Resource Utilization Balancing in Disk Storage of VBR Streams

Stergios V. Anastasiadis*, Kenneth C. Sevcik* and Michael Stumm†

*Department of Computer Science
†Department of Electrical and Computer Engineering
University of Toronto
Toronto, ON M5S3G4

November 1999

CSRG-TR 401

Abstract

Economic viability of stored video streaming services is critically dependent on the existence of efficient storage servers that amortize their operating cost over large numbers of concurrent streams. Simplifications in device modeling and resource reservation schemes, or restrictions in the disk transfer scheduling and memory buffer allocation policies can create resource utilization imbalance and reduce considerably the amount of usefully consumed resources. We propose using detailed and accurate resource reservation for each stream, and minimize the disk time variability in a way that guarantees server memory utilization bounded above by the corresponding disk time utilization. We compare, in a consistent experimentation environment, this policy with previously proposed video streaming methods and demonstrate significant increase in the number of successfully supported streams.

1 Introduction

Despite the advances in processing speed, data storage capacity and networking bandwidth, the wide deployment of streaming video over the Internet or in the form of interactive TV services that was foreseen a decade ago, has been slow. The necessary networking infrastructure has only started to fall into place in the last two years. Early video server designs based on expensive supercomputers were more recently replaced by commercial products based on off-the-self workstations with switched interconnects; however, large scale installations of such systems don't exist yet. Even though functional video servers can be built with low cost hardware, further understanding of their native resource needs is required, in order to improve their efficiency and make the supported services viable [10].

In the present work, we focus on the server side of a video distribution system, Fig. 1, and identify as potential bottleneck resources the disk bandwidth, the memory buffer, and the network bandwidth. We are interested in variable bit rate encoding (VBR), where constant quantization factors keep the perception quality constant with a transfer rate that changes as required [18]. Supporting only constant bit rates (CBR) implies reservation and

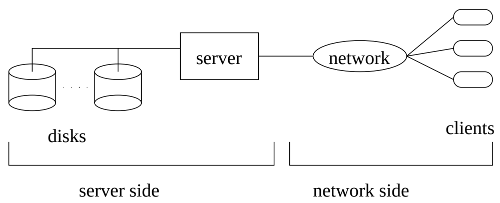


Figure 1: The server and network side of the video distribution system.

consumption of resources at a level sufficiently high to meet the highest requirements anywhere in the stream for a particular perception quality. Using streams with variable bit rates but treating them as constant bit rate streams, can sometimes even increase the amount of resources that need to be reserved or consumed relative to the previous case. Retrieving VBR streams according to their native requirements allows both the amount of resources reserved and actually consumed to decrease.

Although a number of experimental systems and studies refer to one or more different streaming strategies, we are unaware of any comprehensive comparison that evaluates the reserved and consumed resource tradeoffs involved. In fact, scalability and availability issues arising in configurations with multiple nodes and disks create enough design difficulties to limit all the existing disk-array based systems to constant rates. A decision to support VBR streams according to their native requirements has broad implications on the structure of the system, and direct comparison with the other two schemes is crucial in order to justify the extra complexity. In this paper, we identify representative policies for each of the above schemes and compare their efficiency on a system structure general enough to accommodate all of them.

The basic problem in VBR streaming comes from resource consumption variation over time and the imbalance that this potentially creates. In single disk systems, there is variation in memory and disk bandwidth requirements over time, while in multiple disk systems, the problem manifests itself as load variation among different disks as well. Prefetching techniques may alleviate the problem; they can on the other hand lead to excessive memory requirements in the server. Even though methods for minimizing buffer requirements are known [3, 17], they have not been tried in systems with variable resource reservations, where they probably have the largest potential. Therefore, the rest of our evaluation is focused on applying prefetching techniques and quantifying the benefit that they add.

In Section 2, we specify the scope of the problem. In Section 3, we describe previous literature related to this study. In Section 4, we identify representative streaming policies that previously were proposed. In Section 5, the structure of the prototype implementation is presented. In Section 6, the experimentation platform is described, along with the workload used. In Section 7, efficiency tradeoffs of the streaming policies are compared. In Section 8, we consider ways of increasing the number of concurrently supported streams. In Section 9, the efficiency tradeoffs of the alternative prefetching techniques are studied. In Section 10, we show how simulated disk and real disk measurements compare using the same system prototype. Section 11 presents conclusions and indicates our plans for on-going work.

2 Problem Specification

Most existing systems that support streaming video use a round-based operation, which facilitates the bookkeeping of the resource requirements of each stream across time. The choice of the round duration is usually a tradeoff involving responsiveness, buffer space and disk access efficiency. The longer the round duration, the less responsiveness, the larger the buffer space required, and the higher the disk efficiency (since a higher fraction of disk time is spent on useful data transfer).

Usually, it is assumed that the resource requirements of stored streams are known a priori. This assumption breaks down in the case that VCR operations (such as pause, rewind and fast-forward) are allowed, unless each jump to a new position is handled as a new request. We presume that normal playback is the usual mode of operation, and we strive to optimize the system operation for this common case. We represent a stream k using the *stream size* vector $\mathbf{s}_k$ of length l_k, where $s_k(i)$, $1 \leq i \leq l_k$, is the number of bytes required to encode all frames of video in the ith round of playback.

In the path from the server disks to the client framebuffer (see Fig. 1), we identify alternating *active* and *passive* resource types. A passive resource mainly accepts data for storage, and its capacity is measured in bytes. A *data schedule* is defined for each stream, that keeps track of the amount of data that must be stored into the passive resource during each round; the schedule is used for resource reservation and admission control. An active resource transfers data between different storage devices and its capacity is measured in bytes/sec. For each stream, there is a *data schedule* that specifies the number of bytes that have to be transferred through the active resource during each round; this schedule is used for request scheduling. The corresponding *transfer schedule* keeps track of transfer time, and is used for time reservation of the active resource.

We consider two active resources, namely disk and network bandwidth, and a passive one, namely the memory buffer in the server. The basic parameters of the disk are the minimum sustained internal transfer rate R_{disk}[1], the full-stroke seek time $T_{fullSeek}$, the track-to-track seek time $T_{trackSeek}$, and the average rotation latency T_{lat}. The network bandwidth is characterized by its transmission rate R_{net}, and the buffer by its storage capacity M_{buf}. We define a *stream schedule* as a triple $(L_k, \mathbf{d}_k^d, \mathbf{n}_k^d)$. The non-negative integer L_k indicates the number of transfer rounds in the stream, the positive integer vector $\mathbf{d}_k^d$ defines the *disk data schedule*, while the positive integer vector $\mathbf{n}_k^d$ is the *network data schedule*.

[1]Sustained rate is the transfer rate from the disk platter to the on-disk cache, when accessing large amounts of data spanning many tracks [16].

The *disk transfer schedule* $\mathbf{d}_k^t$ can be derived from the *disk data schedule* using the expression $d_k^t(i) = T_{dsk}(d_k^d(i)) = \frac{d_k^d(i)}{R_{disk}} + m \times (T_{trackSeek} + T_{lat})$, where m is the maximum number of disk arm movements for the stream disk requests during a round. Similarly, the *network transfer schedule* $\mathbf{n}_k^t$ is given by $n_k^t(i) = T_{net}(n_k^d(i)) = \frac{n_k^d(i)}{R_{net}}$. The *buffer data schedule* $\mathbf{b}_k$ is given by the expression $b_k(i) = \sum_{j \le i} d_k^d(i) - \sum_{j<i} n_k^d(i)$. The first sum is the amount of data retrieved from the disk through round i, while the second sum is the amount of data sent through the network by the start of round i. The *disk utilization* is the function $U^d(x) = \frac{T_{dsk}(x)}{T}$ which is the fraction of round duration T occupied by a disk request of size x. Similarly, let *buffer utilization* be the function $U_b(x) = \frac{x}{M_{buf}}$ which is the fraction of the buffer occupied by x bytes.

The buildup latency of stream k, $l_k^{bld} = L_k - l_k$ is the number of rounds from the first disk transfer to the first network transfer. In existing systems, the l_k^{bld} is a small number of rounds, since there are usually additional rounds of delay prior to the first disk transfer and also after the first network transfer and before decoding starts. By increasing l_k^{bld}, it is possible to reduce the maximum disk rate at the cost of increasing the memory requirements on the server side. More importantly it decreases system responsiveness by causing the user to have to wait longer for the video to start. In the scheduling policies considered in the present study, we restrict the latency to be only a few rounds. The network data schedule ensures that the buffer size on the client side is not exceeded. Although making use of the client buffer could reduce the maximum memory and network bandwidth requirements on the server side, ongoing research in networking makes it unclear what network schedule maximizes system efficiency. A possible (rather stringent) constraint is to assume that the network data schedule $\mathbf{n}_k^d$ is equal to the stream vector $\mathbf{s}_k$.

In existing production systems, bandwidth capacity limitations of individual disks make it desirable to stripe data across multiple devices so that concurrent user requests for different parts of the same stream can be serviced by different disks. In most existing video servers based on disk arrays, only one disk or disk cluster in case of striping across clusters, is accessed for each stream during a round [5, 13]. By initiating different playback requests on different rounds modulo the number of array disks it is possible to achieve disk load balancing. This also means that the amount of data retrieved for a stream from a disk, is the amount of data originally stored on that disk for this particular round. Although in the present study, we assume that streams are fully stored on a single disk, we keep the restriction of deciding the retrieval schedule before recording. Thus we make sure that our results are easily applicable to disk striping, as well.

As metric of performance we use the number of streams N that can be concurrently active. This number is controlled by the admission control process, which considers the available resources. We consider the network schedule fixed and assume that only memory buffer and disk bandwidth are critical resources. Any disk data schedule is valid provided that it does not exceed the available resource amounts and brings data into buffer in time to be sent over the network. We are looking for the valid disk data schedule $\mathbf{d}_k^d$ that is decided during stream storage and maximizes the number of accepted requests in the system during retrieval, assuming that future requests are not known in advance. More formally,

find d_k^d that maximizes N, such that for each round i and active stream k dispatched at i_k^{dsp}:

Stream Constraints		System Constraints	
(i)	$\sum_{j=0}^{i-i_k^{dsp}} d_k^d(j) \geq \sum_{j=0}^{i-i_k^{dsp}-l_k^{bld}+1} s_k(j+1)$	(i)	$\sum_{k=1}^{N} d_k^t(i-i_k^{dsp}) + 2T_{fullSeek} \leq T$
(ii)	$\sum_{j=0}^{i-i_k^{dsp}} n_k^d(j) \geq \sum_{j=0}^{i-i_k^{dsp}-l_k^{bld}} s_k(j+1)$	(ii)	$\sum_{k=1}^{N} n_k^t(i-i_k^{dsp}) \leq T$
		(iii)	$\sum_{k=1}^{N} b_k(i-i_k^{dsp}) \leq M_{buf}$

Stream constraints (i) and (ii) ensure that disk and network transfer occur in time. System constraints (i), (ii), (iii) guarantee that the available disk bandwidth, network bandwidth and buffer space capacities are not exceeded, respectively.

3 Related Work

The number of concurrent streams accessed from a disk is limited to the order of tens, so the statistical benefit from multiplexing might not be as high as in the case of high capacity links. In addition, the modeling requirements of devices with electromechanical parts, such as hard disks, are different and more complex than network interface cards and links. Thus, deterministic methods based on accurate modeling or measurement of the available resources and stream requirements are most frequently used for admission control in disks [6].

Many of the existing streaming systems can support only streams with constant bit rates, as described by Bolosky et al. [5]. Alternatively, it is possible to support VBR streams by using constant bandwidth and some initial buildup time, similarly to what is described for network transfers by Sen et al. [7, 22]. Another streaming method permits the bandwidth to switch between zero and some other fixed value [2]. In the Fellini system proposed by Martin et al. [13], the server buffer size is fixed to some maximum value for each stream and the disk bandwidth varies as required to keep the buffer full. Finally, detailed bookkeeping of the required buffer and disk bandwidth is possible on a per-round basis for each stream, as in the Continuous Media Server by Makaroff et al. [6, 11].

In considering the network side, it was shown recently by Salehi at al. [21] that increased network utilization can be achieved by minimizing the peak rate and variance across different frame times using prefetching at the client with a fixed buffer constraint. Alternative methods, described by Feng and Rexford [8], attempt to minimize the number of transfer bandwidth increases or decreases in order to reduce bandwidth renegotiation overhead and network capacity overloads. When multiplexing streams on a shared transfer medium, a greedy earliest deadline first based algorithm was shown to minimize the peak bandwidth requirement by Zhao and Tripathi [23]. A dynamic programming-based approach was shown by McManus and Ross [14] to accommodate several optimization objectives such as minimizing the peak rate, or the number of rate changes with fixed client buffer

	Reserved		Utilized	
Policies	Disk Bandwidth	Buffer	Disk Bandwidth	Buffer
Maximum Bandwidth	Fixed	Fixed	Fixed	Fixed
Constant Bandwidth	Fixed	Fixed	Fixed	Variable
Switched Bandwidth	Fixed	Fixed	Variable	Variable
Fixed Buffer	Fixed	Fixed	Variable	Fixed
Raw Bandwidth	Variable	Variable	Variable	Variable

Table 1: The policies considered and their reserved or utilized resource variability.

constraints.

For the server side, Biersack and Hamdi describe a method to estimate the disk transfer rate that minimizes the total cost of the server when disk transfers of fixed size take place as required to prevent client starvation [3]. Paek and Chang [17] estimate the fixed disk bandwidth reservation that should be made for each stream in order to maximize the number of streams that can be started at the same time. These techniques assume that one or more of the resources use fixed size reservation for the duration of each stream. When more detailed and accurate characterization of disk bandwidth and memory buffer is assumed, it remains unclear how to allocate resources among different streams in a way that does not underutilize disk bandwidth or overallocate the available memory buffer space, as it is pointed out by Makaroff et al. [11].

In summary, all proposed solutions on the network side of the system, assume a fixed client buffer size. Additional constraints on the server side include the increased complexity of the disk bandwidth modeling and the shared nature of the server buffer space. Previous techniques for video servers usually make simplifying assumptions about resource reservations. Our goal is to efficiently allocate disk bandwidth and buffer space among competing VBR streams using ideas from traditional application-driven prefetching [19]. In particular, we describe a greedy algorithm for generating a disk data schedule that is no lower with respect to utilization than the server buffer schedule and thus balances the usage of the two resources. We compare it with other "optimal" schemes and demonstrate benefits in the amount of resources consumed and the number of accepted requests.

4 Basic Resource Scheduling Policies

In this section, we describe five representative scheduling policies that will be compared with respect to their resource requirements. In all cases, the disk data schedule, before used, has to be quantized with respect to the disk transfer unit and the buffer schedule has to be adjusted accordingly. (See appendix A for pseudo-code description.)

1. We call as *Maximum Bandwidth* the policy of only supporting streams with constant bit rates [4, 5]. The amount of data retrieved from disk and sent over the network is kept fixed within a round for each stream (

although it might be different for different streams). The amount of memory buffer reserved for each stream is also fixed at twice the amount of data retrieved from disk during a round. The reserved disk time and buffer space are fixed, too. The Maximum Bandwidth policy allows the structure of the system to be kept simple, but the resource utilization remains high even for parts of the video stream with low information content [18].

2. The rest of the schemes we consider are based on variable bit rate MPEG encoding. In the *Raw Bandwidth* policy, we allow both the disk transfer size and the buffer space to vary according to the requirements of the decoder. The disk data schedule is the same as the stream vector $\mathbf{s}$ (stream index k is dropped from now on for simplicity) but shifted one time step earlier. The network data schedule is same as the stream vector $\mathbf{s}$. The reservation of buffer and disk bandwidth is done on a round by round basis [6]. A version of this scheme was implemented in the UBC Continuous Media Server [11].

3. The policy *Constant Bandwidth* keeps the disk request size fixed during a playback session. For our study, we choose the disk request size to be the minimum required to avoid client starvation. The reserved disk time and buffer space are chosen fixed and large enough to satisfy the maximum requirements of the stream. To avoid excessive buffer requirements with this policy, we allow an initial latency of up to a maximum value (specified in Section 7). Up to a point, the larger the allowed latency, the smaller the maximum buffer required for this policy [22].

4. It was reported by some researchers, that strictly constant rate can lead to excessive buffer space requirements [3, 7, 17]. In the *Switched Bandwidth* policy, a slightly different approach is to keep the disk transfer size fixed, but to initiate transfers only when necessary. Although the original proposals allowed an initial build-up period up to the length of the stream, we assume initial latency of one round. Thus we minimize the buffer requirements and can evaluate the relative benefit to the previous policy. We choose the minimum disk transfer rate that guarantees no starvation at the client. As before, the reserved disk time and memory buffer are set to the values $\max_i d^t(i)$ and $\max_i b(i)$ respectively.

5. In the *Fixed Buffer* policy, we set the server buffer space of each stream to a value twice the maximum network transfer size. The disk transfer size does not exceed the maximum network transfer, and is large enough to fill up but not overflow the buffer space. This policy was implemented in the Fellini video server from Lucent [13], and both the reserved disk bandwidth and buffer were fixed to their maximum values for each stream.

Table 1 shows how the reserved and consumed resources are handled under the different policies described above. Although, the previous policies have been studied individually in previous publications through analytical arguments, simulation experiments, or actual prototypes, this study compares the different policies using a common prototype platform.

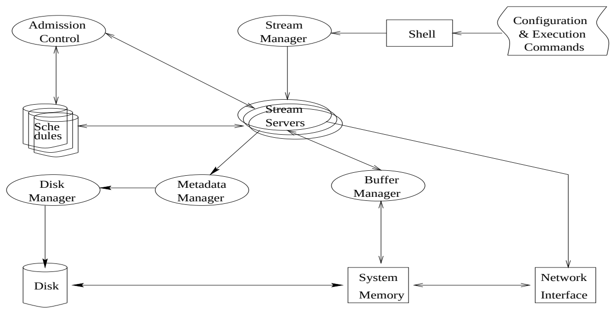


Figure 2: System architecture overview.

5 Experimentation Environment

We have designed and built an experimentation platform, that has allowed us to evaluate the resource requirements of the different policies, and investigate possibilities for improving the efficiency of the existing policies. Since the basic difference among the alternative policies lies in the resource schedules, the basic resource management functionality is kept the same for each policy. Algorithms that implement the policies produce schedules that are passed as parameters to the playback command.

All the modules are implemented in C++ using Pthreads on AIX4.1, in about 10K lines of code. The DiskSim disk simulation package from the University of Michigan can be used for detailed disk measurements [9], and also the MPEG-2 decoder from the MPEG Software Simulation Group for frame size identification [1] (the MPEG-2 decoder functionality was not used for the experiments in this study since we had the frame size traces already). The bottom layer of the Disk Manager can communicate either with the DiskSim disk simulation package, or with a real disk through the character device interface. The metadata are stored in the unix file system as regular files (similar to Fellini), and during operation are kept in main memory similar to Microsoft's Tiger system.

Systems Modules

We now describe the modules of our prototype shown in Figure 2. At the bottom layer lies the disk device driver (or the disk simulator). Directly above is the *Disk Manager* which receives requests to transfer consecutive blocks of each stream between the disk and system memory. Requests are reordered according to their starting physical disk block number, according to Circular Scan disk scheduling. In order to keep disk transfers efficient and predictable, the disk space is allocated independently of the disk accesses [15]. Thus, during recording, disk space is allocated in multiples of consecutive tracks, such that a maximum request will not span more than two different extents of consecutive tracks. Thus, for each request, we have to pay at most two disk rotation latencies and two seeks, which is the minimum possible without storing each file contiguously. The function $T_{dsk}()$ of the stream disk transfer schedule therefore becomes $T_{dsk}(x) = \frac{x}{R_{disk}} + 2 \times (T_{trackSeek} + T_{lat})$, where x is number of bytes transferred for the stream in one round (we set $m = 2$ to the formula of Section 2).

The *Metadata Manager* is located above the Disk Manager. It manages file indexing information, and it performs logical to physical block number translations. Each disk transfer request arriving from the higher layers is broken up into one or more requests of multi-track size, which are passed down to the Disk Manager along with completion synchronization information. In the current implementation, the Metadata Manager can address only an individual disk device.

The *Buffer Manager* allocates memory in groups of fixed size buffers (4KB each) consecutive in virtual memory. Thus, it avoids dynamically acquiring a single chunk of memory for each request. We use a bitmap for space allocation, and counters to keep track of the number of references. Partial space deallocation is permitted which allows the disk data schedule to be different from the network schedule. With measurements using the character device interface of a SCSI disk, we found a bandwidth degradation in the order of $25 - 30\%$ when transferring into non-contiguous buffers. We avoid that by allowing the interface of the buffer allocator to handle groups of consecutive buffers.

The *Stream Server* manages stream playback. Each stream playback is serviced by two threads, *reader* and *sender*, in a producer-consumer relationship. For each round, a timer signals all the reader threads to submit read requests through the metadata manager to the disk manager according to the disk data schedule. In addition, each reader thread passes a record with the newly allocated buffer space specification into a fifo queue that is read by the send thread. The sender thread uses the locking/signaling mechanism attached to the individual buffers to detect the completion of the disk transfer. It then submits the amount of data specified by the network schedule into the network interface. The *Stream Manager* is the interface between the user and the rest of the systems. It is mainly responsible for starting stream server threads.

In the *Admission Control* module, we use two circular vectors D and B of length L_{ac}, with $L_{ac} \geq L_k$ for stream k, to represent the currently allocated disk time and buffer space respectively. The disk time vector D is initialized to twice the full-stroke seek time, $D[i] = 2T_{fullSeek}$, and the buffer space is set to zero, $B[i] = 0,\ i = 0, \ldots, L-1$. When a new stream request arrives, the current disk time and buffer allocations are checked to see if sufficient resources are left to accommodate the stream disk transfer and buffer data schedules. (It is assumed that the available network bandwidth is enough to guarantee no network resource bottleneck.) If at round r, a new request arrives for a stream of buffer data schedule $\mathbf{b}_k$, disk transfer schedule $\mathbf{d}^t_k$ and length L_k, then the new stream is accepted for round $i^{dsp}_k = r + 1$, if and only if:

$$for\ i = i^{dsp}_k, \ldots, i^{dsp}_k + L - 1\ (\ mod\ L_{ac}\ )$$

$$D[i] + d^t_k[i] < u_{disk} \times T_{round}$$

$$B[i] + b_k[i] < u_{buffer} \times M_{buf}$$

The parameter u_{disk} has been chosen equal to 85% leaving some margin for disk time modeling discrepancies due to SCSI channel contention, command overhead, or rotation latency that exceeds the half rotation average assumed per transfer. The parameter u_{buffer} has been chosen equal to 85% in order to account for fragmentation problems arising in the buffer manager. In practice these factors seem to prevent transient overloads of the system. If the resource availability test is passed, the disk transfer and buffer data schedules are added to the corresponding vector of the module, and the stream is dispatched for playback.

6 Experimentation Setup

We used twelve MPEG-1 stream frame size traces from the University of Wuerzburg [20] [2]. They all contain 40,000 frames except `news_1`, which is 31,515 frames long, captured at 25 frames per second, and compressed using the MPEG pattern IPBBPBBPBBPBB. That is, each clip runs for 26.67 minutes, except `news_1` which runs for 21.01 minutes. Characteristics of the clips are given in Table 2. The experiments were done on a cluster of IBM AIX 4.1 RS/6000 nodes with PowerPC 604 processors at 133MHz and 96 MB main memory. All the machines are interconnected with both 100Mbps switched Ethernet and 155Mbps ATM. Some of systems are used as clients, one per server, through one of the two networks. The amount of memory reserved for buffering was 64MB, although this caused excessive paging activity with some of the policies, when doing real disk and network transfers [3]. To avoid this problem, we used the DiskSim simulated version of the HPC3323A SCSI disk, with the features shown in Table 3. Thus we could get disk access times from the simulator without any of the buffer pages being written. The simulation execution time for each disk operation was less than half the simulated disk time, so an extra delay was added to each disk transfer. In Section 10, we compare the simulated disk numbers against real disk measurements using the same platform, but with only 32MB reserved for the buffer.

Each experiment was about half an hour playback of the maximum number of streams accepted when successive streams were submitted every 5 secs. The choice of the interarrival time was intended to be long enough to allow some realistic multiplexing, but also short enough to achieve full loading of the system. The experiments were completed successfully with no reported constraint violations. In our homogeneous experiments, a single stream is submitted repeatedly. In the heterogeneous case we used groups of streams instead. The groups include `movie` (`bond, dino, lambs, starwars`), `sport` (`mtv_1, mtv_2, soccer_1, soccer_2`), `talk` (`talk_1, talk_2, news_1, news_2`) and `mixed` (`bond, news_2, mtv_2`). The last group uses the most resource demanding stream of each of the other three groups. Submission in the heterogeneous experiments is done in round-robin form from within each group.

[2] The data sets can be found at ftp-info3.informatik.uni-wuerzburg.de /pub/MPEG/

[3] Recall that the buffers are all located in virtual memory

clip name	short name	avg bytes (per round)	max bytes (per round)	CoV (per round)
		movie		
James Bond: Goldfinger	bond	75,873	228,393	0.368103
Jurassic Park	dino	40,819	137,586	0.389989
The Silence of the Lambs	lambs	22,821	105,136	0.596946
Star Wars	starwars	29,068	113,990	0.559454
		sport		
MTV Video Clips	mtv_1	76,793	307,394	0.434986
MTV Video Clips	mtv_2	61,738	326,931	0.665479
World Cup Final 1994	soccer_1	84,673	230,010	0.366496
World Cup Final 1994	soccer_2	78,373	281,625	0.468220
		talk		
German TV News	news_1	64,505	233,021	0.433177
German TV News	news_2	47,934	293,536	0.458830
German Talk Show	talk_1	45,371	124,575	0.319957
German Talk Show	talk_2	55,915	171,641	0.268355

Table 2: The video clips used with the average, the maximum bytes required during a round and the coefficient of variation.

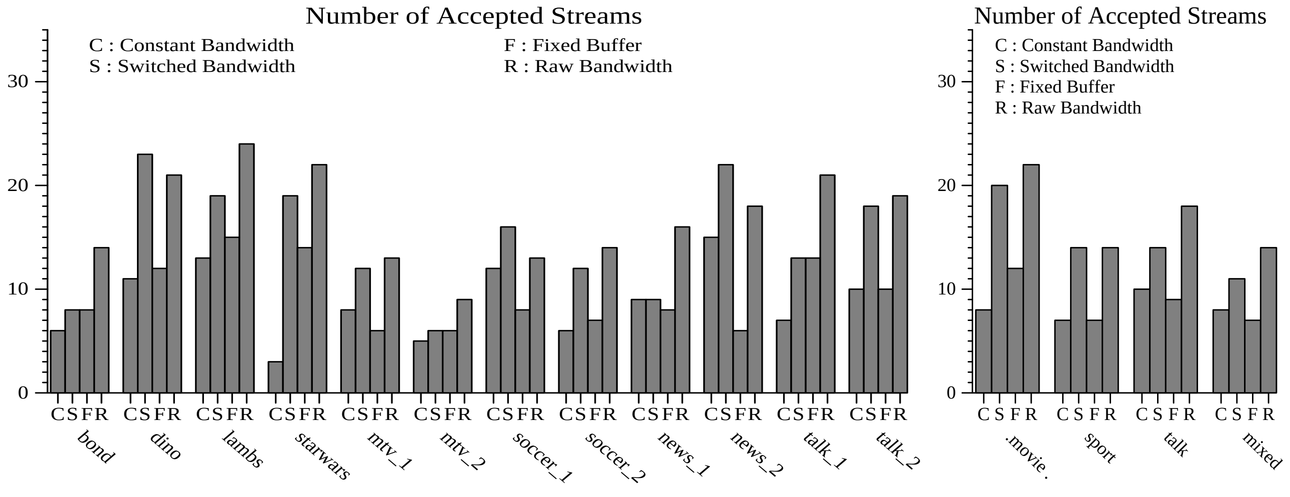
HP C3323			
Data Bytes per Drive	1,052,491,776	Head Switch Time	$<$ 1 msec
Data Sectors per Track	72 to 120	Maximum Seek	22 msec
Data Cylinders	2910	Rotational Latency	5.56 msec +/- 0.5%
Data Surfaces	7	Command overhead Time	$<$ 0.5 msec
Zones	8	Internal Transfer Rate	
Buffer Size	512 Kbytes	Inner to Outer Zone Burst	4.0 to 6.6 Mbytes
Track to Track Seek	$<$ 2.5 msec	Inner to Outer Zone Sust.	2.8 to 4.7 Mbytes

Table 3: Features of the SCSI disk simulation model used in the experiments.

Basic Parameters

In general, contiguous disk space allocation cannot be guaranteed. When used, it means that during retrieval of a single stream the measured disk bandwidth is that of the SCSI channel 10MB/sec, thus hiding both the internal transfer rate 2.8 to 4.7 MB/sec sustained and the seek/latency overhead. So, rather than using contiguous disk space allocation, we assume the stream blocks are distributed round-robin across all the zones of the disk. The disk space is allocated in chunks of multiple contiguous tracks, the number of which is the minimum that gives chunk size above 512KB.

We decided to set round duration $T = 1$ sec for our experiments, in order to keep consistent with previous studies. Here, we analyze the impact of this choice in the estimated number of accepted streams. This analysis refers to the choice of T only, and not the admission control process which considers the requirements of each stream in each round. The streams that we used have maximum average display rate $R_{disp} = 84,673$ bytes/sec. The HPC3323 specification reports minimum sustained internal transfer rate R_{disk} = 2.8MB/sec, which is 70% of the burst inner disk transfer rate $R_{burst} = 4MB/sec$. The full-stroke seek time is $T_{fullSeek}$ = 22 msecs, the average rotation latency T_{lat} = 5.5 msecs, and the track-to-track seek time $T_{trackSeek}$ = 2.5 msecs. The disk allocation scheme guarantees two accesses at most for each stream per round. Assuming B bytes transferred for each stream



(a) Homogeneous (b) Heterogeneous

Figure 3: Number of streams successfully accepted, when submitting every 5 secs (a) the same stream (b) streams of each category round-robin. Stream schedules were generated using the basic policies. The DiskSim HPC3323 disk model has been used and 64MB of memory buffer.

per round, for round time T, and number of streams N, we have: $2T_{fullSeek} + N(B/R_{disk} + 2T_{lat} + 2T_{trackSeek}) \leq T$ and since the display has to last for one round $T = B/R_{disp}$. Thus we get:

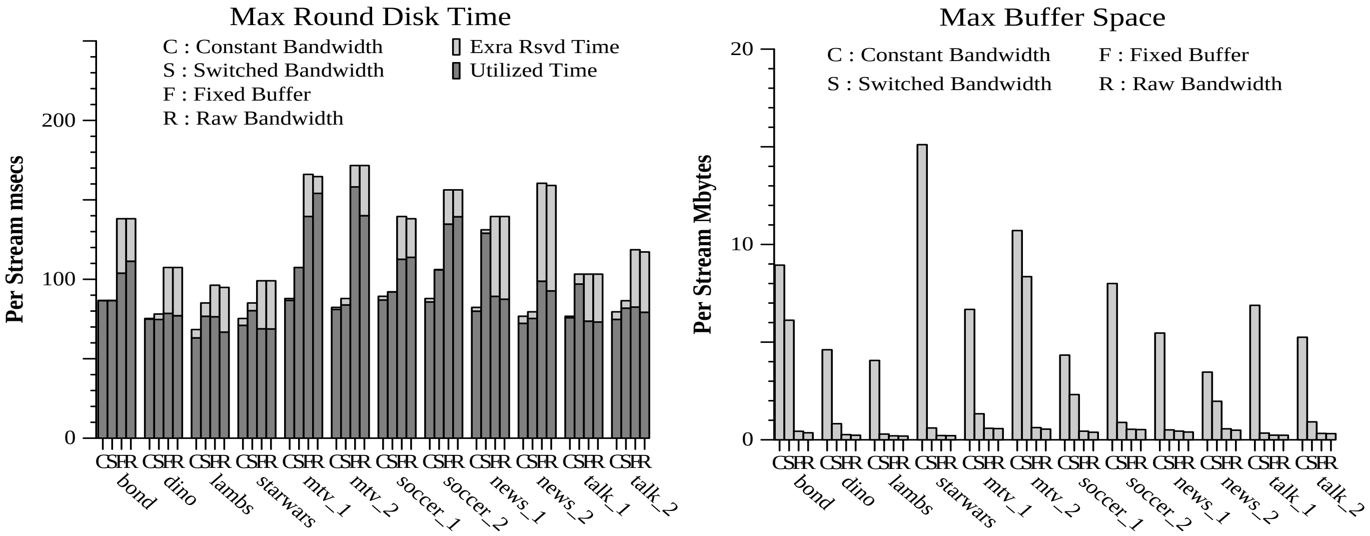
$$N \leq \frac{T - 2T_{fullSeek}}{TR_{disp}/R_{disk} + 2T_{lat} + 2T_{trackSeek}}$$

Initial latency of multiple rounds in disk-array systems usually leads to choosing the value $T = 1$ sec in order to keep waiting time small. With the above numbers the choice $T = 1$ corresponds to 21 streams, that is 60% of the maximum 35 streams ($T > 10$).

7 Efficiency Tradeoffs in the Basic Policies

In this section, we compare the number of accepted streams and the resource requirements of the policies presented in Section 4. For our experiments we used the frame size traces of the video stream. Thus, for policy Maximum Bandwidth we used a fixed disk transfer size equal to the maximum required in a round by the corresponding stream. Also, in the original version of the Fixed Buffer policy, the first transfer fills up the buffer, which creates a disk transfer peak that might exceed twice the maximum network transfer size. Therefore, in our platform, an extra round is introduced at the beginning of this policy schedule, and we divide up the initial peak transfer into two rounds. This allows a more fair comparison with the rest of the policies. Finally, for policy Constant Bandwidth we used linear search to find an initial build-up latency of up to 30 rounds, that minimizes its buffer requirements.

Figure 3 shows the number of accepted streams for the policies Fixed Buffer (F), Constant Bandwidth (C), Switched Bandwidth (S), and Raw Bandwidth (R). We don't report numbers for the Maximum Bandwidth (M)



(a) Disk Time

(b) Buffer Space

Figure 4: The maximum disk time (reserved or utilized) and buffer space measured with only one stream active in the system. The DiskSim HPC3323 disk model is used and 64MB of memory buffer.

policy because they are always the same as those for the Fixed Buffer scheme. Figure 4 compares the maximum disk time and buffer space required in a round when only one stream is active in the system. For disk time, both the actually utilized and the extra reserved fraction of the round duration are shown. In policies C and S, the reserved time is very close to the utilized one. The fixed transfer rate makes it highly probable that the worst case assumption of the reservation will happen. In contrast, policies F and R with actual disk transfer varying between different rounds, have a higher discrepancy between maximum utilized and reserved times. Both the on-disk cache and the bandwidth variability due to zoning make it possible for the worst case "maximum transfer size" from the "inner track" with "on-disk cache miss" events not to occur simultaneously.

In Figure 5, we can see the average measured seek, rotation and transfer time in a round per stream. The average disk response time is comparable among the different policies, although it varies considerably across the different streams. The transfer time, which includes all the delays not covered by seek and latency, is about 70% of the total disk time utilized by each stream. This is relatively high when compared with traditional applications, and is attributed to the large transfer sizes that characterize video streams. Finally, the time spent on latency is close to that spent on seeks.

Note that in Figure 3(a), the number of accepted streams with policy Raw Bandwidth can be up to a factor of three higher than Fixed Buffer (`news_2`). Interestingly, the maximum disk access time reservations of the individual streams in both these policies correspond to the same maximum transfer size (Figure 4). The difference is that policy Fixed Buffer reserves this time for the whole duration of the stream, while Raw Bandwidth reserves it only for the rounds where this transfer actually happens. It turns out that a significant increase in the number of

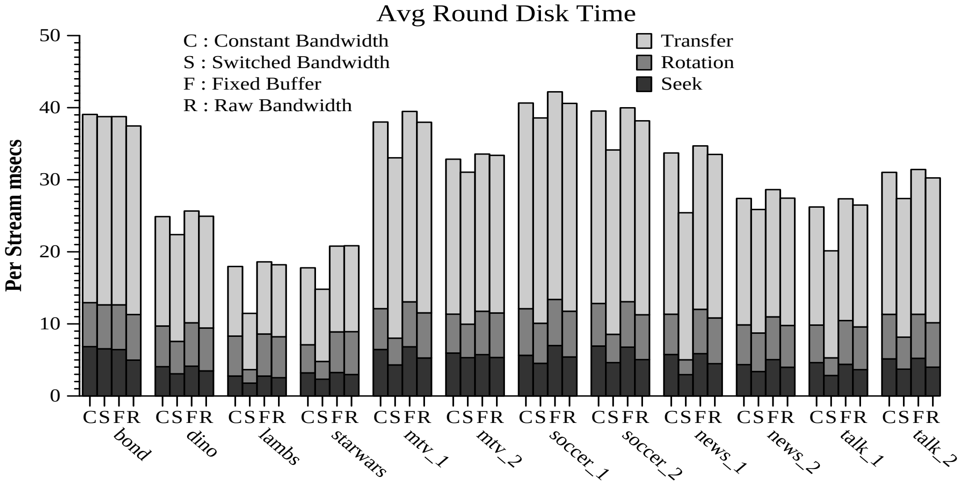


Figure 5: Average round disk time per accepted stream among different policies and streams types broken down into seek, rotation and transfer. The disk time in a fully loaded system is divided by the number of accepted streams. The DiskSim HPC3323 disk model is used and 64MB of memory buffer.

accepted streams can be achieved by removing the constant resource reservation in variable bit rate streams, and allowing multiplexing to take effect on the shared resources. Comparing the above policies, we see that both have the disk bandwidth as the bottleneck resource.

The difference between Constant Bandwidth and Switched Bandwidth has to do with the fixed rate used continuously with Constant Bandwidth but only intermittently with Switched Bandwidth. This affects both the maximum access time and the maximum buffer space required. As expected, the Constant Bandwidth policy requires an order of magnitude larger maximum buffer when compared to the other three policies. Although Switched Bandwidth alleviates the excessive buffer consumption in many cases, it fails to do so in others (`bond, mtv_2, soccer_1, news_2`). For streams `dino, soccer_1` and `news_2`, Switched Bandwidth outperforms the other policies, because it benefits from lower maximum utilization in both buffer space and access time, and this overcomes the negative effect of fixed resource reservation.

In fact, Swiched Bandwidth has less variability in transfer sizes and thus requires a smaller maximum access time than Fixed Buffer or Raw Bandwidth. However, in streams `news_1` and `talk_1`, both Switched and Raw Bandwidth have the same or similar maximum access time. In these cases, it seems that the Raw Bandwidth policy outperforms the Switched Bandwidth by almost a factor of two in the number of streams accepted. The fact that buffer utilization is very low provides strong evidence that fixed disk time reservation can lead to reduced number of accepted streams. Figure 3(b) depicts the number of accepted streams in an heterogeneous workload. Similarly to the homogeneous case, Fixed Buffer and Constant Bandwidth have poor performance relative to Raw Bandwidth, accepting only half the number of streams. Switched Bandwidth performs the same as Raw Bandwidth in `sport` clips, and has 10-25% lower throughput in the other three stream groups.

The general conclusion we draw from the results is that reducing variability in the utilized disk bandwidth increases the number of streams that can be accepted, unless there is a side-effect of excessive buffer buildup. Fixing the amount of resources reserved for the duration of a stream playback can prohibit the multiplexing benefit of the resource sharing and reduce the system utilization. Among the policies studied in this section, policies Switched and Raw Bandwidth demonstrate the best performance across the range of streams used in the experiments.

8 Balancing the Utilization of Resources

The previous section showed that in most cases the maximum memory utilization of the Switched Bandwidth and Raw Bandwidth policies is an order of magnitude lower than that of the disk time utilization (Fig. 4). One reason for this is the particular system configuration used due the current prices. In addition, memory is easier to administer. An interesting question is whether it would be helpful to modify the resource scheduling algorithm in a way that explicitly takes into account the available resources in each upcoming round.

Network Smoothing

The general problem of maximizing the number of active VBR streams in a shared medium has been previously studied in the data networking literature. One basic difference there was the fixed client buffer size dedicated to each individual stream, and the assumption that the communication link bandwidth is the only shared resource. On the server side, the shared resources that are potential performance bottlenecks also include the buffer space and the disk bandwidth. However, assuming that the disk bandwidth is treated as network bandwidth and the server memory as client buffer, it is possible to study one representative such algorithm, the Optimal Smoothing algorithm of Salehi et al. [21].

That algorithm was shown to produce majorization-minimal transfer schedules that minimize peak value and variance of the transfer rate under the constraints of avoiding starvation or overflow of the decoding client. A key parameter is the size of client buffer that should be assumed [4]. In the original publication [21], the larger sizes were considered better due to the increased smoothing effect that they achieved. Of course, on the server side, increased buffer for a few streams prohibits others from being admitted to the system. For our experiments, we applied the above algorithm assuming 4MB buffer size for each stream, including the double buffering required in disk/network transfers; this corresponds to 2MB client buffer of the original algorithm.

[4] For $x, y \in \mathbb{R}^n$, $x \prec y$ if $\sum_{i=1}^{k} x_{[i]} \leq \sum_{i=1}^{k} y_{[i]}$, $k = 1, \ldots, n-1$ and $\sum_{i=1}^{n} x_{[i]} = \sum_{i=1}^{n} y_{[i]}$. When $x \prec y$, x is said to be *majorized* by y. We call x majorization-minimal when there is no other valid vector z such that $z \prec x$.

Balanced Prefetching

As was pointed out, the shared nature of the buffer space adds an extra complication when accessing streams from disks. In order to maximize the number of accepted streams, it would be desirable to minimize the peaks (local and global) of both the disk and buffer data schedules. Alternatively, we smooth out the disk data schedule of each individual stream, without allowing the buffer data schedule to exceed it in any round with respect to utilization. It is expected that the aggregation of peak-minimal individual transfers will less likely create disk hot spots; with the buffer constraint, we also eliminate the possibility of rounds with buffer overflow.

The goal of transfer smoothing is consistent with producing majorization-minimal vectors used in the Network Smoothing algorithm. The extra complication that we have here is how to generate majorization-minimal transfer vectors with bounded buffer utilization. Despite the fixed client buffer size that creates a predefined upper bound for the transfer size, here it is not known in advance what is the upper bound for the transfer itself. In the Balanced Prefetching policy we use a greedy algorithm in order to deal with this problem (Figure 6). Basically, we visit each schedule round in increasing order. For each block of this round, we try to identify previous rounds that could prefetch it with lower total transfer size. We also prevent the buffer utilization from becoming higher than the disk utilization in any round. Below we show that this algorithm leads to a majorization-minimal disk utilization vector.

We define the adjusted disk time function $T_{dsk}^{adj}()$ using the expression

$$\begin{aligned} T_{dsk}^{adj}(x) &= \frac{x}{R_{disk}} + 2 \times (T_{trackSeek} + T_{lat}) + 2T_{fullSeek} \times \frac{1}{N} \\ &= \frac{\frac{x}{R_{disk}} + 2 \times (T_{trackSeek} + T_{lat} + T_{fullSeek})}{(1 - \frac{2T_{fullSeek}}{T})} \end{aligned}$$

with $N = \frac{T - 2T_{fullSeek}}{\frac{x}{R_{disk}} + 2T_{lat} + 2T_{trackSeek}}$ from section 6. Thus, we assume that each request is responsible for $\frac{1}{N}$ of the seek time $2T_{fullSeek}$, where N is the maximum number of size x requests that can be serviced in a round of duration T. We define the adjusted disk utilization as: $U_d^{adj}(x) = T_{dsk}^{adj}(x)/T$. All disk accesses are made in multiples of a fixed block size S_B, thus let U_b^B be the buffer utilization of a single block S_B, and U_d^B the corresponding adjusted disk utilization. We should note that the adjusted disk utilization is used by the Balanced Prefetching algorithm in order to approximate better the total disk utilization. However, the final disk transfer schedule, as in the previous policies, is still calculated using the original disk utilization function introduced in Section 2.

Definition 1 We say that the buffer (or round) is *prefetch-full*, if the condition $0 \leq U_d^{adj}(d^d(i)) - U_b(i) < U_b^B$ is satisfied in round i. A round j is called *candidate* with respect to a prefetch-full round i, if j has lower disk utilization than i and there is no in-between k having disk utilization higher than that of i by one or more blocks, $U_d^{adj}(d(k)) - U_d^{adj}(d(i)) > U_d^B$, $j < k < i$. A round is called *invalid*, if $U_b(b(i)) - U_d^{adj}(d_d(i)) > 0$.

```
proc balPref
input  : l, s[], S_B
output : b[], d^d[], d^t[], n^d[], n^t[]
L = l+1, d^d[L-1] = 0, n^d[L-1] = s[l]
for i:0..L-2
    d^d[i] = s[i+1], n^d[i] = s[i]                                   (* assume s[0] = 0 *)
    b^d[i] = b[i-1] + d^d[i] - n^d[i-1]                              (* b[-1] = n^d[-1] = 0 *)
end-for
quantize d^d[] wrt block size S_b and adjust b[]
for t_rnd:1..L-1                                       (* visit each round in increasing order *)
    repeat                                             (* as long as prefetching rounds are found *)
        t_min = t_rnd, U_d^min = U_d^adj(d(t_rnd)), t'_min = -1,   (* keep track of min disk util U_d^adj *)
        t_prf = t_rnd-1, minFound = 0
        while ( t_prf >= 0 and minFound = 0 )                        (* check previous rounds *)
            if ( U_d^adj(d(t_prf)) < U_b(b(t_prf) + S_B) )           (* t_prf is prefetch-full round *)
                if ( U_d^adj(d(t_prf) + S_B) <= U_d^min )    (* t_prf less or equal to t_min wrt disk util *)
                    if ( t'_min = -1 ) t'_min = t_prf end-if (* store t_prf for the case it has candidates *)
                else                                          (* t_prf worse than t_min wrt disk util *)
                    minFound = 1                                     (* don't look further back *)
                end-if
            end-if
            if ( U_d^adj(d(t_prf) + S_B) < U_d^min )                 (* t_prf better than t_min *)
                if ( t'_min = -1 )                                   (* no prefetch-full round yet *)
                    t_min = t_prf, U_d^min = U_d^adj(d(t_prf) + S_B)         (* update t_min *)
                else                                                 (* t_prf is candidate *)
                    t_min = t'_min, minFound = 1      (* update t_min with prefetch-full round t'_min *)
                end-if
            end-if
            t_prf = t_prf - 1
        end-while
        if ( t_min < t_rnd )
            d^d(t_min) = d^d(t_min) + S_B, b(t_min) = b(t_min) + S_B        (* update round t_min *)
            for t_prf:t_min+1..t_rnd-1   b(t_prf) = b(t_prf) + S_B  end-for  (* update in-between rounds *)
            d^d(t_rnd) = d^d(t_rnd) - S_B                                    (* update round t_rnd *)
        end-if
    until (t_min >= t_rnd )
for i:0..L-1                                           (* create disk/network transfer schedules *)
    d^t[i] = T_dsk( d^d[i] ), n^t[i] = T_net( n^d[i] )
end-for
```

Figure 6: The balanced prefetching algorithm produces schedule with majorization-minimal disk utilization vector such that the buffer utilization never exceeds the disk utilization.

The intuition behind this definition is that, when the round i is prefetch-full, no block can be prefetched prior to round i without the buffer constraint ($U_d^{adj}(d_d(i)) - U_b(b(i)) \geq 0$) being violated. We use the term candidate in order to identify rounds that would probably be used for prefetching blocks of following rounds, unless some prefetch-full round occurs in-between to prevent that. By scheduling a block retrieval on a prefetch-full round with candidates, we allow other blocks in rounds following the prefetch-full round to be prefetched during the candidate round.

Definition 2 We define the *deadline* round of a block to be the latest round at which the block can be accessed from the disk without causing a playback timing violation. Then, with respect to a specific block, all the rounds before the deadline round are considered *prefetch* rounds and all the rounds after the deadline round are considered *overdue* rounds.

Lemma 1 *The Balanced Prefetching algorithm schedules each block in a round that satisfies the following properties (in decreasing order of significance):*

i. *No time-constraint violation occurs (overdue rounds)*

ii. *No buffer-constraint violation occurs (invalid rounds)*

iii. *It has the lowest possible disk utilization*

iv. *If there is round with prefetch-full buffer and candidates, it is favored*

v. *It is closest to the deadline round*

Proof: The property (i) comes from lines 12-13 of the algorithm, which limit the range of prefetching rounds to those preceding the current one. The property (ii) is due to lines 14-20, which ensure that prefetching does not happen before prefetch-full rounds. The property (iii) is due to lines 21-27. The prefetch-full round with minimal disk utilization (iv) is recognized in lines 14-16, while the fact that it has candidate rounds is used in lines 24-25. Finally, the closeness to the deadline (v) is controlled by the descending loop in line 28, and the strict inequality in line 21.

Definition 3 We define as *upgrade*, the block movement from round i of the schedule to round j, such that the disk utilization satisfies the condition $U_d^{adj}(d_d(i)) - U_d^{adj}(d_d(j)) \geq 2 \cdot U_d^B$, and no violation occurs with respect to the time and buffer constraints.

Lemma 2 *The Balanced Prefetching algorithm produces a stream schedule in which the disk utilization vector is no lower than the buffer utilization vector, $U_d^{adj}(d_d(i)) - U_b(b(i)) \geq 0$, in any round i and has no possible upgrade.*

Proof: The algorithm is greedy and we will use induction on the length of the schedule L. The schedule trivially satisfies the claim at $L = 2$, with round 0 to access the data from disk and round 1 to send the data over the network. We assume that at $L = k + 1$ the claim is valid. We will try to prove that it is also valid for $L = k + 2$. Let 's assume that we get the schedule of the k first disk accesses to satisfy the above claim, before starting dealing with the disk access of round k (the round $k + 1$ of the schedule is used to only send data over the network).

During the off-line preprocessing, due to the property (i) of Lemma 1, it is not possible to schedule in round k, transfers from the previous rounds. However, by placing blocks of the k round on prefetch-full previous rounds, it is possible to allow blocks of the previous rounds to move farther back into rounds with lower disk utilization.

However, since the algorithm already favors prefetch-full rounds among rounds of the same disk utilization (property (iv)), we really cannot have such blocks in the already optimally scheduled k first rounds.

Therefore, the only optimizing possibility that remains is moving blocks of the round k into previous rounds (i) with lower disk utilization, or (ii) with the same disk utilization, but prefetch-full buffer and candidates (candidates are rounds further back with lower disk utilization, that can be reached without the need to place blocks at higher disk utilization rounds in-between). This search is done in the while-loop of the lines 13-29 of the algorithm. Each of the previous rounds is visited, and a record is kept of the earliest round that can prefetch a block with minimal total disk utilization. If a prefetch-full round shows up in-between, it is going to be the next prefetching round from property (iv). Otherwise, the utilization-minimal round is chosen from property (iii). Finally, property (ii) guarantees that in no round buffer utilization exceeds disk utilization.

The decision to choose prefetch rounds closest to the deadline round (property (v)) leads to buffer minimization. (This could be violated when a prefetch-full round with candidates shows up and it is favored. If these candidates are not subsequently used, the favored rounds could be transformed into prefetch-full rounds again by moving blocks forward. For simplicity, we do not include this modification in our algorithm.)

Definition 4 If $b_1 \geq \cdots \geq b_n$ are integers and $b_i > b_j$, then the transformation $b'_i = b_i - 1$, $b'_j = b_j + 1$, $b'_k = b_k$, $k \neq i, j$ is called a *transfer from i to j.*

Lemma 3 *(Muirhead, 1903) If $\alpha_1, \ldots, \alpha_n, b_1, \ldots, b_n$ are integers and $\alpha \prec b$, then α can be derived from b by successive applications of a finite number of transfers*

Proof: Available by Marshall and Olkin ([12], page 135).

Theorem: 1 The Balanced Prefetching algorithm produces schedule with majorization-minimal disk utilization vector that is not lower than the buffer utilization vector in any round.

Proof: From lemmas 2 and 3.

The algorithm in Fig. 6 includes two nested loops at lines 9 and 13, and another repeat-loop between them at line 10. The for-loop is executed always L times, and the while-loop up to L times in the worst case. It might be less than L when the buffer constraint takes effect and prevents searching for low disk utilization rounds further back. The repeat-loop is dependent on the number of blocks in each round of the original stream vector $\mathbf{s}$, that is, $\frac{max_i s[i]}{S_B}$. Thus, the worst case complexity becomes $O(\frac{max_i s[i]}{S_B} L^2)$. Practically, it takes a few seconds to generate schedules for half-hour clips on the 133MHz RISC servers.

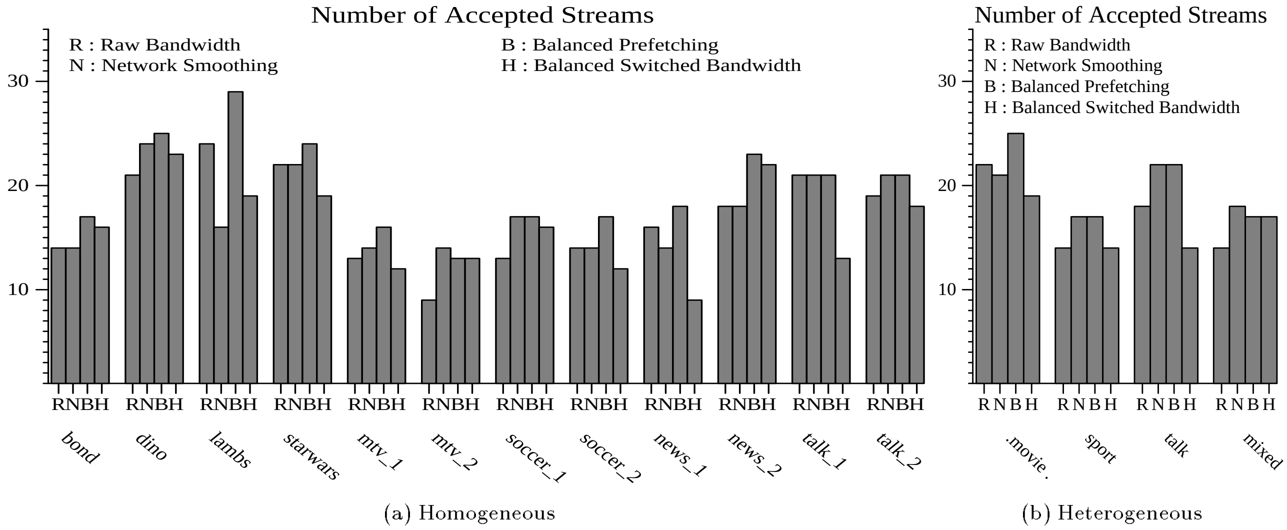


Figure 7: Number of streams successfully accepted, when submitting every 5 secs (a) the same stream (b) streams of each category round-robin. Stream schedules have been generated using the prefetching techniques and policy Raw Bandwidth. The DiskSim HPC3323 disk model has been used and 64MB of memory buffer.

Balanced Switched Bandwidth

Finally, we consider the case that in Switched Bandwidth policy, the transfer rate used is not the minimum that avoids starvation, but the one that keeps the maximum buffer space and adjusted disk time utilizations as close as possible, that is, $\min \| \max_{0 \le i \le L-1} U_d^{adj}(d_d[i]) - \max_{0 \le i \le L-1} U_b(b[i]) \|$. Linear searching is used for finding this disk transfer size at the block granularity. We keep all the other features of Switched Bandwidth the same. This policy will be referred to as Balanced Switched Bandwidth (H).

9 Efficiency Tradeoffs in the Prefetching Techniques

Scheduling properties with negative effect identified in the experiments of Section 7 were the fixed resource reservation, the increased variability in each individual resource, and the imbalance in the maximum utilizations between different resources. Both Network Smoothing and Balanced Prefetching are variations of the Raw Bandwidth policy and do detailed bookkeeping of resource requirements on per-round basis. In addition the Network Smoothing policy reduces explicitly the variation in one of the resources, namely disk bandwidth. The Balanced Prefetching policy adds the extra feature of controlling the relative utilization values between two different resources. Balanced Switched Bandwidth strives to do the same, but using a fixed disk transfer size.

In Figure 7, we compare the number of accepted streams across the policies Raw Bandwidth (R), Network Smoothing (N), Balanced Prefetching (B) and Balanced Switched Bandwidth (H). In Figure 8, the maximum disk time and buffer space utilization are divided by the corresponding numbers of accepted streams. Thus, we can see which resource was the bottleneck in the admission process. For disk time, both the actually utilized and the extra

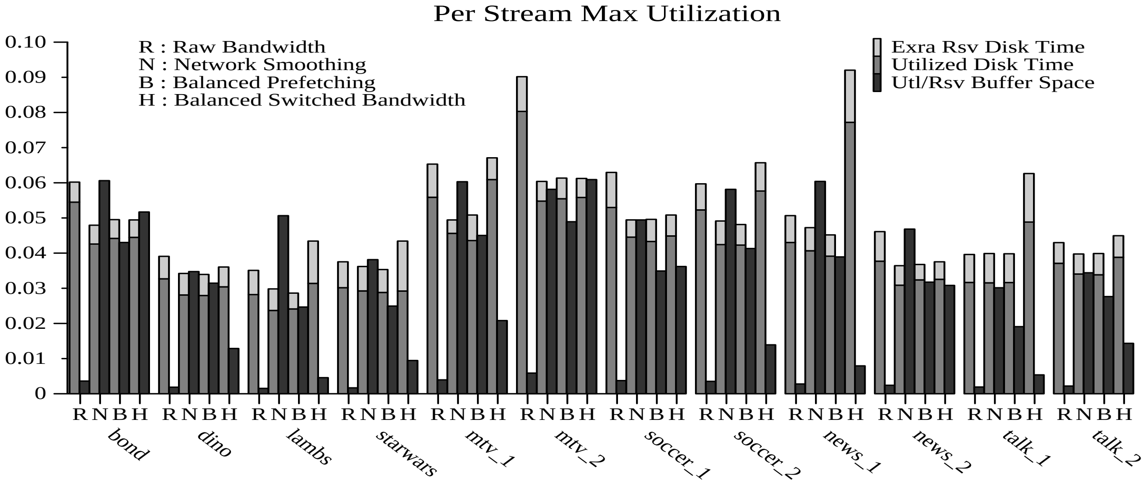


Figure 8: The maximum disk time and buffer space utilization per stream when using the prefetching techniques. The maximum utilization in a fully loaded system has been divided by the corresponding number of streams. The DiskSim HPC3323 disk model is used and 64MB of memory buffer.

reserved fractions of the round time are shown.

It can be seen in Fig. 8 that the drawback of policy Raw Bandwidth is the imbalance in the maximum disk time and buffer space utilizations. Policy Network Smoothing manages to reduce the maximum round disk time. It turns out that the choice of 4MB as buffer size prohibits the server buffer from becoming bottleneck resource in several cases, as well. Thus, Network Smoothing outperforms policies Balanced Switched Bandwidth and Raw Bandwidth in all streams except `bond`, `lambs`, `news_1` and `news_2`. In streams `bond, lambs` and `news_2`, policy Balanced Switched Bandwidth has the maximum of disk and buffer utilization maxima lower than that of Network Smoothing. In streams `lambs` and `news_1`, the disk utilization of Raw Bandwidth is lower than the buffer utilization of Network Smoothing. Thus, Raw Bandwidth seems to outperform Network Smoothing in these cases.

The extra feature added by Balanced Prefetching when compared with Network Smoothing is the balancing of the utilizations between different resources. Looking back on Figure 8, we find out that in almost half of the streams, namely `bond, dino, lambs, starwars, mtv_1, soccer_2, news_1, news_2`, the maximum of transfer and buffer utilizations is lower in Balanced Prefething than in Network Smoothing. In the rest of the streams, high transfer peaks at the beginning rounds, define the maximum utilization and don't leave space for improvement with our minimal initial latency assumption. Not surprisingly, Balanced Prefetching manages to track or outperform the best of the other three policies in most of the homogeneous and heterogeneous workload experiments (except `mtv_2` and `mixed`).

The cases `mtv_2` and `mixed`, where Network Smoothing accepts one more stream than Balanced Prefetching, are due to the way that local resource maxima combine with global resource maxima for the interarrival time used.

Fujitsu M1606S			
Data Bytes per Drive	1,091,580,000	Head Switch Time	N/A
Data Sectors per Track	70 to 125	Maximum Seek	25 msec
Data Cylinders	3,456	Rotational Latency	5.56 msec
Data Surfaces	6	Command overhead Time	N/A
Zones	N/A	Internal Transfer Rate	
Buffer Size	512 Kbytes	Inner to Outer Zone Burst	4.34 to 7.84 Mbytes
Track to Track Seek	3 msec	Inner to Outer Zone Sust.	N/A

Table 4: Features of the Fujitsu disk used for the measurements.

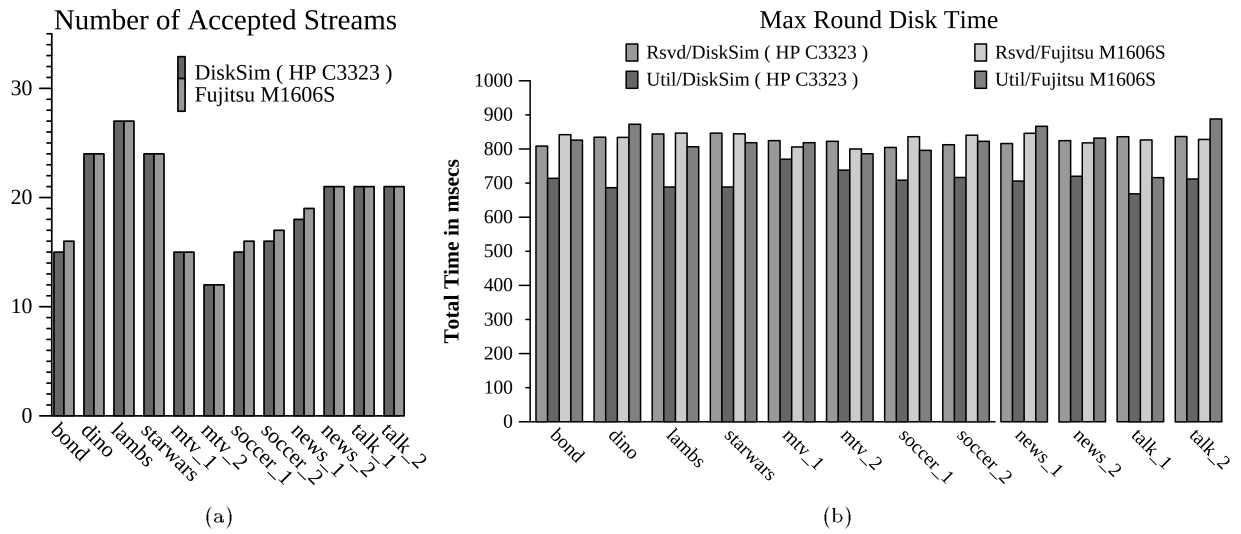


Figure 9: The (a) number of streams and (b) maximum disk time, for scheduling policy Balanced Prefetching at 32MBytes memory buffer. Both reserved and actually utilized disk time are shown for the DiskSim HPC3323 model and the hardware Fujitsu M1606S disk. (The Fujitsu max disk time is averaged over five repetitions.)

In general, the maximum utilization measurements illustrate the basic difference between the policies. However, local resource maxima can also have an effect on the number of accepted streams, when stream requests arrive with interarrival time larger than zero. In fact, policy Network Smoothing demonstrated some tendency to use most of the available buffer size (up to the limit of 4MB), even when that is not necessary, as in `talk_1`.

10 Hardware Disk Experiments

In this section we compare results from the DiskSim simulator with measurements using a hardware Fujitsu M1606S disk at 32Mbytes server memory buffer. We didn't have the HPC3323 disk that was modelled by DiskSim. Also, modification of the DiskSim to match the Fujitsu disk was not possible since the parameters required were not available either in the data sheet or the mode pages of the M1606S disk. The basic parameters of HPC3323 (Table 3) and Fujitsu M1606S (Table 4) are relatively close, though (within 10%). The Fujitsu data sheet doesn't report sustained internal transfer rate; thus, we used the 70% of the burst 4.34MBytes/sec in accordance with the HPC3323. The network bandwidth of 100Mbit switched Ethernet was used for sending the data to the client.

In Figure 9, we show the reserved and actually utilized disk time for both the DiskSim model and the Fujitsu disk. The number of accepted streams for the two cases are the same within one. The difference is due to the slightly higher transfer rate of the Fujitsu disk (4.34MB/sec vs 4MB/sec). The Fujitsu disk utilized time is the average of five repetitions; transient overloads, although rare, did occur. In general, the difference between simulation and measurement numbers are within 10-20%. This error is tolerable given the minor differences in the disk characteristics. Other possible reasons are the system bus contention between network and disk transfers and also the SCSI bus contention (10 MB/sec) that was shared between the Fujitsu and the system disk (IBM DSAS-3720 720MB SCSI disk).

11 Conclusions and Future Work

In the present work, we focused on the server side of a video distribution system. We identify the basic system resource and stream timing constraints, and attempt to maximize the number of concurrent streams. We compare previously proposed scheduling policies with respect to the number of concurrent streams that they successfully accept, and also the maximum required disk time and server buffer space that they require for each stream. As expected, keeping high the maximum disk bandwidth needed by each stream, when combined with fixed resource reservations, can limit significantly the number of streams serviced concurrently. In fact, we demonstrate that the schemes Switched Bandwidth and Raw Bandwidth can outperform Constant Bandwidth and Fixed Buffer by more than a factor of two. Switched Bandwidth, by reducing the maximum required disk bandwidth seems to perform well, unless the characteristics of the stream require excessive buffer space. Raw Bandwidth, by keeping memory requirements always to the minimum and making separate resource reservations for each round, outperforms the other polices in almost all the cases.

Subsequently we study alternative techniques for reducing simultaneously the maximum disk bandwidth and server memory buffer required by the system. We propose the policy Balanced Prefetching that minimizes per-stream disk bandwidth variability without allowing the buffer utilization to exceed disk bandwidth utilization in any round. We compare this policy with the Raw Bandwidth, and also with the Network Smoothing adapted to the server context, and the Balanced Switched Bandwidth. Network Smoothing tends to fill up the available buffer space in order to maximize the smoothing of the data transfers. This increases buffer utilization and can prevent new streams from being admitted. On the other hand, Balanced Switched Bandwidth does not always reduce the maximum disk bandwidth required depending on the characteristics of the stream. We show that Balanced Prefetching manages to keep the maximum of the disk bandwidth and memory buffer utilizations lower than that of the other policies. Thus, it is within one stream in the worst case and usually outperforms the other policies by up to a factor of two with respect to the streams accepted.

Accurate disk modeling, improved multiplexing from round by round resource reservation, reduced variability within each resource requirement over the duration of a stream, and appropriate balancing between different shared resources, all proved beneficial properties for improving video server efficiency. In future work, we would like to extend our techniques to multiple disk configurations, and study how the extra complexity affects disk load balancing. Another challenging extension would be to consider all of disk bandwidth, memory buffer and network bandwidth as bottleneck resources simultaneously, and try to adapt the stream schedules appropriately for improving throughput.

References

[1] *MPEG-2 Encoder/Decoder, Version 1.2.* 1996. MPEG Software Simulation Group.

[2] Biersack, E., Thiesse, F., and Bernhardt, C. Constant Data Length Retrieval for Video Servers with Variable Bit Rate Streams. In *IEEE Multimedia Computing and Systems* (Hiroshima, Japan, June 1996), pp. 151–155.

[3] Biersack, E. W., and Hamdi, M. Cost-optimal Data Retrieval for Video Servers with Variable Bit Rate Video Streams. In *Int'l Work. Network and Operating System Support for Digital Audio and Video* (Camb., UK, July 1998), pp. 295–302.

[4] Bolosky, W. J., Barrera, J. S., Draves, R. P., Fitzgerald, R. P., Gibson, G. A., Jones, M. B., Levi, S. P., Myhrvold, N. P., and Rashid, R. F. The Tiger Video Fileserver. In *Int'l Workshop on Network and Operating System Support for Digital Audio and Video* (Zushi, Japan, Apr. 1996), pp. 97–104.

[5] Bolosky, W. J., Fitzgerald, R. P., and Douceur, J. R. Distributed Schedule Management in the Tiger Video Fileserver. In *ACM Symp. Operating Systems Principles* (Saint-Malo, France, Oct. 1997), pp. 212–223.

[6] Chang, E., and Zakhor, A. Cost Analyses for VBR Video Servers. *IEEE Multimedia* (Winter 1996), 56–71.

[7] Chang, R., Chen, M., Ko, M., and Ho, J. Optimizations of Stored VBR Video Transmission on CBR channel. In *Proc. SPIE VVDC:Performance and Control of Network Systems* (Dallas, TX, Nov. 1997), pp. 382–392. vol. 3231.

[8] Feng, W.-C., and Rexford, J. A Comparison of Bandwidth Smoothing Techniques for the Transmission of Prerecorded Compressed Video. In *IEEE INFOCOM* (Kobe, Japan, Apr. 1997), pp. 58–66.

[9] Ganger, G. R., Worthington, B. L., and Patt, Y. N. The DiskSim Simulation Environment: Version 1.0 Reference Manual. Tech. Rep. CSE-TR-358-98, Dept. Electrical Engineering and Computer Science. University of Michigan, Ann Arbor, Michigan, Feb. 1998.

[10] Jones, M. B. The Microsoft Interactive TV System: An Experience Report. Tech. Rep. MSR-TR-97-18, Microsoft Research, 1997. ftp://ftp.research.microsoft.com/pub/tr/tr-97-18/tr-97-18.html.

[11] Makaroff, D., Hutchinson, N., and Neufeld, G. An Evaluation of VBR Disk Admission Algorithms for Continuous Media File Servers. In *ACM Multimedia* (Seattle, WA, May 1997), pp. 143–154.

[12] Marshall, A. W., and Olkin, I. *Inequalities: Theory of Majorization and its Applications.* Academic Press, New York, 1979.

[13] Martin, C., Narayanan, P. S., Ozden, B., Rastogi, R., and Silberschatz, A. The Fellini Multimedia Storage System. In *Multimedia Information Storage and Management* (Boston, MA, 1996), Kluwer Academic Publishers.

[14] McManus, J., and Ross, K. A Dynamic Programming Methodology for Managing Prerecorded VBR Sources in Packet-Switched Networks. Telecommunications Systems **9** (1998), 223–247.

[15] McVoy, L., and Kleiman, S. R. Extent-like Performance from a Unix File System. In *USENIX Winter Technical Conference* (Dallas, TX, 1991), pp. 33–43.

[16] Ng, S. W. Advances in Disk Technology: Performance Issues. *Computer* **31**, 15 (May 1998), 75–81.

[17] Paek, S., and Chang, S.-F. Video Server Retrieval Scheduling for Variable Bit Rate Scalable Video. In *IEEE Multimedia Computing and Systems* (Hiroshima, Japan, June 1996), pp. 108–112.

[18] Pancha, P., and Zarki, M. E. MPEG Coding for Variable Bit Rate Video Transmission. *IEEE Communications Magazine* (May 1994), 54–66.

[19] Patterson, R. H., Gibson, G. A., Ginting, E., Stodolsky, D., and Zelenka, J. Informed Prefetching and Caching. In *ACM Symp. Operating Systems Principles* (Copper Mountain Resort, CO, Dec. 1995), pp. 79–95.

[20] Rose, O. Statistical properties of MPEG video traffic and their impact on traffic modeling in ATM systems. In *Annual Conference on Local Computer Networks* (Minneapolis, MN, 1995), pp. 397–406.

[21] Salehi, J. D., Zhang, Z.-L., Kurose, J. F., and Towsley, D. Supporting Stored Video: Reducing Rate Variability and End-to-End Resource Requirements through Optimal Smoothing. In *ACM SIGMETRICS* (Philadelphia, PA, May 1996), pp. 222–231.

[22] Sen, S., Dey, J., J.Kurose, Stankovic, J., and Towsley, D. CBR transmission of VBR stored video. In *SPIE Symposium on Voice Video and Data Communications* (Dallas, TX, Nov. 1997), pp. 26–36.

[23] Zhao, W., and Tripathi, S. K. Bandwidth-Efficient Continuous Media Streaming Through Optimal Multiplexing. In *ACM SIGMETRICS* (Atlanta, GA, May 1999), pp. 13–22.

Appendix A. Algorithmic derivation of the basic scheduling policies

In all the algorithms below except `quantize`, input is the vector **s**, its length l, and the block size S_B and output, the schedule length L and the vectors **b**, $\mathbf{d}^d$, $\mathbf{d}^t$, $\mathbf{n}^d$, $\mathbf{n}^t$. It is assumed that $s[0] = b[-1] = n^d[-1] = d^d[-1] = 0$.

1. **proc** `rawBandwidth`

    $\texttt{L = l+1, d}^d\texttt{[L-1] = 0, n}^d\texttt{[L-1] = s[l]}$

    for $\texttt{i:0..L-2}$

    $\quad \texttt{d}^d\texttt{[i] = s[i+1], n}^d\texttt{[i] = s[i], b[i] = b[i-1] + d}^d\texttt{[i] - n}^d\texttt{[i-1]}$

    end-for

    $\texttt{quantize(L, d}^d\texttt{, b, S}_B\texttt{)}$

    for $\texttt{i:0..L-1 d}^t\texttt{[i] = T}_{dsk}\texttt{(d}^d\texttt{[i]), n}^t\texttt{[i] = T}_{net}\texttt{(n}^d\texttt{[i])}$ **end-for**

    end-proc

2. **proc** `maximumBandwidth`

    $\texttt{L = l+1, s}_{max}\texttt{ = ceil((max}_{1\le i\le l}\texttt{ s[i]) / S}_B\texttt{) * S}_B$

    $\texttt{b[0] = s}_{max}\texttt{, d}^d\texttt{[0] = s}_{max}\texttt{, n}^d\texttt{[0] = 0, b[L-1] = s}_{max}\texttt{, d}^d\texttt{[L-1] = 0, n}^d\texttt{[L-1] = s}_{max}$

    for $\texttt{i:1..L-2 b[i] = 2*s}_{max}\texttt{, d}^d\texttt{[i] = s}_{max}\texttt{, n}^d\texttt{[i] = s}_{max}$ **end-for**

    for $\texttt{i:0..L-1 d}^t\texttt{[i] = T}_{dsk}\texttt{(max}_{0\le j\le L-1}\texttt{d}^d\texttt{[j]), n}^t\texttt{[i] = T}_{net}\texttt{(max}_{0\le j\le L-1}\texttt{n}^d\texttt{[j])}$ **end-for**

    end-proc

3. **proc** `constantBandwidth`

    $\texttt{L}_{bld}\texttt{ = min l}_{bld} \le 30$ `to minimize` $\max_{0\le j\le L-1} b[j]$

    $\texttt{L = l + L}_{bld}\texttt{ + 1, s}_{sum} = \sum_{1\le j\le l} s[j]$

    $\texttt{s}_{max}$ `= min request (` $\times S_B$ `) to keep` $\texttt{d}^d\texttt{[]}$ `valid with the` $\texttt{L}_{bld}$

    for $\texttt{i:0..L-1}$

    $\quad$ **if** $\texttt{(i <= L}_{bld}\texttt{) n}^d\texttt{[i] = 0}$ **else** $\texttt{n}^d\texttt{[i] = s[i-L}_{bld}\texttt{]}$ **end-if**

    $\quad$ **if** $\texttt{(s}_{sum}\texttt{ >= 0) d}^d\texttt{[i] = s}_{max}\texttt{, s}_{sum}\texttt{ = s}_{sum}\texttt{ - s}_{max}$ **else** $\texttt{d}^d\texttt{[i] = 0}$ **end-if**

    $\quad \texttt{b[i] = b[i-1] + d}^d\texttt{[i] - n}^d\texttt{[i-1]}$

    end-for

    for $\texttt{i:0..L-1 d}^t\texttt{[i] = T}_{dsk}\texttt{(max}_{0\le j\le L-1}\texttt{d}^d\texttt{[j]), n}^t\texttt{[i] = T}_{net}\texttt{(max}_{0\le j\le L-1}\texttt{n}^d\texttt{[j])}$ **end-for**

    end-proc

4. **proc** `switchedBandwidth`

    $\texttt{L = l+1, s}_{sum} = \sum_{1\le j\le l} s[j]$

    $\texttt{s}_{max}$ `= min request (` $\times S_B$ `) to keep` $\texttt{d}^d\texttt{[]}$ `valid under this policy`

    `reqBytes = 0, accBytes = 0`

```
    for i:1..L-1
        reqBytes = reqBytes + s[i],                              (* reqBytes = Σ_{j=1}^{i} s[j] *)
        while ( reqBytes > accBytes )
            find max k, 0 <= k <= i, with d^d[k] = 0
            d^d[k] = s_max
            accBytes = readBytes + d^d[k]                        (* accBytes = Σ_{j=0}^{i} k[j] *)
        end-while
    end-for
    for i:0..L-1  n^d[i] = s[i], b[i] = b[i-1] + d^d[i] - n^d i-1]  end-for
    for i:0..L-1  d^t[i] = T_dsk( max_{0≤ j≤ L-1} d^d[j] ), n^t[i] = T_net( max_{0≤ j≤ L-1} n^d[j] )  end-for
end-proc
```

5. **proc** fixedBuffer

```
    L = l+2, s_max = max_{1 ≤ j ≤ l} s[j]
    d^d[0] = d^d[1] = s_max, b[0] = s_max, b[1] = 2 * S_max, n^d[0] = n^d[1] = 0
    for i:2..L-1
        d^d[i] = min( s_max, n^d[i-1] )
        n^d[i] = s[i-1], b[i] = b[i-1] + d^d[i] - n^d[i-1]
    end-for
    quantize ( L, b_d, b, S_B )
    for i:0..L-1  d^t[i] = T_dsk( max_{0≤ j≤ L-1} d^d[j] ), n^t[i] = T_net( max_{0≤ j≤ L-1} n^d[j] )  end-for
end-proc
```

6. **proc** quantize

```
Input: L, d^d, b, S_B, Output: d^d, b
    for  i:0..L-2
        increase d_d[i] to the closest multiple of S_B
        adjust b[i]
        reduce accordingly d_d[j] in rounds j:i+1..L-2
    end-for
end-proc
```