithm C in the time interval $[t_1, t_2]$, and define $S_{\mathrm{off}}(t_1, t_2)$ analogously for the off-line algorithm. The following lemma states that the difference is modified according only to the difference in the packets released.

Lemma 5.3 *For any two time points $t_1 \le t_2$, $\mathrm{diff}(t_2) = \mathrm{diff}(t_1) + S_{\mathrm{on}}(t_2, t_1) - S_{\mathrm{off}}(t_2, t_1)$.*

Proof: Consider the events in the time interval $[t_1, t_2]$. A packet arrival increases the number of stored packets for both the off-line and Algorithm C, and hence does not change their difference. It follows that $\mathrm{diff}(t_2) - \mathrm{diff}(t_1)$ is exactly the difference in the number of packets sent by the two algorithms in the given interval. ▮

The significance of Lemma 5.3 is in that it allows us to ignore packet arrivals when analyzing the space requirement of an algorithm: all we need is to consider the difference from the space requirement of the off-line algorithm. The following lemma, which bounds the minimal inter-departure time of Algorithm C, is an example for that.

Lemma 5.4 *For all times t,* $\mathrm{diff}(t) \le U + 1$.

Proof: Let t be any point in time. If $B_{\mathrm{on}}(t) \le U + 1$, the lemma follows immediately. So assume that $B_{\mathrm{on}}(t) > U+1$, and let $t_0 < t$ be a point such that $B_{\mathrm{on}}(t_0) \le U$ and $B_{\mathrm{on}}(t') \ge U$ for all $t' \in [t_0, t]$. Such a point exists since $B_{\mathrm{on}}(T^*) = S < U$. Consider the time interval $[t_0, t]$: in this interval, at most $\left\lfloor \frac{t-t_0}{Y_{\min}} \right\rfloor + 1$ packets were released by the off-line algorithm, while Algorithm C has released at least $\left\lfloor \frac{t-t_0}{I_U} \right\rfloor \ge \left\lfloor \frac{t-t_0}{Y_{\min}} \right\rfloor$, and hence $S_{\mathrm{on}}(t_0, t) - S_{\mathrm{off}}(t_0, t) \le 1$. Since by Lemma 5.2 we have that $\mathrm{diff}(t_0) \le U$, the result follows from Lemma 5.3. ∎

Similarly, we bound the difference from below.

Lemma 5.5 *For all times $t > T^*$,* $\mathrm{diff}(t) \ge (L - 1) - B$.

Proof: Let $t > T^*$ be a point in time. The case of $B_{\mathrm{on}}(t) \ge L + 1$ is trivial; we consider $B_{\mathrm{on}}(t) < L + 1$. Let $t_0 < t$ be a point such that $B_{\mathrm{on}}(t_0) \ge L$ and $B_{\mathrm{on}}(t') \le L$ for all $t' \in [t_0, t]$. The point t_0 must exist since $B_{\mathrm{on}}(T^*) = S \ge L$. For the time interval $[t_0, t]$, we have that $S_{\mathrm{off}}(t_0, t) \ge \left\lfloor \frac{t-t_0}{Y_{\max}} \right\rfloor$, and $S_{\mathrm{on}}(t_0, t) \le \left\lfloor \frac{t-t_0}{I_L} \right\rfloor + 1 \le \left\lfloor \frac{t-t_0}{Y_{\max}} \right\rfloor + 1$, and hence $S_{\mathrm{on}}(t_0, t) - S_{\mathrm{off}}(t_0, t) \ge -1$. Since $\mathrm{diff}(t_0) \ge L - B$, the result follows from Lemma 5.3. ∎

We now prove Theorem 5.1.

Proof of Theorem 5.1: By Lemma 5.4, at all times t, $B_{\mathrm{on}}(t) \le B + U + 1$ and hence the minimal inter-departure time of Algorithm C is smaller than $Y_{\min}$ by less than $(B+2)\frac{I_{\max}-I_{\min}}{h}$. By Lemma 5.5, for all times $t > T^*$, the maximal inter-departure time of Algorithm C is larger than $Y_{\max}$ by less than $(B + 2)\frac{I_{\max}-I_{\min}}{h}$. Since $J = Y_{\max} - Y_{\min}$ and since no packets is released before time T^*, the theorem follows. ∎

It is worthwhile noting that doubling the space is mandatory for on-line rate-jitter control (as well as for delay-jitter control), as the following theorem implies.

Theorem 5.6 *Let $1 \le \ell < B$. There exist arrival sequences for which an off-line algorithm using space B gets 0-jitter, and any on-line algorithm using $2B-\ell$ buffer space gets rate-jitter at least $X_{\mathrm{a}}\frac{B}{2B-\ell}$.*

The proof of Theorem 5.6 is similar to the proof of Theorem 3.7, and we therefore omit it.

5.2 Adapting to Unknown X_{a}

We can avoid the need of knowing X_{a} in advance, if we are willing to tolerate slow rate in an initial segment of the online algorithm. This is done by changing the specification of the loading stage of Algorithm C to terminate when the buffer contains B packets (which corresponds to inter-arrival time of $I_{\max}$, as opposed to inter-arrival time of X_{a} in the original specification). Thereafter, the algorithm starts releasing packets according to the specification of IDT. Call the resulting algorithm C^-. Below, we bound the time which elapses in an execution of C^- until the buffer size will reach the value of L. Clearly, from that point onward, all guarantees made in Theorem 5.1 hold true for Algorithm C^- as well.

Lemma 5.7 *Consider an execution of Algorithm C^-. Let T^* be the time where the initial loading ends, and let T^+ be the first time such that $B_{\text{on}}(T^+) = L$. Then*

$$T^+ - T^* \leq \frac{Y_{\max}^2 \ln(L-B)}{\delta} + Y_{\max} L + \frac{Y_{\max}^2 B}{\delta}$$

Proof: For $0 \leq i \leq L-B$, define t_i to be the first time after T^* where $B_{\text{on}}(t_i) \geq L-i$. Consider a time interval $[t_i, t_{i-1}]$, and denote its length by τ_i. Denote the number of packets arriving in the interval by A_i. Consider the off-line algorithm: For all $1 \leq i \leq L-B$, we have that

$$\begin{aligned} A_i &= S_{\text{off}}(t_i, t_{i-1}) + B_{\text{off}}(t_{i-1}) - B_{\text{off}}(t_i) \\ &\geq \frac{\tau_i}{Y_{\max}} + B_{\text{off}}(t_{i-1}) - B_{\text{off}}(t_i) \end{aligned} \tag{8}$$

Consider now the execution of Algorithm C^-: in the time interval $[t_i, t_{i-1}]$, the inter-departure time is at least $Y_{\max} + i\delta$, and therefore $S_{\text{on}}(t_i, t_{i-1}) \leq \frac{\tau_i}{Y_{\max}+i\delta}$. Using Eq. (8) and since $B_{\text{on}}(t_{i-1}) - B_{\text{on}}(t_i) = 1$ by definition, we have

$$\begin{aligned} 1 &= B_{\text{on}}(t_{i-1}) - B_{\text{on}}(t_i) \\ &= A_i - S_{\text{on}}(t_i, t_{i-1}) \\ &\geq \frac{\tau_i}{Y_{\max}} + B_{\text{off}}(t_{i-1}) - B_{\text{off}}(t_i) - \frac{\tau_i}{Y_{\max} + i\delta} \end{aligned}$$

i.e.,

$$\begin{aligned} \tau_i &\leq \frac{(1 + B_{\text{off}}(t_i) - B_{\text{off}}(t_{i-1}))\, Y_{\max}(Y_{\max} + i\delta)}{i\delta} \\ &= \frac{Y_{\max}^2}{\delta} \cdot \frac{1}{i} + Y_{\max} - \frac{Y_{\max}^2}{\delta} \cdot \frac{B_{\text{off}}(t_{i-1}) - B_{\text{off}}(t_i)}{i} \\ &\quad - Y_{\max}\left(B_{\text{off}}(t_{i-1}) - B_{\text{off}}(t_i)\right) \end{aligned}$$

Summing over $i = 1, \ldots, L-B$ and noting that $\sum_{i=1}^{L-B} \tau_i = T^+ - T^*$ and that $-B \leq B_{\text{off}}(t_{L-B}) - B_{\text{off}}(t_0) \leq B$, we obtain

$$T^+ - T^* \leq \frac{Y_{\max}^2 \ln(L-B)}{\delta} + Y_{\max} L + \frac{Y_{\max}^2 B}{\delta}\ .$$

∎

5.3 Multiplicative Rate Jitter

For some applications, it may be useful to define jitter as the *ratio* between the maximal and minimal inter-arrival times. We call this measure the *multiplicative rate jitter*, or m-rate jitter for short. It is easy to adapt Algorithm C to the case where we are interested in the m-rate jitter. All that is needed is to define

$$\mathrm{IDT}_m(j) = \begin{cases} I_{\max}, & \text{if } 0 \leq j \leq B \\ I_{\max} \cdot \delta^{j-B}, & \text{if } B \leq j \leq B+h \\ I_{\min}, & \text{if } B+h \leq j \leq 2B \end{cases}$$

for $\delta = (I_{\min}/I_{\max})^{1/h}$. In this case we obtain the following result, using the same proof technique as for Theorem 5.1.

Theorem 5.8 *Let J be the best m-rate-jitter attainable (for an off-line algorithm) using buffer space B for a given arrival sequence. Then the maximal m-rate-jitter in the release sequence generated by Algorithm C using function* IDT_m *is at most* $J \cdot (I_{\max}/I_{\min})^{\frac{2B+4}{h}}$.

6 Conclusion

In this paper we have studied jitter control algorithms, measured in terms of guarantees relative to the best possible by an off-line algorithm. Our results for delay jitter show that the simple algorithm of filling half the buffer has a very strong relative property. For rate jitter, we proposed a simple algorithm where the release rate is proportional to the fill level of the buffer, and showed that its relative guarantees are quite strong as well. We have studied a very simple distributed model for jitter control. We leave for further work analyzing more realistic models of systems, including multiple streams and more interesting network topology.

References

[1] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.

[2] D. Ferrari. Client requirements for real-time communication services. *IEEE Communications Magazine*, November 1990. See also Intentet RFC 1193, November 1990.

[3] N. R. Figuera and J. Pasquale. Leave-in-Time: A new service discipline for real-time communications in packet-switching networks. In *Proceedings of ACM SIGCOMM*, pages 207–218, 1995.

[4] S. J. Golestani. A Stop-and-Go queueing framework for congestion management. In *Proceedings of the ACM SIGCOMM*, pages 8–18, 1990.

[5] R. Händel, M. N. Huber, and S. Schröder. *ATM Networks: Concepts, Protocols, Applications*. Addison-Wesley Publishing Co., third edition, 1998.

[6] C. R. Kalmanek, H. Kanakia, and S. Keshav. Rate control servers for very high-speed networks. In *Proc. GLOBECOM '90*, pages 12–20, 1990.

[7] A. Karlin, M. Manasse, L. Rudolph, and D. Sleator. Competitive snoopy caching. *Algorithmica*, 3(1):79–119, 1988.

[8] S. Keshav. *An Engineering Approach to Computer Networking*. Addison-Wesley Publishing Co., 1997.

[9] C. Partridge. Isochronous applications do not require jitter-controlled networks, September 1991. Appeared as Intentet RFC 1257.

[10] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Comm. of the ACM*, 28(2):202–208, 1985.

[11] A. Tanenbaum. *Computer Networks*. Prentice-Hall, Inc., third edition, 1996.

[12] The ATM Forum Technical Committee. Traffic management specification version 4.0, Apr. 1996. Available from `www.atmforum.com`.

[13] D. Verma, H. Zhang, and D. Ferrari. Guaranteeing delay jitter bounds in packet switching networks. In *Proceedings of TriComm*, April 1991.

[14] C.-S. Wu, J.-C. Jiau, and K.-J. Chen. Charcterizing traffic behavior and providing end-to-end service guarantees within ATM networks. In *Proc. of IEEE INFOCOM 97*, pages 336–344, 1997.

[15] D. Yates, J. Kurose, D. Towsley, and M. Hluchyj. On per-session end-to-end delay distributions and the call admission problem for real-time applications with qos requirements. In *Proceedings of ACM SIGCOMM*, Sept. 1993.

[16] H. Zhang. Service disciplines for guaranteed performance service in packet-switched networks. *Proceedings of the IEEE*, 83(10), Oct. 1995.

[17] H. Zhang and D. Ferrari. Rate-controlled service disciplines. *Journal of High Speed Networks*, 3(4), 1994.

[18] H. Zhang and S. Keshav. Comparison of rate-based services disciplines. In *Proceedings of ACM SIGCOMM*, 1991.

[19] L. Zhang. *A New Architecture for Packet Switched Network Protocols.* PhD thesis, Massachusetts Institute of Technology, July 1989.