Lexicographically Optimal Smoothing for Broadband Traffic Multiplexing

Stergios Anastasiadis* Peter Varman† Jeffrey Scott Vitter‡ Ke Yi§

Abstract

We investigate the problem of smoothing multiplexed network traffic, when either a streaming server *transmits data* to multiple clients, or a 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. In the data transmission problem, we consider the case in which the clients have different buffer capacities but no bandwidth constraints, or no buffer capacities but 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 in the literature for the data access problem handled either the case of only a single stream or did not compute the lexicographically optimal schedule.

Lexicographically optimal smoothing (*lexopt* smoothing) has several advantages. By provably minimizing the variance of the required aggregate bandwidth, maximum resource requirements within the network become more predictable, and useful resource utilization increases. Fairness in sharing a network link by multiple users can be improved, and new requests from future clients are more likely to be successfully admitted without the need for frequently rescheduling previously accepted traffic. Efficient resource management at the network edges can better meet quality of service requirements without restricting the scalability of the system.

*Department of Computer Science, Duke University, Durham, N.C. 27708–0129. Support was provided in part by the National Science Foundation through research grant CCR–0082986

†Department of Electrical and Computer Engineering, Rice University, Houston, TX 77005. Part of this work was done while visiting Duke University. Support was provided in part by the National Science Foundation through research grant CCR–0105565.

‡Department of Computer Science, Duke University, Durham, N.C. 27708–0129. Support was provided in part by the National Science Foundation through research grants CCR–9877133 and CCR–0082986 and by the Army Research Office through grant DAAD19–01–1–0725.

§Department of Computer Science, Duke University, Durham, N.C. 27708–0129.

1. Introduction

Increasing demand for real-time delivery of digital content over data networks motivates the problem of building scalable data storage and transfer systems with quality of service guarantees. Improving the network capacity by continuously adding link bandwidth into the system is not always feasible or efficient. Appropriate shaping of the data traffic at the network edge has the potential of achieving higher bandwidth utilization, without imposing scale limitations onto the network architecture [17, 18]. In the description that follows, we focus on the case of on-demand delivery of stored media files to multiple different clients. However, this is only one type of application example with potential benefit from the above services.

Figure 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 is required, and when each portion

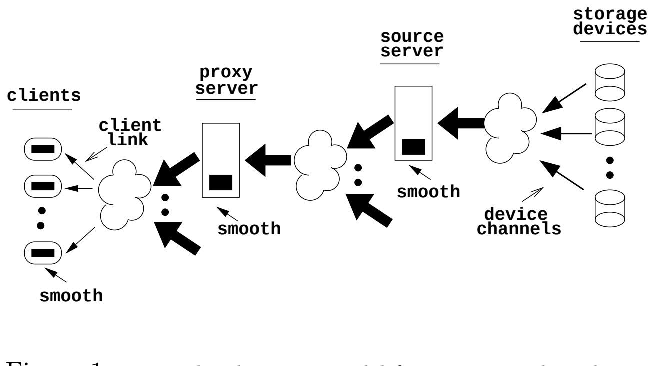

Figure 1: Hierarchical system model for streaming data distribution. Downstream nodes are on the left and upstream nodes on the right.

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 communicating them appropriately 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 Figure 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 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 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*) 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 maximum 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 work in the next section.

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 minimal the bandwidth requirements in every time step of the aggregate data transfer sequence. 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. By avoiding having to reschedule 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.

The rest of this paper is structured as follows. In Section 2, previous related work is presented. In Section 3, the general data transmission problem is defined, while in Section 4 and 5, algorithms for optimally solving two special cases are described. In Section 6, the data access problem is more formally introduced, and an optimal algorithm is provided for a particular case. In Section 7, some preliminary results from our experimentation with actual video streams are summarized, 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 average bit rate that is significantly lower [13, 3, 7]. Data prefetching techniques have been previously applied in order to reduce bandwidth requirements of variable bit-rate data transfers.

To compare different data transfer sequences with respect to uniformity of bandwidth requirements over time, we use the lexicographic optimality criterion [6, 5], formally described in the next section. Salehi et al. [16] describe an algorithm for lexicographically optimizing the data transfer sequence of a single stream, when given a limited buffer capacity at the client for prefetching. Feng and Rexford [2] compare alternative methods for smoothing individual streams and demonstrate differences in the rate change properties of the generated transfer sequences. McManus and Ross [10] 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. All these solutions apply to individual streams without explicit consideration of contention for network link bandwidth from other concurrently served clients.

Zhao and Tripathi [19] 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. Comparative experimentation is limited to static sets of streams rather than streams arriving and initiated dynamically over time, as would typically happen in practice.

Hoang and Vitter [4] propose a lexicographic approach for choosing the quantization level during MPEG video coding, subject to the buffering constraints, so as to achieve the most uniform visual quality of the transmitted video. Other related research tries to improve network link utilization for live video, where the resource requirements of a stream can only be known for a limited time window rather than the entire playback duration. Rexford et al. [14] 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. Mansour et al. [9] examine several resource tradeoffs in live video smoothing.

Other studies have considered server-side resource management [11, 15, 12]. In particular, Anastasiadis et al. [1] investigate the problem of lexicographically optimizing the transfer of individual streams transferred from multiple disks into a shared buffer.

In our experimentation, we assume that resource requirements are reserved deterministically, rather than being estimated based on statistics of the data transfer sizes. Accurate statistical representation of video traffic is a non-trivial problem, which we can avoid in the case of stored streams sequentially accessed, because of the additional information available about 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 on the client side for improved network link bandwidth utilization.

3. Data Transmission Problem Definition

We consider a set of K clients connected over a shared network link to a source (or proxy) streaming media server. Each stream k is described by the *demand sequence* $\mathbf{d_k}$ of length T, that specifies the minimum data requirements $d_k(i)$ at time step i, $1 \leq i \leq T$. That is, client k must have in its buffer $d_k(i)$ bytes of stream k at time step i in order for the decoding stream to continue uninterrupted. The actual data can be transferred earlier (prefetched), but must be held in the client's buffer until the required time. We define as $L_k(i)$ the cumulative minimum amount of data that the client must receive by time step i; in other words, $L_k(i) = \sum_{q \leq i} d_k(q)$.

When the local client buffer space is limited, the maximum cumulative amount of data that client k can receive by time step i is given by $U_k(i)$. The general constraint that has to be met is $U_k(i) \geq L_k(i)$, for all $1 \leq i \leq T$. Assuming available buffer space at the client k 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 might have limited bandwidth ρ_k.

We define the *transfer schedule* of an individual stream k to be the sequence $\mathbf{s_k}$ that specifies the amount of data $s_k(i)$ from stream k that is transferred to client k during time i. A schedule is valid when it transfers to client k the cumulative amount $L_k(i)$ by step i without violating the buffer and bandwidth constraints of the client. More specifically, for each client k, we have $L_k(i) \leq \sum_{1 \leq j \leq i} s_k(j) \leq U_k(i)$ and $s_k(i) \leq \rho_k$, for $i = 1, \ldots, T$. The *aggregate transfer schedule* is the sequence $\mathbf{s_A}$ of the total amount of data,

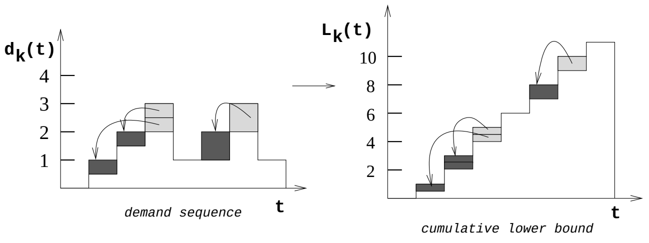

Figure 2: Example of pruning the lower bound of an individual stream with bandwidth capacity $\rho = 2$.

$s_A(i) = \sum_k s_k(i)$, transferred to all clients during time step i.

To compare different transfer schedules with respect to smoothness, we advocate using the lexicographic optimality criterion [6, 5]. This criterion compares two vectors starting from their first largest element. When the kth largest elements are the same, it proceeds with comparing the $(k+1)$st largest element and so on until two respective elements differ. The vector with the smaller $(k+1)$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 $x_{[1]} \geq \cdots \geq x_{[n]}$. Given a parameter G, a sequence $\mathbf{x} \in \mathbf{R}^n$ is *valid* if $x_i \geq 0$, for each $1 \leq i \leq n$, 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 $1 \leq k \leq n$ we have

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

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 in which, for each $1 \leq k \leq K$, 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, and thus data can be transmitted to (prefetched at) the client at any time prior to its deadline.

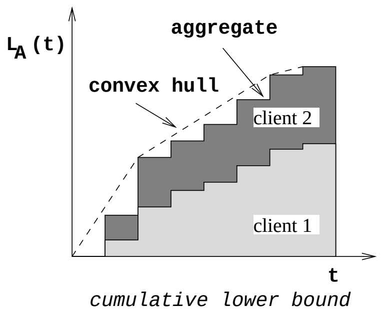

Figure 3: Example of aggregating the pruned lower bounds of two streams, for the unlimited buffer/limited bandwidth network multiplexing problem. Pruning keeps every demand sequence at each time step bounded by the corresponding bandwidth limit. Each segment on the convex hull corresponds to a critical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

4.1. The Algorithm

As a preprocessing step, the demand sequence of each stream k is pruned so that its demand at any time step i is no more than its link capacity ρ_k. This is done by examining successive elements of the demand sequence from the last element towards the first, moving at each step demand exceeding ρ_k to the previous time step. We have to move the data transfers backwards 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 Figure 2. A cumulative lower bound $L_A(i)$ at time step i is obtained by summing $L_k(i)$ for each stream k; that is, $L_A(i) = \sum_{k=1}^{K} L_k(i)$ (see Figure 3).

Starting from point $i = 0$ of the cumulative lower bound curve, we identify consecutive non-overlapping longest intervals $(i, j]$, called *critical intervals*, such 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. Our algorithm has a complexity of $O(KT)$ for pruning of the demand sequences, identification of the critical intervals, and scheduling of the critical intervals.

All that remains is to show how to schedule each critical interval. Let $(i, j]$ be a critical interval, and let R^c denote its *critical rate*, which corresponds to the slope of the line segment from $L_A(i)$ to $L_A(j)$. We start with an initial schedule consisting of the pruned lower bounds. That is, the schedule consists of the pruned demand vectors with no prefetching other than what was done 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. Precisely how we move the data (i.e., how we determine which streams and how much data from each stream) is described below in the proof of Theorem 1. The steps of the algorithm are outlined in Figure 4.

```
Input: $K$, $\rho_k$, $\mathbf{d_k}$, $1 \le k \le K$
Output: $\mathbf{s_k}$, $1 \le k \le K$ with lexopt $\mathbf{S_A}$

1. prune $\mathbf{d}_k$ into $\mathbf{d}_k^p$ according to $\rho_k$, $1 \le k \le K$
2. generate $\mathbf{L_k}$ from $\mathbf{d}_k^p$, $1 \le k \le K$
3. generate $\mathbf{L}_A$ from $\mathbf{L}_k$, $1 \le k \le K$
4. repeat
5.     identify next critical interval $(i,j]$ with $R^c$
6.     calculate $\mathbf{s}_k(t)$, $i \le t < j$, $1 \le k \le K$ (from Theorem 1)
7. until (total length of critical intervals $= T$)
```

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

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 Figure 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 corresponds to a valid aggregate transfer schedule.*

Proof: For each line segment connecting consecutive outer corners of the aggregate lower bound at time step i, the corresponding rate is equal to $\sum_k d_k^p(i)$, where $d_k^p(i)$ is the demand of stream k after pruning its lower bound. Therefore, each stream k receives bandwidth $d_k^p(i)$, 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, which corresponds to scheduling each stream 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 why we would not be able to transfer the desired amount of data from time q to time p would be if some streams at time step p would then be above their bandwidth limit. Since the initial 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. By the nature of the way R^c is computed, the average bandwidth for the time steps in the critical interval is at least R^c. 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. ∎

5. Data Transmission to Clients with Limited Buffer and Unlimited Bandwidth

As previously, we consider the scheduling of K distinct streams, each of which consists of T transmission steps. For the client model considered in this section, each client k has a fixed amount μ_k of memory for prefetching, but there is no bandwidth constraint (i.e., $\rho_k = +\infty$). We allow real-valued amounts of data transfer at each step. Our goal is to compute the *lexopt* schedule.

5.1. The Algorithm

Our new algorithm initially follows the approach of Zhao and Tripathi [19] and constructs, for each stream k, the lower bound $L_k(i)$ and upper bound $U_k(i)$ on the data for that stream that needs to be transmitted by time step i. The lower bound $L_k(i)$ is just the sum of the demands of stream k between time steps 0 and i. The upper bound $U_k(i)$, which is μ_k units larger than $L_k(i-1)$, 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, transmissions are scheduled for one or more disjoint intervals, $(i, j]$. These *critical intervals* are then removed from further consideration by the algorithm. When removing critical interval $(i, j]$, the demands at time step i and time steps greater than 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 that we can remove each identified critical interval, followed by appropriate adjustment of the transmission sequence, to compute the *lexopt* schedule.

Every critical interval has an associated critical rate R^c, which is the aggregate bandwidth required in that interval. All intervals identified in the same iteration have the same value of R^c, which is smaller than the critical rate in any previous iteration. The critical rate computed in any iteration is the minimum bandwidth necessary to schedule the modified aggregate workload at that iteration. This sequence of critical rates generates the lexicographically optimal aggregate transfer schedule.

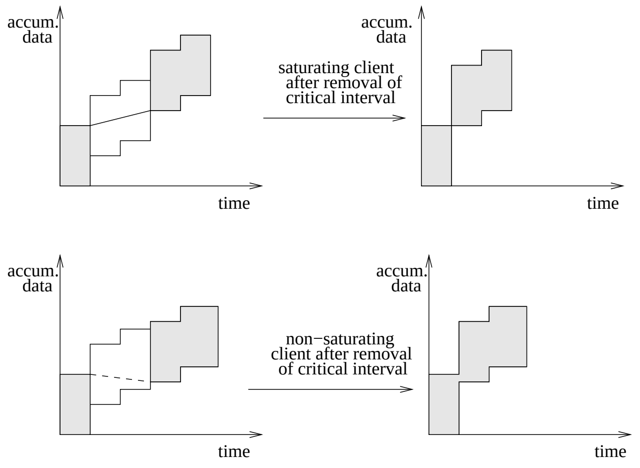
accum. data
time
saturating client after removal of critical interval
accum. data
time
accum. data
time
non−saturating client after removal of critical interval
accum. data
time

Figure 5: Example of critical interval removal in the case of saturating and non-saturating streams. Note that no data transfer occurs during the critical interval in the case of non-saturating streams.

Like the *minmax* algorithm of Zhao and Tripathi [19], our algorithm achieves the following lower bound on the peak transfer rate for the K streams:

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

Let us now consider any interval $(i, j]$ in which this lower bound on peak transfer rate is achieved. As before, call such an interval a *critical interval.* We schedule each critical interval with a constant bandwidth equal to its critical rate, using the algorithm implicit in Lemma 2.

For a critical interval, let the streams that have a positive contribution to the above sum in the critical interval be called *saturating streams*, and *non-saturating streams* be those streams with zero contribution to the sum. Any saturating stream k satisfies two key properties:

1. It *cannot transmit* any data needed for time steps beyond j in any time step less than or equal to j.
2. At time step i, there *must be* exactly μ_k data units in client k's prefetch buffer.

In other words, for each critical region, the client buffer for a saturating stream starts full and ends up empty. Non-saturating 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 (Figure 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

```
Input: $K$, $\mu_k$, $\mathbf{d}_k$, $1 \le k \le K$
Output: $\mathbf{s}_k$, $1 \le k \le K$

1.  generate $\mathbf{L}_k$ from $\mathbf{d}_k$, $1 \le k \le K$
2.  generate $\mathbf{U}_k$ from $\mathbf{L}_k$, $1 \le k \le K$ and $\mu_k$
3.  repeat
4.      identify next critical interval $(i,j]$ with $R^c$,
         and list of saturating clients
5.      update $\mathbf{s}_k(t)$, $i < t \le j$, $1 \le k \le K$,
         for all saturating clients
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 $j < q$ and each saturating client $k$
8.      set $L_k(i) = L_k(j)$ for each non-saturating client $k$
9.      remove interval $(i,j]$ from $\mathbf{L}_k$, $\mathbf{U}_k$, $1 \le k \le K$
10. until (total length of critical intervals=$T$)
```

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

$U_k(i) - L_k(i)$, while the corresponding lower bound for non-saturating 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 (Figure 6).

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

The above lemma is proved by Zhao and Tripathi [19]. 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 full buffer at i and end with 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

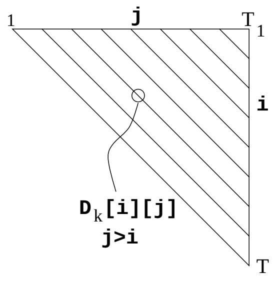

Figure 7: Upper-triangular matrix for efficiently identifying critical intervals in the algorithm for transmissions to clients with limited buffer and unlimited bandwidth. All intervals of equal length $\delta = j - i$ lie along a common diagonal.

are preserved. Since only saturating streams participate in $(i,j]$, the data transfer requirements of non-saturating 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 *lexopt* algorithm has a worst-case time complexity of $O(KT^3)$. Every time an interval is stripped out, recomputing the new minimum bandwidth takes $O(KT^2)$ time, and in the worst case there are $O(T)$ stripping steps. 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 below.

For each stream k, an upper-triangular matrix $\mathcal{D}_k[i][j]$, for $1 \le i < j \le T$, is initialized with the values of $(L_k(j) - U_k(i))/(j-i)$, for $j > i$ (Figure 7). In computing R^c the maximum value in each $\mathcal{D}_k$ needs to be determined. Stripping out interval $(s,t]$ requires deleting entries in $\mathcal{D}_k$ corresponding to the vertical strips $s < j \le t$, $1 \le i < j$ and the horizontal strips $s < i \le t$, $i < j \le T$. 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 $1 \le i \le s$, $t < j \le T$ (Figure 8).

In order to update the maxima of the entries in the matrix after a critical interval is stripped, the matrix is organized as a set of $T-1$ diagonal stripes, with stripe w containing entries $\mathcal{D}_k[i][w+i]$, for $1 \le w \le T-1$. When a critical interval $(s,t]$ is removed, each diagonal stripe may be broken 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

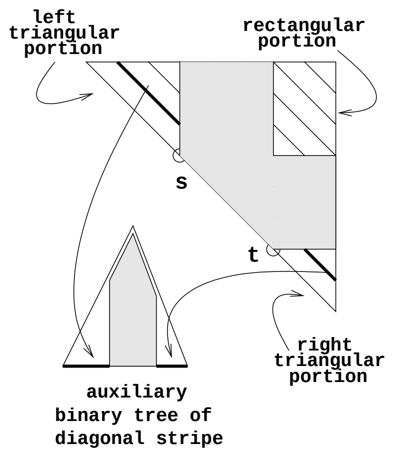

Figure 8: Supplementary binary tree for each diagonal stripe. The highlighted part has become inactive due to the removal of the interval $(s, t]$.

straddle the interval $(s, t]$, and change value as discussed earlier, for saturating and non-saturating streams. In particular, if the current value for an entry $\mathcal{D}_k[i][j]$ in the rectangular region is d, its value after removing the interval $(s, t]$ is given by $d_{\text{new}} = d \times w/(w - \delta) - \Delta/(w - \delta)$, where $w = j - i$ identifies 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 surviving segment left after the removal of several critical intervals.

Each diagonal strip is organized 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 away, 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 traverses to the root from the edges of the invalidated sections, recomputing the new maxima along the paths. 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 (non-invalidated) 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 $O(KT^2 \log T)$.

```
Input: $K$, $M$, $\rho_k$, $\mathbf{d}_k$, $1 \le k \le K$
Output: $\mathbf{s}_k$, $1 \le k \le K$

1. generate $\mathbf{L}_k$ for each $k$, $1 \le k \le K$, and $\mathbf{L}_A$
2. generate $\mathbf{U}_A$ from $\mathbf{L}_A$, $M$
3. prune $\mathbf{L}_k$,$1 \le k \le K$ (from $\rho_k$) and update $\mathbf{L}_A$
4. calculate $\mathbf{s}_A$ from shortest-path algorithm on $\mathbf{L}_A$,$\mathbf{U}_A$
5. update $\mathbf{s}_k$, $1 \le k \le K$ from $\mathbf{s}_A$
```

Figure 9: Algorithm outline for the data access problem with shared buffer and channels with limited bandwidth.

6. Shared Buffer Data Access over Limited Bandwidth Channels

We now consider the general data access problem of accessing data with 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 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, the cumulative upper bound $U_A(i)$ is computed as $U_A(i) = L_A(i-1)+M$. We then prune $L_k(i)$ for each individual stream as described in Section 4.1, and the cumulative lower bound $L_A(i)$ is updated correspondingly.

Subsequently, we apply a smoothing algorithm, similar to that of Lee and Preparata [8, 16], with lower and upper bounds $L_A(i)$ and $U_A(i)$ respectively, in order to identify the sequence of aggregate rates required over time. The total complexity is $O(KT)$, since the construction of the lower and upper bounds require $O(KT)$ time, while running the smoothing algorithm on a single stream (L_A, U_A) takes $O(T)$ time [8, 16]. Our algorithm is outlined in Figure 9.

6.2. Optimality Proof

For purposes of proving the optimality of the schedule generated by the above algorithm, we use an iterative process of identifying critical intervals instead of the functionally equivalent shortest-path algorithm for calculating critical rates [8]. It has been shown in previous work [16] show that the lexicographically optimal schedule for (L, U) is exactly the shortest path in the feasible region.

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

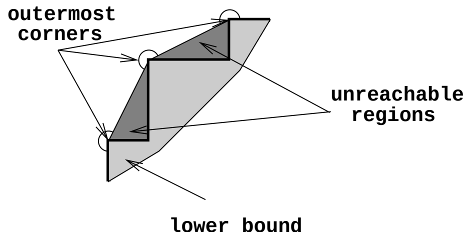

Figure 10: Example of region that is unreachable by channel transmission schedules.

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. By 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 because the critical rate would be higher. Similarly, if there is some i', for $i < i' < j$, where there is buffer overflow, 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 properties:

1. The critical rate is never more than the sum of the bandwidth capacities of the channels, and
2. There is no need for a channel to exceed its bandwidth in order to avoid violating its corresponding 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. Then, as a result of pruning the lower bounds of the individual streams and the upper bound of the aggregate requirements, the critical rate is no more than the sum of the bandwidth capacities of the individual streams. (Note that when we strip off a critical interval, the lower bound of the left endpoint is set equal to its upper bound.)

Claim 2 is a consequence of the fact that, in any interval of unitary length, a channel transmission never has to start from a point located left of the (outermost) corner of its lower bound. An example is shown in Figure 10. From the aggregate transfer schedule $\mathbf{s_A}$, the individual channel transfer schedules $\mathbf{s_k}$ can be derived. 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 for meeting 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: Validity is result of Lemma 3, while optimality is consequence of the shortest-path approach followed for constructing the aggregate schedule. ∎

7. Experimentation and Discussion

For the problem of data transmission to clients with limited buffer space, the ability of *lexopt* smoothing to minimize the utilized bandwidth during the entire aggregate transfer schedule can improve the probability of accepting new playback requests under conditions of limited shared link capacity.

In Figures 11 and 12, we can observe the bandwidth requirements over time when the aggregate transfer schedule of different streams has been smoothed using the *lexopt* and *minmax* methods, respectively. For these experiments, six different MPEG-2 clips with distinct statistical features have been used that were generated with variable bit-rate encoding parameters [1]. When multiplexing over the same link three streams of low bit-rate variability (Figure 11), the difference between the average and the maximum required aggregate bandwidth is relative small, and keeps insignificant the comparative advantage of *lexopt*. However, when multiplexing three streams of high bit-rate variability, the above difference becomes larger (Figure 12). Correspondingly, in the time period after 1000 seconds, *lexopt* keeps the required bandwidth about 20% lower than *minmax*. This is the result of better managing the buffer space available across the different clients in order to avoid coincidence of peak bandwidth requirements across the different streams.

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 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

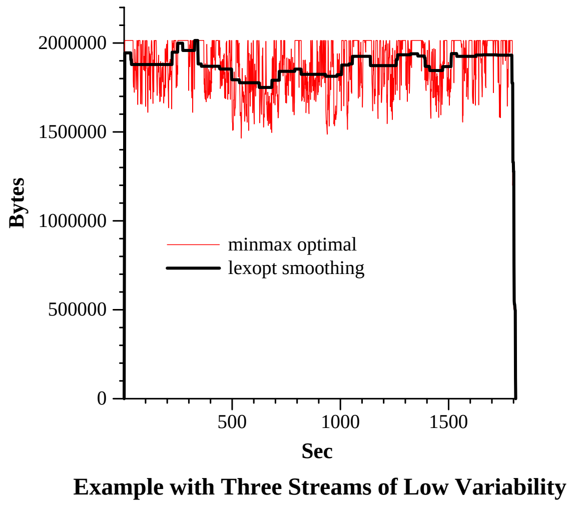

Figure 11: For video streams of low variability, the average bandwidth (bytes/s) required over time using the *minmax* and *lexopt* smoothing methods is very close to the maximum. The methods were applied to three different streams of low variability with buffer space per client of 2MB.

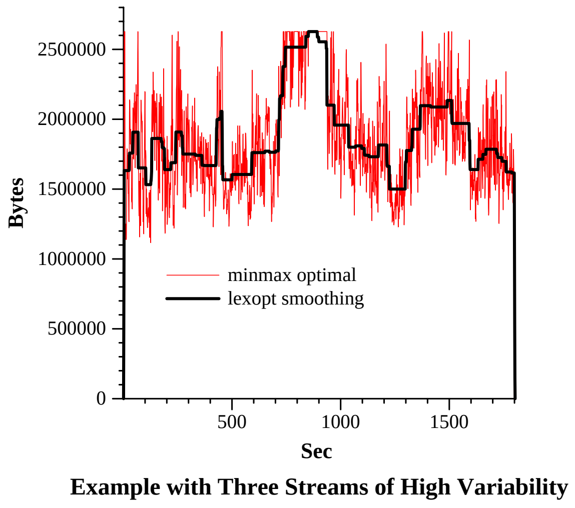

Figure 12: For streams with high variability, the average bandwidth (bytes/s) required over time is significantly lower with the *lexopt* smoothing method than with *minmax*. The methods were applied to three different streams of high variability with buffer space per client of 2MB.

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

In this paper, we address the problem of smoothing multiplexed network traffic for streaming data so as to achieve a lexicographically optimal smoothing (*lexopt* smoothing) of the aggregate bandwidth. We considered *lexopt* smoothing for the complementary problems of data transmission to multiple clients and data access from multiple storage devices (or other downstream servers).

For the data transmission problem, we developed *lexopt* smoothing algorithms for two client models: clients having different buffer capacities but no individual bandwidth constraints, or clients having different bandwidth constraints but no buffer limitations. The general case in which clients have different buffer capacities and bandwidth constraints is open. For the data access problem, we developed a *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 for hierarchical scheduling that results into increased resource utilization and flexibility in handling unknown future traffic. By provably minimizing the variance of the required aggregate bandwidth, maximum resource requirements within the network become more predictable, and useful resource utilization increases. Fairness in sharing a network link by multiple users can be improved, and new requests from future clients are more likely to be successfully admitted without the need for rescheduling previously accepted traffic. Efficient resource management at the network edges can better meet quality of service requirements without restricting the scalability of the system.

Continuing work is directed not only at the scheduling problem but also at other factors affecting total performance. For example, caching is a key technique for filtering (and thereby reducing) the amount of data required from the upstream node(s), and thus caching can significantly reduce even further the amount of control traffic sent upstream. Scheduling flexibility can be traded for reduced amount of control traffic needed to specify the data received by different clients over time, by increasing the length (granularity) of individually requested file segments.

9. Acknowledgments

We would like to thank the anonymous reviewers for suggestions that improved the presentation of the paper.

References

[1] S. V. Anastasiadis, K. C. Sevcik, and M. Stumm. Server-based smoothing of variable bit-rate streams. In *ACM Multimedia Conference*, pages 147–158, Ottawa, ON, Oct. 2001.

[2] W.-C. Feng and J. Rexford. A comparison of bandwidth smoothing techniques for the transmission of prerecorded compressed video. In *IEEE INFOCOM*, pages 58–66, Kobe, Japan, Apr. 1997.

[3] 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*, 12(6):94–107, Nov/Dec 1998.

[4] D. T. Hoang, P. M. Long, and J. S. Vitter. Efficient cost measures for motion compensation at low bit rates. In *IEEE Data Compression Conference*, Snowbird, UT, April 1996. IEEE Computer Society Press.

[5] D. T. Hoang and J. S. Vitter. *Efficient Algorithms for MPEG Video Compression*. John Wiley & Sons, New York, 2002.

[6] T. Ibaraki and N. Katoh. *Resource Allocation Problems: Algorithmic Approaches*. Series in the Foundations of Computing. MIT Press, 1988.

[7] T. V. Lakshman, A. Ortega, and A. R. Reibman. Vbr video: Tradeoffs and potentials. *Proceedings of the IEEE*, 86(5):952–973, May 1998.

[8] D. T. Lee and F. P. Preparata. Euclidean shortest path in the presence of rectilinear barriers. *Networks*, 14:393–410, 1984.

[9] Y. Mansour, B. Patt-Shamir, and O. Lapid. Optimal smoothing schedules for real-time streams. In *ACM Principles of Distributed Computing*, pages 21–29, Portland, OR, July 2000.

[10] J. McManus and K. Ross. A dynamic programming methodology for managing prerecorded vbr sources in packet-switched networks. *Telecommunications Systems*, 9:223–247, 1998.

[11] S. Paek and S.-F. Chang. Video server retrieval scheduling for variable bit rate scalable video. In *IEEE Multimedia Computing and Systems*, pages 108–112, Hiroshima, Japan, June 1996.

[12] A. L. N. Reddy and R. Wijayaratne. Techniques for improving the throughput of vbr streams. In *ACM/SPIE Multimedia Computing and Networking*, pages 216–227, San Jose, CA, Jan. 1999.

[13] A. R. Reibman and A. W. Berger. Traffic descriptors for vbr video teleconferencing over atm networks. *IEEE/ACM Transactions on Networking*, 3(3):329–339, June 1995.

[14] 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. on Multimedia*, 2(1):37–48, Mar. 2000.

[15] S. Sahu, Z.-L. Zhang, J. Kurose, and D. Towsley. On the efficient retrieval of vbr video in a multimedia server. In *IEEE Multimedia Computing and Systems*, pages 46–53, Ottawa, Canada, June 1997.

[16] 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 Transactions on Networking*, 6(4):397–410, Aug. 1998.

[17] I. Stoica, S. Shenker, and H. Zhang. Core-stateless fair queueing: Achieving approximately fair bandwidth allocations in high speed networks. In *ACM SIGCOMM*, pages 118–130, Vancouver, BC, Sept. 1998.

[18] B. Yener, G. Su, and E. Gabber. Smart box architecture: A hybrid solution for ip qos provisioning. *Computer Networks Journal*, to appear.

[19] W. Zhao and S. K. Tripathi. Bandwidth-efficient continuous media streaming through optimal multiplexing. In *ACM SIGMETRICS*, pages 13–22, Atlanta, GA, June 1999.