Optimal Lexicographic Shaping of Aggregate Streaming Data

Stergios V. Anastasiadis, Peter Varman, *Senior Member*, *IEEE*, Jeffrey Scott Vitter, *Fellow*, *IEEE*, and Ke Yi

Abstract—We investigate the problem of smoothing multiplexed network traffic when either a streaming server *transmits data* to multiple clients or a storage server *accesses data* from multiple storage devices or other servers. We introduce efficient algorithms for lexicographically optimally smoothing the aggregate bandwidth requirements over a shared network link. Possible applications include improvement in the bandwidth utilization of network links and reduction in the energy consumption of server hosts. In the data transmission problem, we consider the case in which the clients have different buffer capacities and unlimited bandwidth constraints or unlimited buffer capacities and different bandwidth constraints. For the data access problem, we handle the general case of a shared buffer capacity and individual network bandwidth constraints. Previous approaches for the data access problem handled either the case of only a single stream or did not compute the lexicographically optimal schedule. By provably minimizing the variance of the required aggregate bandwidth, lexicographically optimal smoothing makes the maximum resource requirements within the network more predictable and increases the useful resource utilization. It also improves fairness in sharing a network link among multiple users and makes new requests from future clients more likely to be successfully admitted without the need for rescheduling previously accepted traffic. With appropriate hardware and system support, data traffic smoothing can also reduce the energy consumption of the host processor and the communication links. Overall, we expect that efficient resource management at the network edges will better meet quality of service requirements without restricting the scalability of the system.

Index Terms—Data smoothing, streaming, prefetching, energy efficiency, content distribution networks, client/server systems, optimization.

1 Introduction

Emerging applications that involve real-time delivery of digital content motivate the deployment of scalable data storage and transfer infrastructure with support for quality-of-service guarantees. However, the high costs of installing and maintaining network and storage equipment make bandwidth overprovisioning an undesirable approach for handling unpredictable surges in user demand. Costly energy consumption in modern data centers is another reason for trying to reduce peaks in the data transfer capacity required over time [1]. Previous published literature has already made the case that appropriate shaping of the data traffic at the network edges has the potential to achieve higher resource utilization, without imposing scale limitations onto the network architecture [2], [3]. In the present paper, we develop algorithms for smoothing out the aggregate bandwidth required during on-demand delivery of stored media files to multiple different clients. We expect similar approaches to also be applicable in other similar environments, such as ad hoc and sensor networks, where individual nodes are both clients and server hosts and can only use limited resources including network bandwidth and energy [4].

Fig. 1 illustrates a simplified network hierarchy that allows on-demand delivery of stored media files to multiple users. By requesting stored streams with known time-dependent resource requirements, the clients essentially specify what data are required and when each portion is actually needed. The source server node manages a collection of secondary storage devices, where the media files are originally stored. Memory space in the server is available for buffering or caching data arriving from the disks. At one or more intermediate layers, proxy servers between the clients and the source server are responsible for handling the client requests and appropriately communicating them back to the source server node. Data transmission to the clients is adjusted according to the buffer space and network link capacity of each client and each proxy.

In this paper, we consider the generic problem of scheduling network data transfers for streaming media files. In Fig. 1, downstream nodes are on the left and upstream nodes on the right. A practical and scalable approach is to do the scheduling one layer at a time, starting at the downstream layers and proceeding to the upstream layers. In the course of this layer-by-layer process, the scheduling problem that arises at each layer falls into one of two basic types:

1. At the proxy-client layer, we have an instance of the *data transmission problem*, where multiple clients

- *S.V. Anastasiadis and K. Yi are with the Department of Computer Science, Duke University, Durham, NC 27708-0129. E-mail: {stergios, yike}@cs.duke.edu.*
- *P. Varman is with the Department of Electrical and Computer Engineering, Rice University, Houston, TX 77005. E-mail: pjv@ece.rice.edu.*
- *J.S. Vitter is with the School of Science, Purdue University, West Lafayette, IN 47907-2068. E-mail: jsv@purdue.edu.*

Manuscript received 12 Nov. 2003; revised 23 Oct. 2004; accepted 10 Nov. 2004; published online 15 Feb. 2005.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0209-1103.

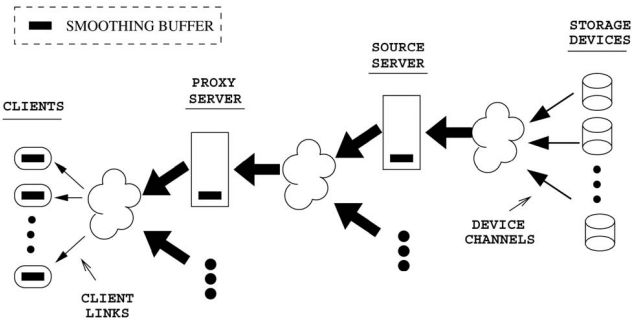

Fig. 1. Hierarchical system model for streaming data distribution. Downstream nodes are on the left and upstream nodes on the right.

share the network transmission bandwidth. After this schedule is determined, we are left with a set of data requests (with modified deadlines) that are passed upstream from the proxy to the server (or perhaps to another intermediate proxy), resulting in another instance of the data transmission problem.

2. After scheduling the data transmissions from the server to the proxies, we are left with a set of data requests (with modified deadlines) that must be passed upstream to the disks. The resulting *data access problem* has the characteristics that the server has a single shared buffer to service disk channels with constraints on individual data transfer rates.

Further variants and combinations of these two problems are possible for more complicated networks. We briefly discuss caching issues, which are the subject of ongoing research, in Section 8.

For both the data transmission and data access problems, we introduce efficient algorithms for *lexicographic optimal smoothing* (or simply *lexopt smoothing*) of the aggregate bandwidth requirements. Among all data transfer schedules, the *lexopt* schedule is the one with minimum bandwidth variance. In the data transmission problem, where multiple clients share the network bandwidth, we consider two client models. In the first, the clients have different buffer capacities, but no individual network bandwidth constraints; in the second model, the clients have no buffer limitations, but different link bandwidth capacities. For the data access problem, we handle the general case of a shared buffer with limited capacity and individual disk channel bandwidth constraints. Previous approaches in the literature for the data transmission and data access problems handled either the case of only a single stream or did not compute the lexicographically optimal schedule. We examine previous related work in Section 2.

Lexopt smoothing of the aggregate data traffic has several advantages over alternative traffic shaping methodologies. It will not only minimize the maximum aggregate bandwidth, but it will also keep the bandwidth requirements in every time step of the aggregate data transfer sequence minimal. By reducing the time period during which the peak bandwidth is actually utilized, the chances for successfully admitting requests from additional clients can be improved [5], [6]. By avoiding rescheduling the entire accepted network traffic every time a new playback request is considered, the computation for admission control becomes independent of the number of clients concurrently supported by the system. The hierarchical approach for applying the *lexopt* process itself distributes the smoothing operation across multiple layers in order to take advantage of buffering resources available in the entire system.

Recent research in processor circuits and network links has also demonstrated how data traffic smoothing can reduce the energy consumption of the host processor and the communication links [7], [8]. Energy usage can be assumed to be a convex function of processor speed, which varies according to the circuitry voltage level. By reducing the maximum transfer rate in a system, we can operate the processor at a lower speed and the circuit at a lower voltage level. As a result, traffic smoothing can reduce the energy consumed by the system. We should make it clear that, by applying the methods that we introduce in the present paper, we reduce the maximum bandwidth required in order to transfer multiplexed data through a network link, while the average consumed bandwidth remains the same. On the other hand, energy consumption is a convex function of the processor speed that depends on the voltage level of the circuit. If we reduce the voltage due to lower demand for processing speed, essentially we end up with lower energy consumption.

The rest of this paper is structured as follows: In Section 2, we present previous related work. In Section 3, we define the general data transmission problem, while, in Sections 4 and 5, we describe algorithms for optimally solving two important cases. In Section 6, we more formally introduce the data access problem and we provide an optimal algorithm for a natural model of the system. In Section 7, we summarize some results from our experimentation with actual video streams, while Section 8 briefly discusses our conclusions and plans for future work.

2 Related Work

Variable bit-rate encoding of video streams can achieve quality equivalent to constant bit-rate encoding while requiring an average bit rate that is significantly lower [9], [10], [11]. Researchers have previously applied data prefetching techniques in order to reduce bandwidth requirements of variable bit-rate data transfers.

We compare different data transfer sequences with respect to uniformity of bandwidth requirements over time by using the lexicographic optimality criterion [12], [13] that we formally specify in Section 3. Previously, Salehi et al. described an algorithm that lexicographically optimizes the data transfer sequence of a single stream when given a limited client buffer capacity for prefetching [5]. Feng and Rexford compare alternative methods for smoothing individual streams and demonstrate differences in the rate change properties of the generated transfer sequences [14]. In particular, they examine a smoothing method that keeps the transfer rate of individual streams below a bandwidth constraint and transmits frames as late as possible. Despite its similarity with part of a method (pruning) that we present in the present paper, the purpose of that application was to minimize the buffer utilization at the client rather than smoothing the aggregate bandwidth. McManus and Ross introduce a general dynamic programming approach

for generating a piecewise constant rate sequence for a single stream, optimized according to criteria related to the required buffer space and transfer rate [15]. All these solutions apply to individual streams without explicit consideration of contention for network link bandwidth from other concurrently served clients.

Similar optimization objectives have also been applied to various other contexts. Hoang et al. propose a lexicographic approach for choosing the quantization level during MPEG video coding, subject to buffering constraints. Their target is to achieve the most uniform visual quality of the transmitted video [16]. Rexford et al. use client data prefetching for smoothing live video streams, where the stream requirements are known only for a limited period of time instead of the entire playback period [17]. Mansour et al. examine several resource trade offs in live video smoothing [18]. Other studies have considered server-side resource management [19], [20], [21]. Anastasiadis et al. investigate the problem of lexicographically optimizing the bandwidth of individual streams transferred from multiple disks into a shared buffer [22].

Zhao and Tripathi describe an algorithm for minimizing the maximum aggregate bandwidth required when multiplexing several video streams over a common network link connected to clients with different buffer constraints [6]. In a preliminary publication, we expanded on that work to achieve lexicographically optimal smoothing of the aggregate bandwidth rather than minimizing only the peak bandwidth of the multiplexed traffic [23]. Additionally, in the present paper, we motivate the aggregate smoothing problem with applications to processor and link energy savings, we strengthen our results with extended complexity analyses, and we improve the readability of our proofs by clearly explaining them in more detail.

Yao et al. introduce an offline processor scheduling algorithm to meet job execution deadlines at minimum energy consumption [24]. They assume that energy usage is a convex function of processor speed and that processor speed can vary according to the circuitry voltage level [7]. Lorch and Smith consider alternative sampling and distribution estimation methods to construct piecewise processing schedules. Their goal is to effectively approximate the theoretical schedule with the minimum energy consumption [25]. Elnozahy et al. combine hardware voltage scaling with software request scheduling to reduce processor energy usage [1]. Even though the job scheduling problem is similar to several aspects of data transfer scheduling, there are main differences between the two due to the buffering constraints that are only involved in the latter and the advance knowledge that is available when streaming stored data. Shang et al. describe hardware that uses historical utilization information to dynamically adjust the frequency and voltage of network links in order to minimize the power they consume [8]. Instead, we describe techniques for data transfer smoothing that can be used for reducing the energy involved in both the server processor and the network links.

In our experimentation, we reserve the resources deterministically, rather than estimating their requirements from data transfer statistics. In fact, accurate statistical representation of video traffic is a nontrivial problem, which we don't need to solve in our case because we take advantage of the detailed information available in the server for each stored stream. Prefetching techniques can be successfully applied explicitly through smoothing, as is demonstrated in the rest of this paper and previous related work. They increase buffer space requirements at the client side in order to improve the network bandwidth utilization and to reduce the energy consumption in processors and data links.

TABLE 1
Summary of Symbols

Symbol	Description
k, $1 \le k \le K$	client index
i, $0 \le i \le T$	time step index
$(i, j]$, $1 \le i < j \le T$	half-open time interval from i to j
$d_k(i)$	data demand of client k at time step i
$d_k^p(i)$	pruned data demand of client k at time step i
$L_k(i)$	cumulative lower bound of $d_k(i)$
$L_A(i)$	aggregate cumulative lower bound of $d_k(i)$
$U_k(i)$	cumulative upper bound of $d_k(i)$
$U_A(i)$	aggregate cumulative upper bound of $d_k(i)$
$s_k(i)$	transferred data of client k at time step i
$s_A(i)$	aggregate transferred data at time step i
ρ_k	bandwidth capacity of client k
R^c	critical aggregate data transfer rate
μ_k	buffer capacity of client k
M	server buffer capacity

3 Definition of the Data Transmission Problem

We consider a set of K clients connected to a source (or proxy) streaming media server over a shared network link. We describe each stream k with the *demand sequence* $\mathbf{d_k}$ of length T. The demand sequence specifies the minimum data requirements $d_k(i)$ at time step i, $1 \le i \le T$. Therefore, the client k must receive $d_k(i)$ bytes at time step i from stream k in order for the stream decoding to continue uninterrupted. We can transfer (prefetch) the requested data to the client earlier provided that we can maintain them in the client's buffer until their actual use. We define as $L_k(i)$ the cumulative minimum amount of data that the client must receive by time step i or, equivalently, $L_k(i) = \sum_{q \le i} d_k(q)$. For reading convenience, we summarize the symbols that we use in the present article in Table 1.

When the local client buffer space is limited, $U_k(i)$ is the maximum cumulative amount of data the client k can receive by time step i. In general, we have to satisfy the constraint $U_k(i) \ge L_k(i)$, for all $1 \le i \le T$. Assuming the available buffer space at client k is equal to μ_k, the cumulative upper bound can be derived from the lower bound: $U_k(i) = L_k(i-1) + \mu_k$. In addition, the link that connects a client k to the network may have limited bandwidth capacity ρ_k. We define the sequence $\mathbf{s_k}$ that specifies the amount of data $s_k(i)$ transferred from stream k to client k during time i as the *transfer schedule* of an individual stream k. A schedule is valid when it transfers to client k the amount of data equal to the cumulative amount

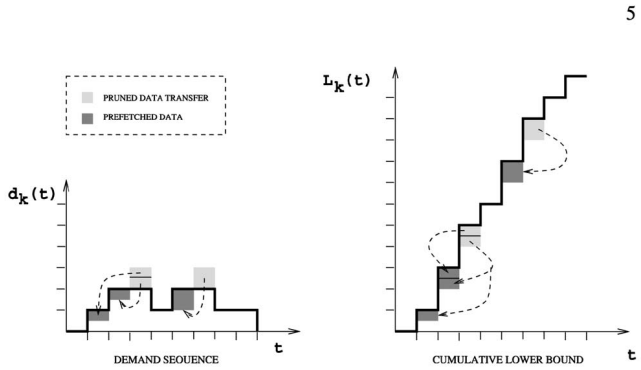

Fig. 2. Example of pruning the lower bound of an individual stream with bandwidth capacity $\rho = 2$. In both graphs, the light gray blocks refer to data units exceeding the bandwidth constraint. The dark gray blocks correspond to data that were prefetched as a result of pruning. The bold line draws a profile of the actual amount of data transferred after pruning the demand sequence.

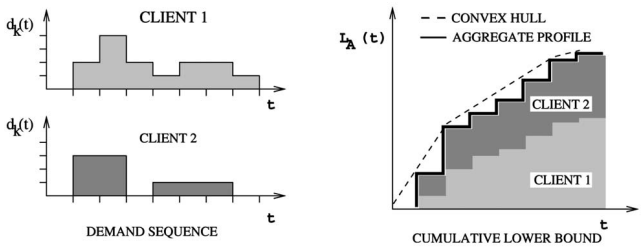

Fig. 3. Example of aggregating the pruned lower bounds of two streams for the unlimited buffer/limited bandwidth network multiplexing problem. Pruning keeps every demand sequence at each time step bounded by the corresponding bandwidth limit. Each segment on the convex hull corresponds to a critical interval, and its slope is the critical rate.

$L_k(i)$ by step i without violating the buffer and bandwidth constraints of the client. More specifically, for each client k, we have $L_k(i) \le \sum_{1 \le j \le i} s_k(j) \le U_k(i)$ and $s_k(i) \le \rho_k$, for $i = 1, \ldots, T$. The *aggregate transfer schedule* is the sequence $\mathbf{s_A}$ of the total amount of data, $s_A(i) = \sum_k s_k(i)$, transferred to all clients during time step i.

To compare different transfer schedules with respect to smoothness, we advocate using the lexicographic optimality criterion [12], [13]. This criterion compares two vectors starting from their first largest element. As long as the respective lth largest elements are the same, it proceeds to compare the $(l+1)$st largest element and so on until two respective elements differ. The vector with the smaller $(l+1)$st largest element is considered smoother than the other. Of course, when transforming a data transfer sequence to a smoother one, additional constraints apply as a result of validity restrictions mentioned in the previous paragraph.

More formally, for any $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbf{R}^n$, let the square bracket subscripts denote the elements of $\mathbf{x}$ in decreasing order $x_{[1]} \ge \cdots \ge x_{[n]}$. Given a parameter G, a sequence $\mathbf{x} \in \mathbf{R}^n$ is *valid* if $x_i \ge 0$, for each $1 \le i \le n$, and $\sum_{i=1}^n x_i = G$. For any two valid sequences $\mathbf{x}$, $\mathbf{y}$, we say that $\mathbf{x}$ is *lexicographically smaller* than $\mathbf{y}$ (denoted $\mathbf{x} \prec \mathbf{y}$) if, for some $1 \le l \le n$, we have

$$\begin{aligned} x_{[i]} &= y_{[i]}, \qquad \text{for } 1 \le i < l; \\ x_{[l]} &< y_{[l]}. \end{aligned}$$

We consider $\mathbf{x}$ to be smoother than $\mathbf{y}$ if $\mathbf{x} \prec \mathbf{y}$. We say that $\mathbf{x}^*$ is *lexicographically optimal* if $\mathbf{x}^* \prec \mathbf{x}$ for all other valid $\mathbf{x}$. As a corollary, the maximum element of $\mathbf{x}^*$ is no larger than the maximum element of any other valid sequence or, equivalently, $\mathbf{x}^*$ is *minmax-optimal* over all valid sequences. Furthermore, among all those sequences with the same smallest maximum element, $\mathbf{x}^*$ minimizes the second-highest element, and so on. Lexicographic optimality is thus a stronger notion than minmax optimality.

4 Data Transmission to Clients with Unlimited Buffer and Limited Bandwidth

In this section, we describe a newly derived algorithm for constructing a lexicographically optimal schedule for data transmission. We assume that, for each $1 \le k \le K$, client k has limited network bandwidth ρ_k, but no memory constraints (i.e., $\mu_k = +\infty$). Practically, this means that the client has sufficient storage space to receive the entire file, which allows the data blocks to be prefetched at the client any time prior to their playback deadlines. Unlimited buffer space with limited bandwidth are typical characteristics of a client device with ample local disk space and "last-mile" network connection of limited bandwidth capacity. Examples of such devices include personal computers with Internet connection and models of set-top boxes carrying hard disks in bandwidth-constrained environments.

4.1 The Algorithm

As a preprocessing step, we prune the demand sequence of each stream k so that its demand at any time step i does not exceed its link capacity ρ_k. We do this by examining successive elements of the demand sequence, starting from the last element and moving toward the first. At each step, we transfer any demand exceeding ρ_k to the previous time step. We have to move the data transfers backward in order to avoid violating any of the deadline restrictions. More specifically, we update $L_k(i) := \max\{L_k(i), L_k(i+1) - \rho_k\}$ as i decreases from $T-1$ to 1. An example of this pruning process is pictured in Fig. 2. We obtain the cumulative lower bound $L_A(i)$ at time step i by summing $L_k(i)$ for each stream k, i.e., $L_A(i) = \sum_{k=1}^K L_k(i)$, as shown in Fig. 3.

Starting from point $i = 0$ of the cumulative lower bound curve, we identify consecutive nonoverlapping longest intervals $(i, j]$. We call them *critical intervals* and choose them in a way that the line segment from $L_A(i)$ to $L_A(j)$ does not meet the lower bound anywhere other than its endpoints. The algorithm below constructs a transfer schedule so that the aggregate bandwidth for each critical interval is equal to the slope of the line segment. The resulting schedule is lexicographically optimal.

We still need to show how to schedule the data transfers. Let $(i, j]$ be a critical interval and R^c denote its *critical rate* corresponding to the slope of the line segment from $L_A(i)$ to $L_A(j)$. We start from the schedule that we constructed from the pruned lower bounds. This schedule consists of the

```
Input: T, K, ρ_k, d_k, 1 ≤ k ≤ K
Output: s_k, 1 ≤ k ≤ K, with lexicographically optimal S_A

1. transform d_k into d_k^p so that d_k^p ≤ ρ_k, 1 ≤ k ≤ K
2. generate L_k by accumulating d_k^p, 1 ≤ k ≤ K
3. generate L_A by aggregating L_k, 1 ≤ k ≤ K
4. repeat
5.     identify next critical interval (i, j] with slope R^c
6.     calculate s_k(t), i ≤ t < j, 1 ≤ k ≤ K using proof of Thm 1
7. until total length of critical intervals becomes equal to T
```

Fig. 4. Algorithm outline for the data transmission problem with unlimited buffer and limited bandwidth.

pruned demand vectors with no prefetching other than what we did during the pruning. This schedule is valid in that the bandwidth for stream k is at most ρ_k. However, the aggregate bandwidth is highly variable. We do the following step, iteratively, until the aggregate bandwidth at each time step is R^c: Let time step q be the latest time in the critical interval for which $s_A(q) > R^c$ and let time step p be the latest time step earlier than q such that $s_A(p) < R^c$. We move $\min\{s_A(q) - R^c, R^c - s_A(p)\}$ data units from time step q to p. We describe below (proof of Theorem 1) how we choose streams and how much data we move from each stream. We outline the steps of the algorithm in Fig. 4. At the end of this process, we are left with a constant-bandwidth transfer schedule of bandwidth R^c for the critical interval. This final schedule is lexicographically optimal and corresponds to an aggregate bandwidth that follows the convex hull of Fig. 3.

4.2 Optimality Proof

Lemma 1. *From the line segments connecting consecutive outer corners of the aggregate lower bound, we get a sequence of rates that correspond to a valid aggregate transfer schedule.*

Proof. Let $d_k^p(i)$ be the demand of stream k after pruning its lower bound. For each line segment connecting consecutive outer corners of the aggregate lower bound at time step i, the corresponding rate is equal to $\sum_k d_k^p(i)$. Therefore, each stream k receives bandwidth $d_k^p(i)$ at time step i, which guarantees avoidance of both data starvation and bandwidth overflow (due to pruning) during step i for client k. □

Theorem 1. *The algorithm in Section 4.1 produces the lexicographically optimal data transmission curve, which satisfies the upper and lower bounds for each stream.*

Proof. We show by construction that, in each critical interval, we obtain a valid aggregate transfer schedule such that the aggregate bandwidth for each step in the interval is equal to the critical rate of the interval. We start with the valid aggregate schedule of Lemma 1. This corresponds to having each stream scheduled according to its pruned demand sequence. Let's focus on any particular critical interval; we denote the critical rate for the interval by R^c. For the algorithm in Section 4.1, we need to show that we can transfer

$$\min\{s_A(q) - R^c, R^c - s_A(p)\}$$

data units from time step q to time step p. The result of this transfer is that either time step q or time step p will end up with an aggregate bandwidth of R^c.

The only reason for which we would not be able to transfer the desired amount of data from time q to time p is because some streams at time step p are going to be above their bandwidth limits. Since the pruned schedule is valid, the individual bandwidth constraints are initially met. In particular, the streams that are transmitting data at step q are at or below their individual bandwidth limits. Since the aggregate bandwidth at time step p is below that of time step q, there must be at least one stream transmitting at time q that is not transmitting at its bandwidth limit at time p. We can therefore move data for that stream from time step q to p. We continue in this manner until either time step q or time step p is at the desired bandwidth R^c. For each time step in the critical interval, we end up with constant aggregate bandwidth R^c. From the way R^c is computed, the average bandwidth is at least R^c for the time steps in the critical interval. The constant-bandwidth schedule for the critical interval is obviously the one that is lexicographically minimal. The concatenation of lexicographically optimal schedules for the critical intervals is lexicographically optimal. □

4.3 Algorithm Complexity

Our algorithm has complexity $O(KT)$ both for pruning the demand sequences and also for identifying and scheduling the critical intervals. In particular, pruning the demand sequence of each individual stream costs O(T), leading to O(KT) total pruning cost across all the streams. We can identify the critical intervals by applying to the aggregate lower bound (simple polygon) a convex-hull calculation algorithm with complexity O(T) [26]. Then, in each critical interval, we first separate time steps with bandwidth above Rc from those with bandwidth below Rc at total cost O(T) across all the critical intervals. Then, for each time step with aggregate bandwidth below Rc, we need to identify clients that have not reached their link capacity at cost O(K). We visit each time step with aggregate bandwidth above Rc and prefetch excess data to time steps with aggregate bandwidth below Rc at cost O(T). The total cost of the last two calculations is O(KT). Thus, the total cost of the algorithm is actually O(KT).

5 Data Transmission to Clients with Limited Buffer and Unlimited Bandwidth

As previously, we examine the scheduling of K distinct streams, each consisting of T transmission steps. We assume that each client k has a fixed amount of memory μ_k for prefetching and unlimited network link capacity $\rho_k = +\infty$. At each step, we allow real-valued amounts of data to be transferred. Our goal is to compute the *lexopt* schedule. The case of limited buffer space and unlimited bandwidth corresponds to a client with minimal local memory space connected to the server within an intranet or a backbone network of high capacity. Inexpensive set-top boxes connected to cable networks with abundant bandwidth qualify for the assumptions of the present section.

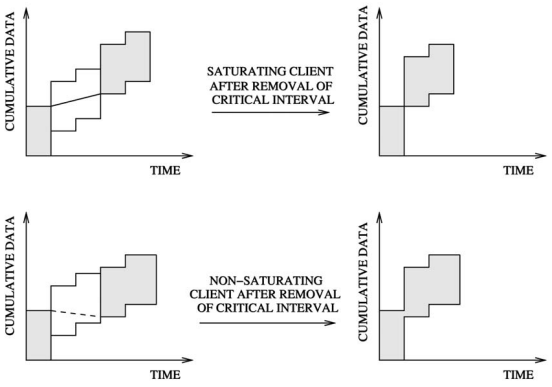

Fig. 5. Example of critical interval removal in the case of saturating and nonsaturating streams. If a stream is saturating during the critical interval, it transfers data at a minimal rate, making its full buffer at the beginning of the interval end up empty at the end of the interval. In fact, the removal of the critical interval creates a transition from full buffer to empty buffer for the saturating stream. Instead, a nonsaturating stream transfers no data during the critical interval. Therefore, the removal of the critical interval enforces some data to be prefetched in order to prevent the corresponding client from data starvation.

5.1 The Algorithm

Initially, our new algorithm follows the approach of Zhao and Tripathi [6]. For each stream k, we construct the lower and upper bounds, $L_k(i)$ and $U_k(i)$, of the stream data that we have to transmit by time step i. More specifically, the lower bound $L_k(i)$ is the sum of the demands of stream k between time steps 0 and i. The upper bound $U_k(i)$ is μ_k units larger than $L_k(i-1)$. The upper bound limits the amount of data over and above the aggregate demand that can be prefetched at time i for use in later time steps. Our algorithm is iterative. In each iteration, we schedule the transmissions for one or more disjoint intervals, $(i, j]$. Then, we remove these *critical intervals* from further consideration by the algorithm. When we remove the critical interval $(i, j]$, the demands at time step i and the time steps following j are modified to appropriately reflect the transmission sequence computed for the interval $(i, j]$. The modified workload is the input to the next iteration of the algorithm. Each individual interval is found using the algorithm of Zhao and Tripathi. Our contribution is to show how we can remove each identified critical interval and adjust appropriately the transmission sequence in order to compute the *lexopt* schedule.

Every critical interval has an associated critical rate R^c. This is the unavoidable aggregate bandwidth required during that interval. All intervals identified in the same iteration have the same value of R^c. This is strictly smaller than the critical rate in any previous iteration. In each iteration, we compute the critical rate, or the minimum bandwidth necessary to schedule the modified aggregate workload at that iteration. The sequence of critical rates that we get corresponds to the lexicographically optimal aggregate transfer schedule. Like the *minmax* algorithm of Zhao and Tripathi [6], for the peak transfer rate of the K streams, our algorithm achieves the following lower bound:

```
Input: T, K, μ_k, d_k, 1 ≤ k ≤ K
Output: s_k, 1 ≤ k ≤ K

1.  generate L_k from accumulating d_k, 1 ≤ k ≤ K
2.  calculate U_k from L_k, 1 ≤ k ≤ K and μ_k
3.  repeat
4.      identify next critical interval (i, j] of rate R^c,
         and list of saturating clients using equation 1
5.      update s_k(t), i < t ≤ j, 1 ≤ k ≤ K,
         for all saturating clients from step 4
6.      set L_k(i) = U_k(i) for each saturating client k
7.      reduce L_k(q), U_k(q) by L_k(j) − U_k(i),
         for each q > j and every saturating client k
8.      set L_k(i) = L_k(j) for every non-saturating client k
9.      remove interval (i, j] from L_k, U_k, 1 ≤ k ≤ K
10. until total length of critical intervals becomes equal to T
```

Fig. 6. Algorithm outline for the problem of data transmission to clients with limited buffers and unlimited bandwidth.

$$R^c = \max_{i<j}\left\{\sum_{1\le k\le K}\max\left\{\frac{L_k(j)-U_k(i)}{j-i},\ 0\right\}\right\}. \qquad (1)$$

This bound follows from the fact that each individual stream k at time step i has delivered an amount of data at most $U_k(i)$ to avoid buffer overflow and, at time step j, has delivered an amount of data at least $L_k(i)$ to avoid starvation. Thus, the stream needs to transfer data at minimal rate $max(\frac{L_k(j)-U_k(i)}{j-i}, 0)$ during the time interval $(i, j]$. If the difference $L_k(j) - U_k(i)$ is negative, the stream transfers no data during $(i, j]$.

Next, we consider any interval $(i, j]$ where this lower bound on peak transfer rate is achieved. As before, we call this a *critical interval*. We use an earliest deadline first approach (implicit in Lemma 2) to schedule each critical interval with constant bandwidth equal to its critical rate. For each critical interval, we call the streams that have a positive contribution to the above sum *saturating streams*, or $L_k(j) - U_k(i) > 0$. We call those streams with zero contribution to the sum *nonsaturating streams*, or $L_k(j) - U_k(i) \le 0$. Any saturating stream k satisfies two key properties:

1. It *cannot transmit* at a time step earlier than or equal to j the data needed at time steps after j.
2. The prefetch buffer of client k *must contain* exactly μ_k data units at time step i.

In other words, for each critical region, the client buffer for a saturating stream starts full and ends up empty. Nonsaturating streams do not transmit any data during the critical interval. These observations allow us to recursively reduce the problem by stripping away all critical intervals (Fig. 5). When stripping any such $(i, j]$ interval, the lower bound at the boundary times i must be adjusted in accordance with the observations above. The lower bound at time step i for each saturating stream k must be increased by $U_k(i) - L_k(i)$, while the corresponding lower bound for nonsaturating streams must be increased by $L_k(j) - L_k(i)$. In addition, the bounds of saturating streams in all time steps following j have to be reduced by the amount of bytes transferred during the $(i, j]$ interval, which is equal to the quantity $L_k(j) - U_k(i)$. Following these changes, the recursion can now be applied (Fig. 6).

If we do this stripping simultaneously for all the critical intervals, then the new problem instance will have a strictly smaller peak rate than before. We find the lower bound on the peak rates for each of the new problems and reapply the

above approach. Finally, we glue back the schedule for the interval $(i, j]$ into the resulting schedule. Because of the strong property that *every* valid schedule that meets the *minmax* bound must schedule the same transmissions during time steps $i+1, i+2, \ldots, j$, we end up with a provably lexicographically minimum schedule.

5.2 Optimality Proof

Lemma 2. *In every point of the critical interval, the aggregate transmission rate R^c is both necessary and sufficient for the clients to avoid underflow and overflow of their buffers. In other words, there is a valid schedule such that the aggregate bandwidth at each time step in the critical interval is exactly R^c.*

Proof. The above lemma is proven by Zhao and Tripathi [6]. We use that approach as a subroutine for finding the lexicographically optimal schedule. □

Theorem 2. *When multiplexing streams for clients with limited buffers, the process of iteratively identifying critical intervals generates a valid lexicographically optimal sequence of aggregate transfer rate requirements.*

Proof. From Lemma 2, we know that the aggregate rate R^c in each successive critical interval $(i, j]$ is both necessary and sufficient for the participating (saturating) stream to avoid starvation or overflow. This results from the fact that streams participating in $(i, j]$ start with a full buffer at i and end with an empty buffer at j. From the way that the lower bound is updated when stripping out the interval $(i, j]$, the requirements up to i and after j of each saturating stream are preserved. Since only saturating streams participate in $(i, j]$, the data transfer requirements of nonsaturating streams between i and j are shifted in time to point i, thus prefetching the corresponding data before or at time step i. Therefore, the original constraints of each stream are correctly maintained during the iterative process. From the way critical intervals are chosen, the *minmax* aggregate bandwidth requirements of the current problem phase are identified. Since, at each iteration, the returned bandwidth is unavoidable, the generated sequence of aggregate rates is lexicographically optimal. □

5.3 Algorithm Complexity

A straightforward implementation of the lexicographic optimization algorithm for clients with limited buffer and unlimited network bandwidth has a worst-case time complexity of $O(KT^3)$. Every time an interval is stripped off, recomputing the new minimum bandwidth takes $O(KT^2)$ time and, in the worst case, there are $O(T)$ stripping steps. However, by the use of appropriate data structures, it is possible to reduce the time for each stripping step to $O(KT \log T)$. We sketch the idea here. For each stream k, we initialize an upper-triangular matrix $\mathcal{D}_k[i][j]$, $1 \le i < j \le T$ with the values $(L_k(j) - U_k(i))/(j-i)$, for $j > i$ (Fig. 7a). In order to compute R^c, we need to determine the maximum value in each $\mathcal{D}_k$. For stripping out the interval $(s, t]$, we delete the entries in $\mathcal{D}_k$ that correspond to the vertical strips $s < j \le t$ and the horizontal strips $s < i \le t$, where $1 \le i < j \le T$. The reduced upper-triangular matrix consists of three disjoint pieces: the left triangular portion $1 \le i < j \le s$, the right triangular portion $t < i < j \le T$, and the rectangular portion $1 \le i \le s$, $t < j \le T$ (Fig. 7b).

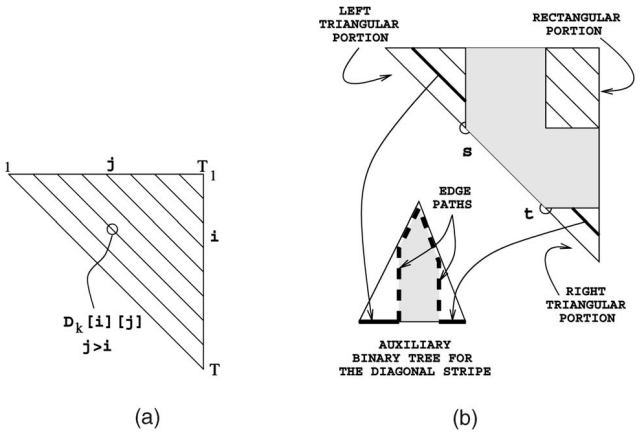

Fig. 7. (a) Upper-triangular matrix for efficiently identifying critical intervals in the algorithm for transmissions to clients with limited buffer and unlimited bandwidth. All intervals of equal length $\delta = j - i$ lie along a common diagonal. (b) Supplementary binary tree for each diagonal stripe. The highlighted part has become inactive due to the removal of the interval $(s, t]$.

After stripping off a critical interval, we update the maxima of the matrix by organizing it as a set of $T-1$ diagonal stripes, where stripe w contains entries $\mathcal{D}_k[i][w+i]$ for $1 \le w \le T-1$. When we remove a critical interval $(s, t]$, we may break each diagonal stripe into two or three smaller segments. Each such segment lies either in the left triangular portion, the right triangular portion, or the rectangular portion of the reduced matrix. Along a diagonal stripe, the entries in the two triangular portions are unchanged by the removal of strip $(s, t]$. Entries in the rectangular region correspond to pairs (i, j) that straddle the interval $(s, t]$. The entries change value for saturating and nonsaturating streams, as discussed earlier. In particular, let d be the current value of an entry $\mathcal{D}_k[i][j]$ in the rectangular region. Then, its value after removing the interval $(s, t]$ becomes $d_{\text{new}} = d \times w/(w-\delta) - \Delta/(w-\delta)$, where $w = j - i$ is the diagonal strip, $\delta = t - s$ is the width of the stripped interval, and Δ is the amount by which $L_k(j)$ is reduced by the stripping of interval $(s, t]$. This linear transformation *preserves the maximum element* in the subset to which it is applied, permitting us to avoid recomputing d_{new} for all the pairs. Successive transformations can be composed to compute new δ and Δ values, requiring only one pair of values to be maintained for each segment that survives after the removal of several critical intervals.

We organize each diagonal strip as an independent complete binary tree with the matrix elements as its leaves. Initially, the entire diagonal forms one strip. The intermediate tree nodes initially contain the maximum elements of their respective subtrees. As intervals are stripped off, contiguous sections of the leaves of the binary tree are made invalid. In any iteration, at most two sections are invalidated. To update the maxima, one has to traverse the tree along the edges of the invalidated sections, moving from the leaves to the root and recomputing the new maxima along each edge path. During this update, the new values of maxima encountered along the path are evaluated using the

transformation above. There are only a constant number of such new endpoints created in every iteration and the traversal from the endpoint to the root requires only a constant-time operation at every node: applying the transformation on the old maxima encountered in the path, invalidating any sibling that is the root of a subtree with invalidated leaves, and computing the new maxima of the (noninvalidated) children of every node on the path. Thus, for each interval stripped away, the update time per diagonal is $O(\log T)$; since there are at most $T-1$ diagonals in any matrix and K streams, the worst-case complexity for each stripping step is $O(KT \log T)$ and the total complexity of *lexopt* becomes $O(KT^2 \log T)$.

6 Shared Buffer Data Access over Limited Bandwidth Channels

We now focus on a different part of a content distribution network. While, until now, we were examining the data traffic transmitted from a server to multiple clients or intermediate proxy nodes, here we study the transfer of data from multiple storage devices into the server. The two problems are complementary since they refer to two distinct parts of the streaming infrastructure. We consider the general data access problem of accessing data using a shared buffer of size M. The data arrive from K different storage devices over separate channels, each with limited bandwidth ρ_k. For each storage device k, there is a separate demand sequence $\mathbf{d_k}$ specifying the amount of data $d_k(i)$ that should be received from that device during time step i. Our objective is to lexicographically optimize the bandwidth requirements of the aggregate data traffic arriving from all the channels. This is useful for the particular case that the shared link connecting the individual channels into the server memory is the bottleneck resource. By achieving the *lexopt* objective, we expect that we can improve the future chance of successfully accepting into the server newly arriving streams.

6.1 The Algorithm

First, we obtain the cumulative lower bound $L_A(i)$ by summing up the corresponding $L_k(i)$s of each stream: $L_A(i) = \sum_{k=1}^{K} L_k(i)$. Next, we compute the cumulative upper bound $U_A(i)$ as $U_A(i) = L_A(i-1) + M$. We then prune $L_k(i)$ for each individual stream, as we described in Section 4.1, and update the cumulative lower bound $L_A(i)$ accordingly. The cumulative upper bound remains unmodified through the pruning process. In order to identify the sequence of aggregate rates required over time, we treat the cumulative bounds as bounds of an individual stream. Then, we apply a smoothing algorithm similar to the shortest-path calculation algorithms for individual streams [27], [5]. As Salehi et al. [5] described previously, we can construct an optimal smooth schedule using linear segments. To ensure the best possible smoothness, each linear segment should be as long as possible. When we change the transfer rate to avoid buffer overflow or starvation, we do so as early as possible. We set the lower and upper bounds to $L_A(i)$ and $U_A(i)$, respectively. The total complexity is $O(KT)$, because the construction of the lower and upper bounds requires $O(KT)$ time and running the smoothing algorithm on a single stream (L_A, U_A) takes $O(T)$ time. Our algorithm is outlined in Fig. 8.

```
Input: T, K, M, ρ_k, d_k, 1 ≤ k ≤ K
Output: s_k, 1 ≤ k ≤ K

1. generate L_k by accumulating d_k for each k, 1 ≤ k ≤ K
2. generate L_A by aggregating L_k across all k, 1 ≤ k ≤ K
3. derive U_A from L_A and the buffer capacity M
4. prune L_k, 1 ≤ k ≤ K to never exceed ρ_k and update L_A accordingly
5. calculate s_A by applying a shortest-path algorithm between L_A and U_A
6. update s_k, 1 ≤ k ≤ K, from s_A derived in step 5
```

Fig. 8. Algorithm outline for the data access problem with shared buffer and channels with limited bandwidth.

6.2 Optimality Proof

We prove the optimality of the schedule generated by the above algorithm by using an iterative process of identifying critical intervals [6], rather than the functionally equivalent shortest-path algorithm that returns critical rates [27]. As before, we define as the critical interval in each iteration the interval with the line of highest slope connecting the upper and lower bound at the beginning and end of the interval, respectively. Previous work by Salehi et al. [5] has shown that the lexicographically optimal schedule for (L, U) is exactly the shortest path that connects the first and last point of the lower bound and always remains within the permissible region specified by the lower and upper bounds.

Lemma 3. *In each critical interval, we meet the minimum data access requirements for all channels without any violation of maximum bandwidth or shared buffer capacity constraint.*

Proof. There is no overflow of the shared buffer space and the data access requirements for all channels are met as a consequence of always remaining between the cumulative lower and upper bounds. From the definition of the critical interval $(i, j]$, if there is some point j', for $i < j' < j$, where a lower bound violation occurs, then j' would have been chosen instead of j as the right endpoint of the critical interval. Otherwise, the critical rate of the interval $(i, j']$ would be higher. Similarly, if there is some time step i', for $i < i' < j$, where buffer overflow occurs, the i' would be the preferred left endpoint of the critical interval instead of i.

In order to prove that there is no point where the bandwidth capacity of a channel has to be exceeded, we show two claims:

1. The critical rate never exceeds the sum of the bandwidth capacities of the channels.
2. A channel never needs to exceed its bandwidth capacity in order to avoid violating its lower bound.

In order to show Claim 1, we notice that the critical rate is calculated as the exact rate to reach a lower bound point from an upper bound point of the aggregate requirements. Note that each upper bound is no less than the corresponding lower bound. Since we have already pruned the lower bounds of the individual streams, a critical rate cannot exceed the sum of the bandwidth capacities of the individual streams. When we strip off a critical interval, the lower bound of the left endpoint is raised to the height of the upper bound (as shown in Fig. 5 for a saturating client). Our claim remains valid in

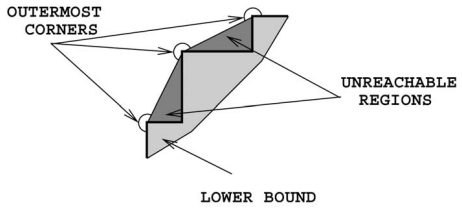

Fig. 9. The two dark gray triangles depict the region that is unreachable by the channel transmission schedule.

that case because critical rates can only decrease from one iteration to the next.

For Claim 2, we need to show that there is no underflow risk when the transfer rate of each stream does not exceed its bandwidth capacity. In any interval of length one, a channel cumulative transmission curve does not need to lie below the line connecting two consecutive lower bound corners of the channel. This is a result of the way the aggregate critical rate is defined. An example is shown in Fig. 9, where the data transmission remains at the left of the region specified by the two dark gray triangles. The individual channel transfer schedules s_k can be derived from the aggregate transfer schedule s_A as follows: The available aggregate bandwidth at each time step is distributed across the individual channels such that each of them meets its next lower bound. Any excessive bandwidth from the current step is used to meet the lower bound of the second-next step of each channel and so on. □

Theorem 3. *The aggregate transfer schedule returned by the algorithm is valid and lexicographically optimal.*

Proof. The validity is a result of Lemma 3, while the optimality is a consequence of the shortest-path approach followed for constructing the aggregate schedule. □

7 Experimentation and Discussion

The ability of *lexopt* smoothing to minimize the maximum utilized bandwidth during the entire aggregate transfer schedule can improve the probability of accepting new playback requests when transmitting data to clients with limited buffer space. In Fig. 10 and Fig. 11, we can observe the bandwidth requirements over time when the aggregate transfer schedule of different streams has been smoothed. For smoothing, we use the *minmax* method by Zhao et al. [6], the *single smoothing* method by Salehi et al. [5] applied on individual streams, and *lexopt*. In these experiments, we use six different MPEG-2 clips with distinct statistical features that were generated with variable bit-rate encoding parameters [22].

When multiplexing three streams of low bit-rate variability (Fig. 10), the difference between the average and the maximum required aggregate bandwidth is relatively small. Nevertheless, we notice how *lexopt* achieves the mostly uniform bandwidth utilization during the entire streaming period. With *single smoothing*, since the algorithm ignores the aggregate bandwidth requirements, there are several local peaks that the algorithm fails to remove. On

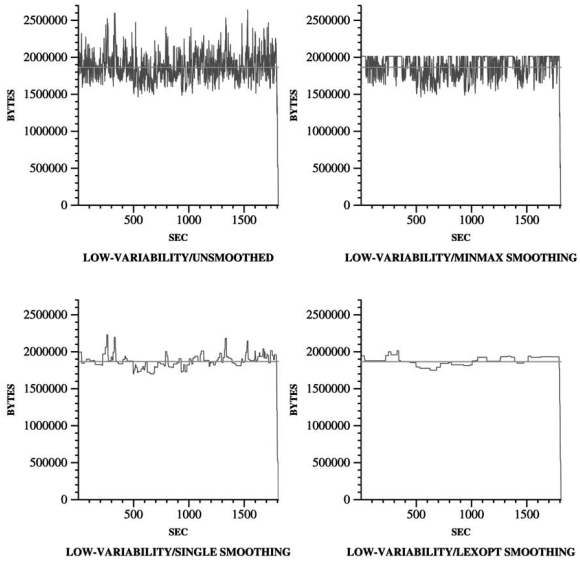

Fig. 10. For video streams of low variability, the local maxima of the bandwidth required over time are closer to the average when using lexopt smoothing in comparison to no smoothing, minmax smoothing, and single smoothing. The three smoothing methods were applied to the bandwidth requirements of three different streams with low variability and 2MB buffer space per client.

the other hand, the *minmax* algorithm keeps the aggregate bandwidth highly variable even though it manages to crop the maximum. In Fig. 11, we repeat the experiment by multiplexing three streams of high bit-rate variability. This makes the benefit of smoothing more visible and, in the time period after 1,000 seconds, *lexopt* keeps the required bandwidth about 20 percent lower than *minmax*. This is the result of better managing the buffer space available across

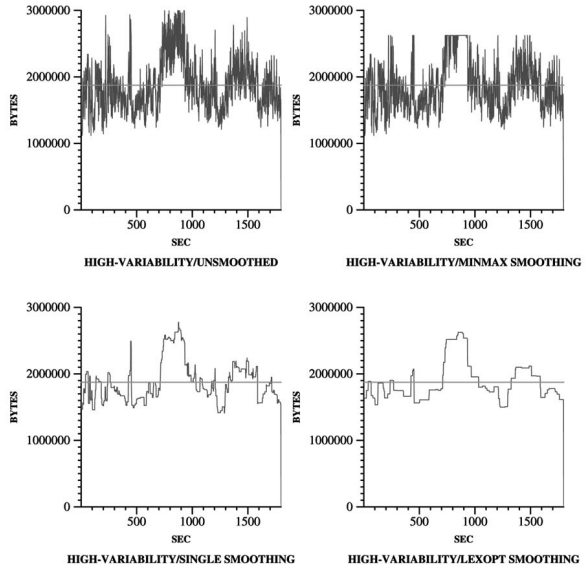

Fig. 11. For streams of high variability, lexopt eliminates significantly more peaks of aggregate bandwidth in comparison to no smoothing, minmax smoothing, and single smoothing. The alternative methods were applied to the bandwidth requirements of three different streams with high variability and 2MB buffer space per client.

the different clients in order to avoid coincidence of peak bandwidth requirements across the different streams. Again, in comparison to *single smoothing*, *lexopt* more effectively eliminates the aggregate bandwidth peaks, such as the one at the 400 seconds time point.

The actual benefit of smoothing can be further quantified through the throughput achieved over a network link when stream requests arrive dynamically over time and are considered for playback. The achieved performance can be affected by several parameters that include the buffer space available for data prefetching at each client, the rate variability of the requested streams, the load arriving into the system, and the scheduling policy used for deciding when a new client can be admitted. In particular, the system load should be kept under control in order to avoid a high request rejection ratio, which makes the service impractical. Possible approaches for the scheduling policy depend on how often the admitted data transfer traffic is reorganized to take advantage of knowledge for resource requirements that becomes available with new client arrivals. However, periodical rescheduling of the entire accepted transfer traffic can become expensive as it increases linearly with the total number of active streams in the system. Furthermore, the advantage from data prefetching through smoothing is reduced as the network link capacity increases and the benefit from statistical multiplexing becomes larger.

An important issue that is not addressed by the present study is adaptation of the aggregate transfer rate requirements according to network bandwidth conditions that vary over time. Such a scenario would more closely resemble best-effort assumptions currently made for the Internet traffic. Instead, we assume that different streams are multiplexed over a link of bandwidth capacity that remains fixed, as would be the case with privately owned links. The appeal of applying lexicographical smoothing to multiplexed traffic when transmitting data to clients with limited bandwidth is similar to the case of limited buffer. A comparable advantage could be achieved when accessing data into shared memory from multiple source devices, assuming that the bottleneck resource is the bandwidth of the shared link connecting the devices to the server. One restriction for accepting new traffic when accessing data from multiple storage devices arises from the fact that, typically, the requested data is only stored on one source device. Should this become an issue, replication of data across multiple devices could be used for better balancing the system load across the different devices, as storage space becomes inexpensive relative to the storage device bandwidth.

8 Conclusions and Future Work

We address the problem of shaping the multiplexed network traffic of streaming data in order to smooth their aggregate bandwidth in a lexicographically optimal way (*lexopt* smoothing). We consider *lexopt* smoothing in the complementary problems of data transmission to multiple clients and data access from multiple storage devices. In the context of the data transmission problem, we developed *lexopt* smoothing algorithms for two client models: clients that have different buffer capacities but no individual bandwidth constraints and clients that have different bandwidth constraints but no buffer limitations. In the context of the data access problem, we developed the *lexopt* smoothing algorithm for the general case of a shared buffer with limited capacity and individual data rate constraints.

Lexopt smoothing has several desirable features that result in increased resource utilization and flexibility for handling unknown future traffic during hierarchical scheduling. By provably minimizing the variance of the required aggregate bandwidth, maximum resource requirements within the network become more predictable and useful resource utilization increases. We can improve fairness in sharing a network link among multiple users and new requests from future clients are more likely to be successfully admitted without the need for rescheduling previously accepted traffic. Finally, we can take advantage of the reduced maximum bandwidth toward operating the host processor and the communication links at lower voltage and achieving lower energy consumption.

It remains open how to solve the general data transmission problem in which the clients differ in both their buffer capacities and bandwidth constraints. Another key issue in the performance of content distribution networks is caching, which filters the amount of data requested from the upstream nodes and reduces the corresponding control traffic sent across the network. Also, by adjusting the length (granularity) of individually requested file segments, we can trade scheduling flexibility for reduced control traffic needed to specify the segments received by different clients over time.

Acknowledgments

Support was provided in part by the US National Science Foundation through research grants CCR-9877133 and CCR-0082986, by the US Army Research Office through grants DAAD19-01-1-0725 and DAAD19-03-1-0321, and by an IBM Faculty Research Grant.

References

[1] M. Elnozahy, M. Kistler, and R. Rajamony, "Energy Conservation Policies for Web Servers," *Proc. USENIX Symp. Internet Technologies and Systems,* pp. 99-112, Mar. 2003.

[2] I. Stoica, S. Shenker, and H. Zhang, "Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks," *Proc. ACM SIGCOMM Conf.,* pp. 118-130, Sept. 1998.

[3] B. Yener, G. Su, and E. Gabber, "Smart Box Architecture: A Hybrid Solution for IP QoS Provisioning," *Computer Networks J.,* vol. 3, no. 3, pp. 357-375, 2001.

[4] S.-B. Lee, G.-S. Ahn, X. Zhang, and A.T. Campbell, "Insignia: An IP-Based Quality of Service Framework for Mobile Ad Hoc Networks," *J. Parallel and Distributed Computing,* vol. 60, no. 4, pp. 374-406, Apr. 2000.

[5] J.D. Salehi, Z.-L. Zhang, J.F. Kurose, and D. Towsley, "Supporting Stored Video: Reducing Rate Variability and End-to-End Resource Requirements through Optimal Smoothing," *IEEE/ACM Trans. Networking,* vol. 6, no. 4, pp. 397-410, Aug. 1998.

[6] W. Zhao and S.K. Tripathi, "Bandwidth-Efficient Continuous Media Streaming through Optimal Multiplexing," *Proc. ACM SIGMETRICS Conf.,* pp. 13-22, June 1999.

[7] T.D. Burd, T.A. Pering, A.J. Stratakos, and R.W. Brodersen, "A Dynamic Voltage Scaled Microprocessor System," *IEEE J. Solid-State Circuits,* vol. 35, no. 11, pp. 1571-1580, Nov. 2000.

[8] L. Shang, L.-S. Peh, and N.K. Jha, "Dynamic Voltage Scaling with Links for Power Optimization of Interconnection Networks," *Proc. Int'l Symp. High Performance Computer Architecture,* pp. 91-102, Feb. 2003.
[9] A.R. Reibman and A.W. Berger, "Traffic Descriptors for VBR Video Teleconferencing over ATM Networks," *IEEE/ACM Trans. Networking,* vol. 3, no. 3, pp. 329-339, June 1995.
[10] S. Gringeri, K. Shuaib, R. Egorov, A. Lewis, B. Khasnabish, and B. Basch, "Traffic Shaping, Bandwidth Allocation, and Quality Assessment for MPEG Video Distribution over Broadband Networks," *IEEE Network,* vol. 12, no. 6, pp. 94-107, Nov./Dec. 1998.
[11] T.V. Lakshman, A. Ortega, and A.R. Reibman, "VBR Video: Tradeoffs and Potentials," *Proc. IEEE,* vol. 86, no. 5, pp. 952-973, May 1998.
[12] T. Ibaraki and N. Katoh, *Resource Allocation Problems: Algorithmic Approaches.* Series on the Foundations of Computing, MIT Press, 1988.
[13] D.T. Hoang and J.S. Vitter, *Efficient Algorithms for MPEG Video Compression.* New York: John Wiley & Sons, 2002.
[14] W.-C. Feng and J. Rexford, "Performance Evaluation of Smoothing Algorithms for Transmitting Prerecorded Variable-Bit-Rate Video," *IEEE Trans. Multimedia,* vol. 1, no. 3, pp. 302-313, 1999.
[15] J. McManus and K. Ross, "A Dynamic Programming Methodology for Managing Prerecorded VBR Sources in Packet-Switched Networks," *Telecomm. Systems,* vol. 9, pp. 223-247, 1998.
[16] D.T. Hoang, P.M. Long, and J.S. Vitter, "Efficient Cost Measures for Motion Compensation at Low Bit Rates," *Proc. IEEE Data Compression Conf.,* Apr. 1996.
[17] J. Rexford, S. Sen, J. Dey, W. Feng, J. Kurose, J. Stankovic, and D. Towsley, "Online Smoothing of Live, Variable-Bit-Rate Video," *IEEE Trans. Multimedia,* vol. 2, no. 1, pp. 37-48, Mar. 2000.
[18] Y. Mansour, B. Patt-Shamir, and O. Lapid, "Optimal Smoothing Schedules for Real-Time Streams," *Proc. ACM Symp. Principles of Distributed Computing,* pp. 21-29, July 2000.
[19] S. Paek and S.-F. Chang, "Video Server Retrieval Scheduling for Variable Bit Rate Scalable Video," *Proc. IEEE Int'l Conf. Multimedia Computing and Systems,* pp. 108-112, June 1996.
[20] S. Sahu, Z.-L. Zhang, J. Kurose, and D. Towsley, "On the Efficient Retrieval of VBR Video in a Multimedia Server," *Proc. IEEE Int'l Conf. Multimedia Computing and Systems,* pp. 46-53, June 1997.
[21] A.L.N. Reddy and R. Wijayaratne, "Techniques for Improving the Throughput of VBR Streams," *Proc. SPIE/ACM Multimedia Computing and Networking Conf.,* pp. 216-227, Jan. 1999.
[22] S.V. Anastasiadis, K.C. Sevcik, and M. Stumm, "Shared-Buffer Smoothing of Variable Bit-Rate Streams," *Performance Evaluation,* vol. 59, no. 1, pp. 47-72, Jan. 2005.
[23] S.V. Anastasiadis, P. Varman, J.S. Vitter, and K. Yi, "Lexicographically Optimal Smoothing for Broadband Traffic Multiplexing," *Proc. ACM Symp. Principles of Distributed Computing,* pp. 68-77, July 2002.
[24] F. Yao, A. Demers, and S. Shenker, "A Scheduling Model for Reduced CPU Energy," *Proc. IEEE Symp. Foundations of Computer Science,* pp. 374-382, Oct. 1995.
[25] J.R. Lorch and A.J. Smith, "Improving Dynamic Voltage Scaling Algorithms with Pace," *Proc. ACM SIGMETRICS Conf.,* pp. 50-61, June 2001.
[26] R.L. Graham and F.F. Yao, "Finding the Convex Hull of a Simple Polygon," *J. Algorithms,* vol. 4, no. 4, pp. 324-331, Dec. 1983.
[27] D.T. Lee and F.P. Preparata, "Euclidean Shortest Path in the Presence of Rectilinear Barriers," *Networks,* vol. 14, pp. 393-410, 1984.

Stergios V. Anastasiadis received the BSc (1994) degree from the Department of Computer Engineering and Informatics, University of Patras, Greece, and the MSc (1996) and PhD (2001) degrees from the Department of Computer Science, University of Toronto, Canada. Subsequently, he joined the Deparment of Computer Science, Duke University as a visiting assistant professor. His research interests include performance evaluation of computer systems, design and implementation of systems software architectures, and development of algorithms for data caching and prefetching problems.

Peter Varman is on the faculty of Rice University with appointments in the Electrical and Computer Engineering and Computer Science Departments. Since 2002, he has been serving as a program director at the US National Science Foundation (NSF) on rotation from his home institution. His research interests span several areas of parallel and distributed computing systems, including computer system architectures, parallel I/O systems, applied algorithms, and resource scheduling. At the NSF, he has managed the computer systems architecture area of the Division of Computing and Communications Foundations, is a team member of the NSF-wide NSE and IGERT initiatives, and the recent interagency taskforce (HECRTF) on high-end computing. He has held short-term visiting positions at the IBM T.J. Watson and Almaden Research Centers, at Duke University, and at NTU, Singapore. He is a senior member of the IEEE.

Jeffrey Scott Vitter received the BS degree in mathematics with highest honors from the University of Notre Dame in 1977, the PhD degree in computer science from Stanford University in 1980, and the MBA degree from Duke University in 2002. He is the Frederick L. Hovde Dean of the College of Science and Professor of Computer Science at Purdue University. From 1993 to 2002, he was the Gilbert, Louis, and Edward Lehrman Professor of Computer Science at Duke University. He also served from 1993 to 2001 as chair of the Department of Computer Science and from 1997 to 2002 as codirector of Duke's Center for Geometric and Biological Computing. From 1980 to 1993, he was on the faculty at Brown University. He is on the Board of Directors of the Computing Research Association (CRA), where he cochairs the Government Affairs Committee. He has served as chair, vice-chair, and member-at-large of ACM SIGACT and has served as a member of the Executive Council of the European Association for Theoretical Computer Science. He is a Guggenheim Fellow, an ACM fellow, an IEEE fellow, a US National Science Foundation Presidential Young Investigator, a Fulbright Scholar, and an IBM Faculty Development awardee. He is coauthor of the books *Design and Analysis of Coalesced Hashing* (Oxford University Press) and *Efficient Algorithms for MPEG Video Compression* (Wiley and Sons), coeditor of the collections *External Memory Algorithms* and *Algorithm Engineering*, and coholder of patents in the areas of external sorting, prediction, and approximate data structures. He has written numerous articles and has consulted widely in the areas of design and analysis of algorithms, external memory algorithms, data compression, databases, indexing, parallel algorithms, machine learning, random variate generation, and sampling. He serves or has served on the editorial boards of *Algorithmica*, *Communications of the ACM*, *IEEE Transactions on Computers*, *Theory of Computing Systems*, and *SIAM Journal on Computing*, and has edited several special issues.

Ke Yi received the BS degree in computer science in 2001 from Tsinghua University, China. He is now a PhD candidate in the Computer Science Department of Duke University and is expected to receive the degree in 2006.