Performance Issues of Multimedia Applications*

Edmundo de Souza e Silva[1], Rosa M. M. Leão[1], Berthier Ribeiro-Neto[2], and Sérgio Campos[2]

[1] Federal University of Rio de Janeiro
COPPE/PESC, Computer Science Department {rosam, edmundo}@land.ufrj.br
[2] Federal University of Minas Gerais
Computer Science Department {berthier,scampos}@dcc.ufmg.br

Abstract. The dissemination of the Internet technologies, increasing communication bandwidth and processing speeds, and the growth in demand for multimedia information gave rise to a variety of applications. Many of these applications demand the transmission of a continuous flow of data in real time. As such, continuous media applications may have high storage requirements, high bandwidth needs and strict delay and loss requirements. These pose significant challenges to the design of such systems, specially since the Internet currently provides no QoS guarantees to the data it delivers. An extensive range of problems have been investigated in the last years from issues on how to efficiently store and retrieve continuous media information in large systems, to issues on how to efficiently transmit the retrieved information via the Internet. Although broad in scope, the problems under investigation are tightly coupled. The purpose of this chapter is to survey some of the techniques proposed to cope with these challenges.

1 Introduction

The fast development of new technologies for high bandwidth networks, wireless communication, data compression, and high performance CPUs has made it technically possible to deploy sophisticated communication infrastructures for supporting a variety of multimedia applications. Among these we can distinguish, for instance, quality audio and video on demand (to the home), virtual reality environments, digital libraries, and cooperative design.

Multimedia objects, such as movies, voice extracts, texts, and pictures, are usually stored in compressed (encoded) form on the disks of a multimedia server. Since the encoded objects might be long, the playing of an object should not be delayed until the whole object is transmitted. Instead, the playing of the object should be initiated as early as possible.

A common characteristic among multimedia applications is the so-called *continuous* nature of their generated data. In continuous media (CM), strict timing relationships exist that define the schedule by which CM data must be rendered

* This work is supported in part by grants from CNPq/ProTeM. E. de Souza e Silva is also supported by additional grants from CNPq/PRONEX and FAPERJ.

M.C. Calzarossa and S. Tucci (Eds.): Performance 2002, LNCS 2459, pp. 374–404, 2002.

(e.g., a video displayed, 3D graphics rendered, or audio played out). These timing relationships coupled with the high aggregate bandwidth needs, the high individual application bandwidth needs, and the high storage requirements pose significant challenges to the design of such systems. This is particularly troublesome in the scenario of the Internet, which is beginning to be used to convey multimedia data but which was not designed for this purpose.

In this work, we discuss the main technical issues involved in the design and implementation of practical (distributed) multimedia systems. We take a particular view, which divides the system in three main components: the multimedia server, the resource sharing techniques for transmitting data across the network, and methods for improving the utilization of network bandwidth and buffers. We look at each of these components, reviewing the related literature, introducing the key underlying technical issues, and providing insights on how each of them impacts the performance of the multimedia system.

2 The Multimedia Server

The *multimedia server* is a key component of a distributed multimedia system. Its performance (in terms of the number of clients supported) affects the overall cost of the system and might be decisive for determining economical viability. As a result, studying the performance of multimedia servers is an important problem which has received wide attention lately [8,42,19,18,17,28,30,34,33,45, 46,54,63]. The server is a computer system containing one or more processors, a finite amount of memory M, and a number D of disks. The disks are used to store compressed multimedia objects, which are retrieved by the clients.

Compressed video objects are composed of frames, where a *frame* is a snapshot of the state of all bits in the screen. To decode the frames in a stream, the client has to store them in memory which requires some level of buffering. The frames are consumed at a constant rate. Since the number of bits in each component frame varies, the input *bit rate* and the output *bit rate* for the buffer at the client side are variable (VBR).

It is common to implement the server such that it always sends data to the client in blocks of fixed size. When it is possible to always send the same number of blocks in the unit of time, we say that the traffic flows at a constant bit rate (CBR). Keeping a CBR (or nearly CBR) traffic implies that the frame rate varies at the input of the client buffer. To avoid interruption of the display, a much larger buffer might be required at the client to compensate for variations in the frame arrival rate. In Sec. 3 we discuss traffic smoothing techniques to compensate for these rate variations. Several proposals in the literature are then based on CBR assumptions [9,14,53,55,67].

Due to disk seek and rotational delays, one or more sectors need to be retrieved from the server during each disk access to attain good performance. The set of disk sectors that the server sends to the client at one time is here called a *data block*. Each data block is stored in the buffer of the client and consumed from there. While the client decodes a data block, other clients can be served. This way the server is able to multiplex the disk bandwidth among various clients, which are served concurrently. The approach works because the total disk bandwidth available at the server far exceeds the display rate with which each client consumes bytes.

Let O_i be a reference to the *ith* multimedia object in the server and b_i be a reference to any data block of the object O_i. Consistently with several prototype implementations, we assume that the data blocks of each object O_i are all of the same size. The data blocks of distinct objects, however, might be of different sizes (i.e., $size(b_i) \neq size(b_j)$).

A client makes a request for an object O_i. If this request is admitted into the system, the server starts sending blocks of the object O_i to the client machine. The client might have to wait until the buffer fills up to a pre-defined threshold before starting to play the object. The time interval between the client request and the beginning of the display is called *startup latency*. To send the blocks to the client, the server first retrieves them from disk into main memory. Thus, *buffers* are also required at the server side.

A client gets a block of data and starts consuming it. Before consuming all the data in that block, the client must get the next block of data for the object it is playing. Otherwise, interruption in the service will occur. In the case of a movie, this means that the motion picture might suddenly freeze in front of the user (also called *hiccup*). Thus, each client must get the blocks of data in a timely fashion.

2.1 The Size of the Multimedia Server

The size of a multimedia server installation is a direct function of the number D of disks in the system. Given the server size, the maximum load that can be imposed to the system is determined, as we now discuss.

The number of disks used in a multimedia server is related to the bandwidth demand, to the storage requirements, and to the amount of capital available for investing in the system. Consider, for instance, movie objects encoded in MPEG-2 (320×240 screen). The typical bandwidth requirement for such objects is 1.5 Mbps (mega bits per second). Thus, to support the display of 1500 MPEG-2 movie objects, a total net bandwidth of 2250 Mbps is required. The scenario 1 below illustrates this situation.

Scenario 1: SCSI Technology: effective disk bandwidth: 60 MBps = 480 Mbps[1]; disk storage capacity: 73.4 GB; maximum number of concurrent customers: 1500; bandwidth requirement of 1 MPEG-2 object: 1.5 Mbps; storage requirement of 1 MPEG-2 object: 1 GB; effective server bandwidth required: 1500 * 1.5 = 2250 Mbps; rough number of disks required in the server: $\lceil 2250/480 \rceil = 5$; number of distinct MPEG-2 objects in storage: $\lfloor 5 * 73.4 \rfloor = 367$; number of disks with 20% redundancy: 6.

[1] Estimated bandwidth in mega bits per second (Mbps), including seek time, for current disk technology.

Thus, to provide 1500 customers with real-time MPEG-2 streams we need a total of 5 SCSI disks (current technology). This computation is quite rough, since it does not consider memory and bus bandwidth bottlenecks, and redundancy for fault tolerance. With a 20% degree of redundancy, a total of 6 disks would be required.

Besides bandwidth, the storage requirements need also to be taken into account. Since the storage capacity of each SCSI disk considered above is 73.4 GB (giga bytes) and each MPEG-2 object of 1 hour and 40 minutes takes roughly 1 GB, with 5 disks we can store up to 367 MPEG-2 objects.

However, 367 is not really the number of movies one would expect to find in a video store. Typically, at least a few thousand movies should be available. One alternative is to use cheaper technology, such as IDE, to provide plenty of storage capacity with good bandwidth delivery. For instance, consider the scenario 2 immediately below.

Scenario 2: IDE Technology: effective disk bandwidth: 16 MBps = 128 Mbps; disk storage capacity: 80 GB; maximum number of concurrent customers: 1500; bandwidth requirement of 1 MPEG-2 object: 1.5 Mbps; storage requirement of 1 MPEG-2 object: 1 GB; effective server bandwidth required: 1500 * 1.5 = 2250 Mbps; rough number of disks required in the server: $\lceil 2250/128 \rceil = 18$; number of distinct MPEG-2 objects in storage: $\lfloor 18 * 80 \rfloor = 1440$; number of disks with 50% redundancy: 27.

Thus, we can now store up to 1440 distinct MPEG-2 objects in 18 IDE disks of 80 GB each. And this is accomplished while attending up to 1500 concurrent customers as before. Notice that we now use a degree of redundancy of 50%, because disks based on IDE technology are not as reliable as disks based on SCSI technology. Since each IDE disk in scenario 2 costs about 1/6 of an SCSI disk in scenario 1, the configuration in scenario 2 is either cheaper or price equivalent to the configuration in scenario 1. Most important, replacing IDE disks is easier because they can be bought everywhere at any time (i.e., IDE technology is really ubiquitous nowadays).

The data block size, contrary to the number of disks, is related more to the design of the system itself. Given a number of disks and a memory space for buffers, usually an optimal block size can be determined. The block size can be chosen to minimize the cost per stream or to maximize the number of streams that can be supported concurrently. One primary constraint has to be met: the block size must be large enough to amortize the delays due to seek and rotational latency. Block sizes ranging from 512 Kbytes to 1 Mbytes are usually large enough to accomplish this effect.

2.2 The Layout

The blocks which compose the various multimedia objects are laid out across the disks in the system. A simplistic approach is to store all blocks of the same object on a single disk. The main advantage of this approach is simplicity and ease of maintenance. However, there is a considerable disadvantage. If a popular video is heavily requested, the disk that stores that video will be overloaded. Thus, severe load imbalance might result, which limits the number of clients that can be served. More sophisticated strategies involve spreading the blocks of the same object across multiple disks (the so called *striping* techniques).

Layout Using Striping. The key idea of striping is to spread out the data blocks of each object across the disks of the server. This way, during the service time of an object, each client request is continuously moved from one disk to another and shares the bandwidth of all the disks in the system. We say that the object storage has been decoupled from the disks and call this effect *object decoupling* (see Figure 1). Object decoupling provides a load balancing effect which allows a higher number of clients in the system and a better utilization of disk bandwidth.

Usually, when a striped layout is used, the server operates in cycles. At each cycle of duration T, the server retrieves one data block for each client in the system (this retrieval incurs in three delays: seek time, rotational latency, and transfer time). While that client consumes the block, other clients can be served. Discontinuities in the service are avoided by guaranteeing that each client is served in every cycle. When all clients in the system have been served, the server *sleeps* if there is still time available in the current cycle of duration T.

To accommodate objects with distinct bandwidth requirements, we can simply allow the sizes of the storage units to vary. For instance, objects O_i and O_j will have blocks sizes b_i and b_j ($b_i \neq b_j$), respectively. Each block is stored as a separate storage unit. For a same object O_i, however, the block sizes are all the same (i.e., $b_i[j] = b_i[j+1]$). At the disk level, one can keep the storage unit size constant to avoid fragmentation. For an object that has higher bandwidth requirements, two or more storage units can be combined to compose a data block, as illustrated in Figure 1. Each data block of the object O_i is composed of a single storage unit while each data block of the object O_j is composed of two storage units. We see that two or more disks might now be involved in the retrieval of a unique data block. Since the storage unit size is kept constant, storage and bandwidth fragmentation problems are minimized.

Random Data Allocation Layout. Striping layouts are good because they provide object decoupling. However, in general, all striping strategies impose a tight coupling between the layout itself and the block access pattern as a way to balance the load among the various disks. To avoid this tight coupling, an alternative is to employ a random data allocation. It can be shown that a random layout is as good as striping in terms of performance [64], but presents important advantages as we briefly point out here.

A *random data allocation* layout uses storage units that are all of the same size. However, contrary to the striping approach, each storage unit is stored in a disk position that is determined according to the following procedure: (a) select a disk at random; (b) within that disk, select a free position at random.

As a result, storage units are placed randomly across the disks of the system. Objects with higher bandwidth requirements are served by combining several

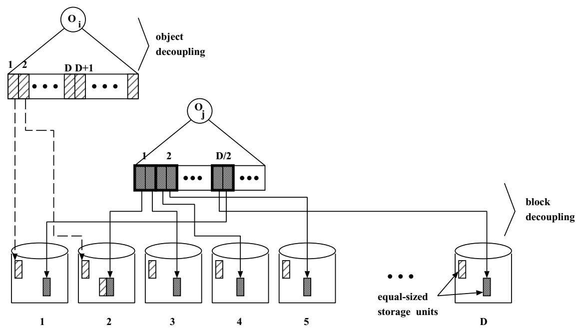

Fig. 1. Hybrid layout with equal-sized stripe units and block sizes which vary from one object to the other.

storage units to form a data block. Also, the physical location of the data blocks is now independent of the block access pattern.

A random data allocation layout provides the following characteristics: object decoupling; access pattern decoupling; no disk storage fragmentation; small probability of prolonged bandwidth fragmentation; good performance. The good performance is attained because the load tends to be statistically balanced among the various disks. Random data allocation is the only layout scheme that provides all these features together. Because of this, it simplifies the overall design and implementation of the system. Therefore, we argue that it is the paradigm of choice for the design and implementation of multimedia servers in general.

Comparative Performance Analysis: Striping versus Random Layout. In [64] a detailed comparison between a server based on striping and a server based on a random layout is presented. The experimental results show that system performance with a random layout is competitive or superior to the performance obtained with a striping layout. This is the case not only for unpredictable access patterns generated by sophisticated interactive applications such as virtual worlds and scientific visualizations, but also for sequential access patterns generated by more standard video applications.

To illustrate, let us focus on the case of standard video applications. When only a small amount of buffer is allowed at the server (say, 1.5 MBytes per stream), a striping layout performs slightly superior to a random layout providing an increase in the maximum number of streams sustained of roughly 5%. If the amount of buffer per stream is allowed to increase to 3.5 MBytes per stream, both layouts lead to the same overall performance. Additional increments in buffer space per stream favor the random layout, whose performance becomes superior.

Assume now that more disk space is made available, such that data blocks can be replicated. This is useful, for instance, to improve reliability against disk failure. Consider a 25% degree of replication of video data blocks. This is good for a random layout because replicated blocks can be used to alleviate the load of momentarily overloaded disks. In this case, with a buffer space of 3.5 MBytes per stream, a server based on a random layout presents performance (maximum number of streams sustained by the server) that is 10-15% higher than the performance of a server based on a striping layout.

2.3 Staging, Reconfiguration, and Fault Tolerance

In practical installations, there are other important issues that have to be considered for proper operation of a multimedia server. Among these, we distinguish the staging of new videos, the reconfiguration of the server to improve performance, and fault tolerance against failures of service in the disks of the server. In this section, we discuss these issues in more detail and compare their relative performance considering random-based and striping-based servers.

The Staging Mechanism. Since multimedia objects might be quite large (particularly movie objects), the number of objects that can be stored on the disks of the system might be quite limited. This implies that the objects in the system need to be replaced by new ones from time to time. Since the new objects are usually loaded from tape, we call this process the *staging mechanism.* This is an issue which has not received much attention in the specialized literature but which is critically important in any practical system.

For offline staging, the use of block decoupling provides an efficient solution. If online staging is desired, the admission control and the scheduling processes are affected because a new stream has to be admitted and scheduled. This type of stream might require higher bandwidth because redundant data (such as copies of the data to support fault tolerance) have also to be updated. For a striped layout under heavy load, it might be the case that two fragmented pieces of bandwidth are available but cannot be used for staging because a coalesced bandwidth is required. For instance, this might be the case of a new stream which is been fed live to the server. If the new stream is not live, then there is no problem because the staging can proceed in non real-time mode.

Staging has similar costs both for a striped and for a random layout, whenever the staging is offline. For online staging, a random layout is advantageous. For instance, a random layout makes it easier to deal with the staging of a new stream which is been fed live. Also, a random layout allows the staging of a new object at a rate which is different from its playout rate which is often more difficult to do with a striped layout.

Disk Reconfiguration. In practical situations, it is reasonable to expect that the demand on a given server system might eventually exceed its planned capacity. For instance, it might be the case that the demand for disk bandwidth exceeds the total disk bandwidth currently available in the server. This problem can be fixed by adding new disks to the system and copying data blocks (of the objects already in the system and of new objects) into the new disks. This is what we call *disk reconfiguration.* We would like to be able to reconfigure the system while maintaining the server fully operational.

Consider that we have D disks in the system and that we want to add K new disks. For simplicity, we consider here that the new disks are of the same capacity and of the same bandwidth as the disks already installed in the server. Consider also that no new objects will be added to the system. With current disk technology, the extra K disks can be "hot" inserted into the system while it is running. Thus, no interruption in service is required. However, the storage units need to be remapped to take advantage of the newly available bandwidth.

To exemplify, assume an installation with 8 disks to which 2 new disks are added. We have that $D = 8$ and $K = 2$. In this case, it can be shown that 80% of all storage units need to be moved if the layout is done with striping, while only 20% of all storage units need to be moved if the layout is random. Thus, we conclude that it is much cheaper to reconfigure an installation when the layout is random.

Fault Tolerance. Maintaining the integrity of the data and its accessibility are crucial aspects of a multimedia server. Particularly critical are failures of the disks of the system. While each individual disk is fairly reliable, a large set of disks presents a considerably higher likelihood of failure of a component. With a multimedia server, it is particularly important to provide tolerance to this type of failure because failure of a single disk might disrupt the service to all clients in the system. Basically, fault tolerance is provided by the maintenance of redundant information about the data. Two basic schemes can be used: full replication and parity encoding.

With parity encoding, the D disks of the system are divided in n_g groups. Let g, $g = D/n_g$, be the number of disks per group. For each group, one of the disks is reserved for storing parity information while the remaining $g - 1$ are used for storing data. The parity information is computed as the *exclusive-or* of the storage units in the $g - 1$ disks. We use storage units instead of data blocks because, in case block decoupling is used, data blocks are not confined to a single disk. Let $su_i[k]$, $su_i[k+1]$, $\ldots$, $su_i[k+g-2]$ be $g-1$ consecutive storage units (belonging to data blocks of object O_i) which appear each in a separate disk (assume this for now). Then, the parity information $p_i[k]$ for this set of storage units is computed as $p_i[k] = su_i[k] \oplus su_i[k+1] \oplus \ldots \oplus su_i[k+g-2]$. The set composed of the parity storage unit $p[k]$ and of the $g - 1$ storage units from $su_i[k]$ to $su_i[k+g-2]$ is called a *parity group* of size g. If the disk that contains $su_i[k+1]$ is lost, this storage unit can be rebuilt by the following computation $su_i[k+1] = su_i[k] \oplus p_i[k] \oplus \ldots \oplus su_i[k+g-2]$. Thus, the disk with the parity information takes the place of the disk which was lost.

The idea of fault tolerance with full replication is to use additional space which is of the same size of the space occupied by the whole set of data blocks. Thus, all data blocks are duplicated. While more expensive in terms of space, this approach allows recovering from some types of catastrophic failures and improving the performance of the system. Gains in performance are possible because any request for a data block can now be served by two different disks and thus, we can always select the disk with a smaller queue.

Full replication can also be useful in situations where a parity-based scheme is not the most appropriate one. For instance, consider a distributed server composed of multiple machines which contain themselves multiple disks. Assume that we stripe the data across the multiple disks. In case a parity-based scheme is adopted for fault tolerance, parity groups should be confined to individual machines to avoid overheads in buffer, networking, and synchronization. This provides tolerance to a disk failure but not to the failure of a machine. To provide tolerance to a machine failure, a full replication scheme can be adopted instead in which each data block and its copy reside on separate machines. This is in fact the approach adopted in [9].

It can be shown that a random layout allows using a parity-based scheme as well as any random layout. Further, full replication can be better taken advantage of with the adoption of a random layout (instead of a striped layout). In fact, the design and implementation of recovery and load balancing algorithms is simplified because one can rely on the randomness of the data block allocation to even out the load.

3 Transmitting Information

There are several performance issues that need to be addressed in order to transmit continuous real-time streams over the Internet with acceptable quality. For instance, real time video encoded in MPEG2 typically requires an average bandwidth of approximately 1-4 Mbps, and a voice stream approximately from 6-64Kbps, depending on the encoding scheme. However, so far the Internet does not allow bandwidth reservation as needed. In addition congestion in the network may cause significant variability on the interval between the arrival of successive packets (jitter). Since real time streams must be decoded and played following strict time constraints, large jitter values will cause the playout process to be interrupted. Packet losses may also severely degrade the quality of the multimedia presentation, depending on the loss pattern. Yet another problem is network heterogeneity and client heterogeneity. Client heterogeneity means that the receivers have different network requirements, due to different capabilities to present the received multimedia information. For multicast applications, the heterogeneity imposes an additional challenge since a stream being transmitted would have to be multicast through several networks and clients (with possibly drastic different characteristics) and somehow adapt to the needs of each client. In this section we discuss a few mechanisms used to mitigate the effects of random delay and losses in the network.

3.1 How to Cope with Network Jitter and the Rate Variability

We start by considering an audio stream encoded with PCM, say with silence detection. The audio stream is sampled at $125\mu sec$ interval and usually 160 samples are collected in a single packet generating a CBR stream of one IP packet per $20msec$ [47] at each active interval. The client consumes the 160 samples at every $20msec$, and thus it is vulnerable to random delays in the network. If the expected information does not arrive on time, annoying distortions may occur in the decoded audio signal. Let T be the packet generation interval and X the corresponding segment interarrival time. The random variable $J = X - T$ is called *jitter*.

One simple mechanism to reduce the jitter is to use a *playout buffer* at the client, where a given number of packets are stored. At the beginning of each *active* period, where packets are generated, the client stores packets till a given threshold is reached before starting to decode the received samples. The threshold value may be fixed at the beginning of the connection or be adjusted dynamically during the duration of the session. Figure 2 illustrates the basic idea. In

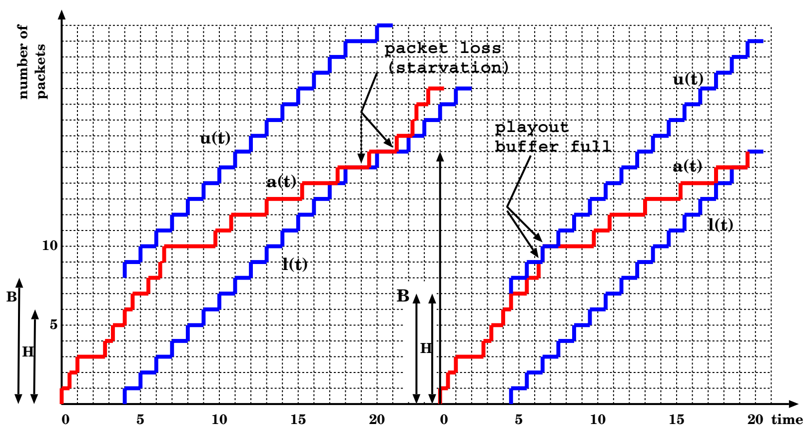

Fig. 2. The playout buffer

that figure, the curve $l(t)$ is equal to the number of packets consumed by the application by time t. (In this example, it is assumed PCM encoding and thus the packet consumption rate is constant.) The upper curve is simply $u(t) = l(t) + B$ where B is the playout buffer space. The curve labeled $a(t)$ is equal to the number of packets that have arrived by time t. Note that the arrival instants are not equally spaced due to the jitter introduced by the network. In the leftmost part of Fig. 2, $B = 8$ and $H = 6$, and so the decoding starts immediately after the arrival of the 6-th packet. The amount of packets stored in the playout buffer as a function of time is $a(t) - l(t)$, while $u(t) - a(t)$ quantifies the buffer space available at t. Buffer starvation occurs if the lower curve touches the bottom curve and buffer overflow occurs when the middle curve crosses the top curve. As shown in the figure, the buffer empties at $t = 18$. Thus, at $t = 19$ there is no packet to be decoded (buffer starvation). When this occurs, some action must be taken perhaps re-playing the last information in the buffer, as an approximation of the data carried by the missing packet at that time. In the right hand part of Fig. 2 the threshold value H is increased to $H = 7$. As a consequence, $l(t)$

is shifted to the right. In this example, this change prevents buffer starvation during the observation period. The value of B is also decreased to 7, and $u(t)$ is moved downwards with respect to the preceding curve.

It is easy to see that this simple technique eliminates any negative jitter. From Fig. 2, it is also clear that larger threshold values decrease the jitter variability. However, latency increases with increasing threshold values. But interactive applications, such as a live conversation, do not tolerate latencies larger than $200 - 300$ msec. This imposes a constrain of 20-25 packets on H. An issue is the choice of H and the amount of buffer space necessary to minimize the loss of packets in case a long burst of packets arrive at the receiver.

Diniz and de Souza e Silva [22] calculate the distribution of the jitter as seen by the client, when a playout buffer is used. The packet interarrival time is modeled by a phase-type distribution that matches the first and second moments of this measure obtained from actual network measurements. Packets are consumed at constant rate (PCM), similar to the example of Fig. 2. Silent periods are included in the model. The goal is to study the tradeoffs between latency and probability of a positive jitter. It was concluded that the probability of a positive jitter can be significantly reduced, while maintaining an acceptable latency for real time traffic.

In addition to the delay variability imposed by the network, compressed audio/video streams exhibit non-negligible burstiness on several time scales, due to the encoding schemes and scene variations. Sharp variations on traffic rates have a negative impact on resource utilization. For instance, more bandwidth may be necessary to maintain the necessary QoS for the application. The issue is to develop control algorithms to smooth the CM traffic before transmission to the clients.

Smoothing techniques can be applied at the traffic source or at another intermediate node (e.g., a proxy) in the path to the client. Sen *et al* [65] address the issue of *online bandwidth smoothing*. To better understand the problem consider Fig. 3, where it is assumed that there is no variable delay imposed by the network when a compressed video stream is sent to a client.

Due to the compression encoding, the rate of bit consumption at the client node varies with time. Video servers however, read fixed size blocks of information from the storage server (each block may be fragmented into packets to fit the network maximum transfer unit (MTU) before transmission). Then in Fig. 3, the interval between the consumption of constant size data blocks at the client varies with time. The jumps in the y-axis (a block of data) are of constant size. This contrasts with the usual representation of variable frame size consumed at fixed intervals of time (e.g. $1/30sec$).

In Fig. 3, a controller at the server site schedules the transmission of video blocks after they are retrieved from the storage server and queued in a FIFO buffer. Two sets of curves are shown in the figure. In set 1, let $l_c(t)$ be the number of bits consumed by the client by time t, and $u_c(t) = l_c(t) + B_c$, where B_c is the size of the playout buffer. Similarly, let $a_s(t)$ (in set 2) be the accumulated number of bits that are read from the server disks during $(0, t)$ according to the

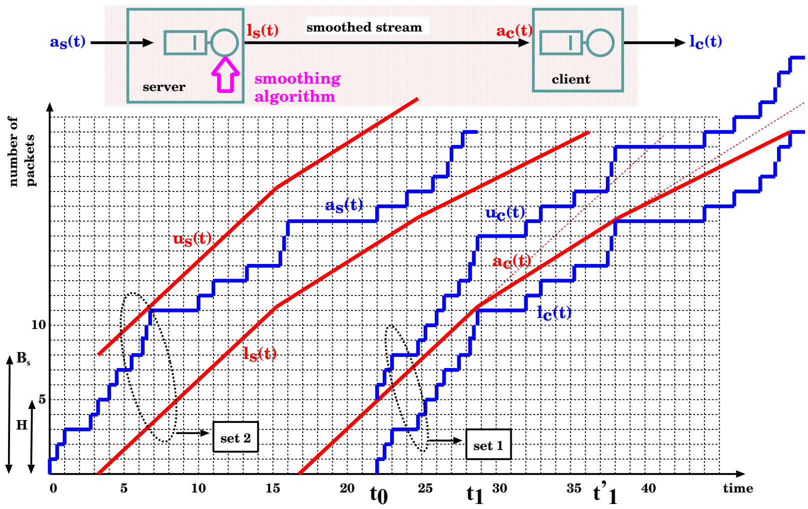

Fig. 3. The concept of smoothing

demand of the client, and $l_s(t)$ be the smoothed stream curve, i.e. the number of bits effectively dispatched by t. Note that: (a) $a_c(t) = l_s(t - \tau)$ where τ is a (assumed constant) network delay from the server to the client; (b) $a_s(t - \tau') = l_c(t)$ where $\tau\prime$ is the constant network delay plus the delay to fill the playout buffer; (c) the jumps of $l_c(t)$ occur at the instants of consumption of a block of data. If we assume that the playout buffer is filled until its capacity before the continuous stream is played back, the number of bits in the playout buffer is given by the difference between the top and the middle curves in set 1.

The server seeks to transmit data to the client as smooth as possible that is, $l_s(t)$ in the figure should resemble a straight line with the smallest possible right angle. Since the shape of $l_c(t)$ and consequently $u_c(t)$ and $a_s(t)$ is determined by the encoding algorithm applied to the video to be transmitted, the issue is how to plan the transmission of the data so that $u_c(t) \leq a_c(t) \leq l_c(t)$, and yet the maximum transmission rate is kept as close as possible to the average consumption rate.

In [62] Salehi *et al* obtained an efficient algorithm that can generate a transmission schedule given the complete knowledge of $a_c(t)$. This is referred to as an *offline* algorithm. Roughly, from a initial time t_i (start from $i = 0$), one should construct the longest possible line that does not violate the constraints imposed by $u_c(t)$ and $l_c(t)$ in Fig. 3. Clearly, by construction, this straight line intersects one of the boundary curves at a time point $t'_{i+1} > t_i$, (and so the rate would have to be changed at this point), and touches one of these curves at a time $t_{i+1} < t'_{i+1}$. To avoid sudden rate changes one should vary the previous rate as soon as possible. Consequently a new starting point is chosen at t_{i+1}. The process is repeated (setting $i = i + 1$) until the end of the stream is reached, and $a_c(t)$ is obtained which determines the scheduling algorithm.

The set 2 in Fig. 3 shows the arrival and transmission curves at the server. Note that the server starts its transmission as soon as a threshold H is reached (in the figure the threshold is equal to 5 blocks). $u_s(t) = l_s(t) + B_s$, where B_s is the FIFO buffer available at the server. One should note that, once $l_s(t)$ is determined, B_s and the threshold can be calculated to avoid overflow and underflow.

We can represent the curves $u_c(t)$, $l_c(t)$ and $a_c(t)$ as vectors $\mathbf{u}$, $\mathbf{l}$, $\mathbf{a}$ respectively, each with dimension N, where N is the number of data blocks in the video stream, and the i-th entry in one of the vectors, say vector $\mathbf{l}$, is the amount of time to consume the i-th block. In [62] majorization is used as a measure of smoothness of the curves. Roughly, if a vector $\mathbf{x}$ is majorized by $\mathbf{y}$ ($\mathbf{x} \prec \mathbf{y}$) then $\mathbf{x}$ represents a smoother curve than $\mathbf{y}$. It is shown in [62] that if $\mathbf{x} \prec \mathbf{y}$ then $var(\mathbf{x}) = \sum_i (x_i - \overline{x})^2$ and since, by definition, the maximum entry in $\mathbf{x}$ is less or equal than the corresponding entry in $\mathbf{y}$, a vector $\mathbf{x}$ that is majorized by $\mathbf{y}$ has smaller maximum rate and smaller variance than $\mathbf{y}$. The schedule algorithm outlined above is shown to be optimum and unique in [62]. Furthermore the optimal schedule minimizes the effective bandwidth requirements.

This algorithm is the basis for the *online* smoothing problem [65]. It is assumed that, at any time τ, the server has the knowledge of the time to consume each of the next P blocks. This is called the *lookahead interval* and is used to compute the optimum smoothing schedule using the algorithm of [62]. Roughly, blocks that are read from the storage server are delayed by w units and passed to the server buffer that implements the smoothing algorithm which is invoked every $1 \leq \alpha \leq w$ blocks.

Further work in the topic include [50] where it is shown that, given a buffer space of size B at the server queue, a maximum delay jitter J can be achieved by an off-line algorithm (similar to the above algorithm). Furthermore, an online algorithm can achieve a jitter J using buffer space $2B$ at the server FIFO queue. While the smoothing techniques described above are suitable for video applications, interactive visualization applications pose additional problems. This subject was studied in [73].

In summary, the amount of buffer space in the playout buffer of Fig. 3 is a key parameter that determines how smooth the transmission of a CM stream can be. It also serves to reduce the delay variability introduced by the network. The queue at the server site in Fig. 3 implements the smoothing algorithm and the amount of buffer used is also a design parameter. As mentioned above, jitter reduction and smoothness are achieved at expense of the amount of buffer space. But the larger these buffers the larger the latency to start playing the stream.

3.2 How to Cope with Losses

In the previous subsection we are concerned with random delays introduced by the network as well as the ability to reduce sudden rate chains in the coded data stream to lower the demand for network resources. Besides random delays, the Internet may drop packets mainly due to congestion at the routers. Furthermore, packet delays may be so large that they may arrive too late to be played at the receiver, in a real time application.

A common method to recover from a packet loss is retransmission. However, a retransmitted packet will arrive at the receiver at least one round-trip-time (RTT) later than the original copy and this delay may be unacceptable for real time applications. Therefore, retransmission strategies are only useful if the RTT between the client and the server is very small compared with amount of time to empty the playout buffer.

A number of retransmission strategies have been proposed in the context of multicast streaming. For example, receiver-initiated recovery schemes may obtain the lost data from neighboring nodes which potentially have short RTTs with respect to the node that requested the packet [69].

The other methods to cope with packet loss in CM applications are *error concealment*, *error resilience*, interleaving and *FEC* (forward error correction). Error resilience and error concealment are techniques closely coupled with the compression scheme used. Briefly error resilience schemes attempt to limit the error propagation due to the loss, for instance via re-synchronization. An error propagation occurs when the decoder needs the information contained in one frame to decode other frames. Error concealment techniques attempt to reconstruct the signal from the available information when part of it is lost. This is possible if the signal exhibits short term self-similarities. For instance, in voice applications the decoder could simply re-play the last packet received when the current packet is not available. Another approach is to interpolate neighboring signal values. Several other error concealment techniques exist (see [57,74] for more details and references on the subject).

Interleaving is an useful technique for reducing the effect of loss bursts. The basic idea is to separate adjacent packets of the original stream by a given distance, and re-organize the sequence at the receiver. This scheme introduces no redundancy (and so does not consume extra bandwidth), but introduces latency to re-order the packets at the receiver.

FEC techniques add sufficient redundancy in the CM stream so that the received bit stream can be reconstructed at the receiver even when packet losses occur. The main advantage of FEC schemes is the small delay to recover from losses in comparison with recovery using retransmission. However, this advantage comes at the expense of increasing the transmission rate. The issue is to develop FEC techniques that can recover most of the *common* patterns of losses in the path from the sender to the receiver without much increase in the bandwidth requirements to transmit the continuous stream.

A number of FEC techniques have been proposed in the literature [57,74]. The simplest approach is as follows. The stream of packets is divided into groups of size $N-1$, and a XOR operation is performed on the $N-1$ packets of each group. The resulting "parity packet" is transmitted after each group. Clearly, if a single packet is lost in a group of N packets the loss can be recovered.

Three issues are evident from this simple scheme. First, since a new packet is generated for every $N-1$ packets the bandwidth requirements increases by a factor of $1/(N-1)$. As an example, considering voice PCM transmission, if $N = 5$ the new necessary bandwidth to transmit the stream would be 80 Kbps instead of 64 Kbps. Second, it is necessary to characterize the transmission losses to evaluate how effective is the FEC scheme. If losses come into bursts of length $B > 1$, then this simple scheme is evidently not effective. Third, a block of N packets must be entirely received in order to recover from a loss. Consequently, N must be smaller than the receiver playout buffer which is in turn limited by the latency that can be tolerated in an interactive application. Furthermore, suppose a loss occurs at position n in a block. Then $N-n$ should be smaller than the number of packets in the playout buffer at the arrival time of the $(n-1)$-th packet in the block. This indicates that the probability that the queue length of playout buffer is smaller than N should be small for the method to be effective. In summary, to avoid a substantial increase in the bandwidth requirements the block size should be larger, but a large block size implies a large playout buffer which in turn increase latency.

Measures in the Internet have shown that loss probabilities are little sensitive to packet sizes [10,29]. Therefore, another scheme to protect against losses is to transmit a sample of the audio stream in multiple packets [12]. For instance, piggybacking the voice sample carried by packet n in packet $n+1$ as well. This technique allows the recovery of single losses with minimum latency, but at the expense of doubling the throughput. A clever way to reduce the bandwidth requirements is to include in packet $n+1$ the original (voice) sample sent in packet n, but further compressed using a smaller bit rate codec than the primary encoding. As an example, suppose the primary (voice) sample of each packet is replicated twice. If we use PCM codec for the primary sample (64k Kbps), GSM for the first copy (13.2 Kbps) and LPC for the second copy (≈ 5 Kpbs) burst sizes of length 2 can be recovered at the expense of increasing the throughput from 64 Kbps to ≈ 82 Kbps. Therefore, with only a 20.8% increase in the throughput, and an increase in latency of 40 msec (2×20 msec packet delay), bursts of length 2 can be recovered in this example.

Packet losses occur mostly due to congestion in the network and so it can be argued that increasing the throughput rate of a stream transmission to recover from losses is unfair with respect to flow controlled sessions such as TCP. Therefore, it is evident that FEC schemes should be optimized for the type of loss incurred by the network in order to have the smallest impact in the consumed bandwidth. The work in [29] was aimed at studying the packet loss process in the Internet and at proposing a new efficient XOR-FEC mechanism extending previous work. Several measures were calculated from traces between Brazil and the USA such as: the distribution of the number of consecutive losses and the distribution of the number of packets received between two losses. These two measures are particularly important for determining the efficiency of a recovery XOR-FEC based algorithm. The approach proposed in [29] can be briefly described as follows. The CM stream is divided into groups each called a *window* and the packets in a window is sub-divided into non-overlapping sets, each protected by an XOR operation. Figure 4(a) illustrates an example with 6 packets

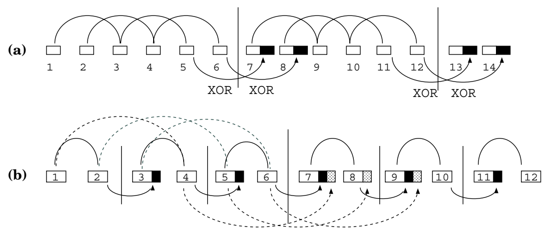

Fig. 4. The FEC scheme of [29]

per window and two subsets, and we call this a 2:6 class of algorithm. The result of the XOR operation for a set of packets is sent piggybacked in a packet of the next window. Clearly we can use codecs of smaller transmission rate as in [12] for saving in bandwidth. Note that burst errors of size at most equal to 2 packets can be recovered, and efficiency in bandwidth is gained at expense of latency to recover from losses. Furthermore, the scheme in Fig. 4(a) has the practically same overhead as the simple XOR scheme first described, but can recover from consecutive losses of size two. Another class of schemes can be obtained by merging two distinct $k : n$ class of algorithms, such that all packets belong to at least two different subsets and therefore are covered by two different XORs. Figure 4(b) illustrates an example where schemes 1:2 and 3:6 are overlapped. In this case, all losses of size one in the larger window can be recovered. Furthermore, complex loss patterns can also be protected. For example, if packets 2, 3, 4, and 5 were lost, they can all be recovered. The overhead of this mixed scheme is clearly obtained by adding the overhead of the individual schemes.

In [29] the algorithm is applied in the real data obtained from measures and the efficiency of different FEC-based algorithms are evaluated. The conclusions show the importance of adapting to different networks conditions. (This issue was also addressed by Bolot *et al* [11], in the context of the scheme of [12].) Furthermore, in all the tests performed, the class of schemes that mixed two windows provided better results than the class with a single window under the same overhead.

Altman *et al* developed an analytical model to analyze the FEC scheme of [12] and other related schemes. The loss process was modeled using a simple M/M/K queue. The aim is to assess the tradeoffs between increased loss protection from the FEC algorithm and the adverse impact from the resulting increase in the network resources usage due to the redundancy added to the original stream. The results show that the scheme studied may not always result in performance gains, in particular if a non-negligible fraction of the flows implements the same FEC scheme. It would be interesting to evaluate the performance of other FEC approaches.

To conclude the subsection we refer to a recent proposed approach to mitigate the effect of losses. As is evident from above, short term loss correlations have an adverse effect on the efficiency of the recovery algorithms. One way to reduce the possible correlations is to split the continuous stream sent from a source to a given destination into distinct paths. For instance, we could split a video stream in two and sent the even packets in the sequence via one path and the odd packets via another path to the destination. This is called *path diversity* [76]. In [6] it is assumed that the loss characteristics in a path can be represented by a 2-state Markov chain (Gilbert model) and a Markov model was developed to access the advantages of the approach. Clearly a number of tradeoffs exists, depending on the loss characteristics of each path, if the different paths share or not a set of links, etc.

3.3 Characterizing the Packet Loss Process and the Continuous Media Traffic Stream

Packet losses is one of the main factors that influence the quality of the signal received. Therefore, understanding and modeling the loss process is imperative to analyze the performance of the loss recovery algorithms. In general, measurements are obtained from losses seen by packet probes sent according to the specific traffic under study (for instance at regular intervals of 20 msec), and models are obtained to match the collected statistics. However, queueing models with finite buffer have also been used.

Bolot [10] characterize the burstiness of packet losses by the conditional probability that a packet is lost given that a previous packet is also lost, and analyze data from probes sent at constant intervals between several paths in the Internet. A simple finite buffer single server queueing model (fed by two streams, one representing the probes and the other representing the Internet traffic) was used as the basis for interpreting the results obtained from the measures.

The most commonly used model for representing error bursts is a 2-state discrete time Markov chain, usually called the Gilbert model. The Gilbert model assumes that the size of consecutive losses is a geometric random variable. However, these models may not capture with accuracy the correlation structure of the loss process. The work in [75] use a 2^k-state Markov chain to model the loss process, aimed at capturing temporal dependencies in the traces they collected. They analyze the accuracy of the models against several traces collected by sending probes at regular intervals. Salamatian and Vaton [61] propose the use of Hidden Markov models (HMM) to model the loss sequence in the Internet, due to their capability to represent dependencies in the observed process. In an HMM each state can output a subset of symbols according to some distribution. The states and transitions between states are not observable, but only the output symbols. In [61] it is shown that HMM models are more appropriate to represent the loss sequence than the 2^k-state Markov chain model used in [75], with less states. However, one disadvantage of the method is the cost of estimating the Markov chain parameters. More recently, the authors of [43] compare four models to represent the loss process in wireless channels, including the 2^k-state Markov model, a HMM with 5 states and a proposed On-Off model where the holding times at each state are characterized by a mixture of geometric phases which are determined by using the Baum-Welch algorithm. The conclusion indicates that the extended On-Off model better captures first and second order statistics of the traces studied. A recent study done by Markopoulou *et al* [51] evaluates the quality of voice in the Internet. In this work a methodology is developed that takes into account delay and loss measurements for assessing the quality of a call. The measures were obtained by sending regularly spaced probes to measurement facilities in different cities. The results indicate the need of carefully evaluating the CM traffic and properly designing the playout buffers.

Traffic characterization is one important topic for understanding the influence of CM streams in the network resources. The topic is related to loss characterization and the main goals are to obtain concise descriptions of the flow under study and to capture in the model relevant statistics of the flow. The objective is to predict, with sufficient accuracy, the impact of the traffic generated by applications on the resources being utilized (both in the network and in the servers), and evaluate the QoS perceived by the applications. The amount of work done on traffic characterization is sufficient vast to deserve surveys and books on the area [31,1,52,56]. Our interest in this chapter is to introduce a few issues on the topic.

In order to build a model of the traffic load, the proper traffic *descriptors* that capture important characteristics of the flows competing for resources have to be chosen. Examples of traffic descriptors are: the mean traffic rate, the peak-to-mean ratio, the autocovariance, the index of dispersion and the Hurst parameter. The issue is to select a set of descriptors such that traces and models with matching descriptors produce similar performance metrics.

A large number of models have been proposed in the literature. They include models in which the autocorrelation function decays exponentially (for instance, the Markovian models), and models in which the autocorrelation function decays at a slower rate, that is, hyperbolically (in this case the corresponding stationary process is called long-range dependent [56]). Although not possessing the long-range dependence property Markov models are attractive due to several reasons. First, they are mathematically tractable. Second, long-range correlations can be approximately obtained from certain kind of models. Third, it may be argued that long-range dependency is not a crucial property for some performance measures and Markov models can be used to accurately predict performance metrics (e.g. see [38,37]). Usually, a traffic model is built (or evaluated) by matching the descriptors calculated from the model against those obtained from measurement data. For Markovian models it is not difficult to calculate first and second order statistics [49].

As can be inferred from the above discussion, Markovian models are a useful tool for the performance evaluation of CM applications. They can be used to model the traffic stream, generate artificial loads, model the loss process and evaluate techniques to efficiently transmit CM streams.

4 Resource Sharing Techniques

Conventional multimedia servers provide each client with a separate stream. As a consequence, the resources available in the multimedia system, particularly network bandwidth, can be quickly exhausted. Consider, for instance, the scenarios 1 and 2 discussed in Sec. 2. To maintain 1500 concurrent streams live, with a separate bandwidth allocated to each of them, it is necessary to sustain a total bandwidth of 2,25 Gbps at the output channel of the multimedia server. While technically feasible, this is quite expensive nowadays and prohibitive from a commercial point of view.

A common approach for dealing with this problem is to allow several clients to share a common stream. This is accomplished through mechanisms, here called *resource sharing techniques*, which allow the clients to share streams and buffers. The goal is to reduce the demand for network bandwidth, disk bandwidth, and storage space. While providing the capability of stream sharing, these techniques have also to provide QoS to the clients.

As is Sec. 3 client QoS is affected by the server characteristics, such as latency and available disk bandwidth, and by the network characteristics, such as bandwidth, loss, jitter, and end-to-end delay. The challenge is to provide very short startup latency and jitter for all client requests and to be able to serve a large number of users at minimum costs.

The bandwidth sharing mechanisms proposed in the literature fall into two categories: *client request oriented* and *periodic broadcast*. *Client request oriented* techniques are based on the transmission, by the server, of a CM stream in response to multiple requests for its data blocks from the client. *Periodic broadcast* techniques are based on the periodic transmission by the server of the data blocks.

Besides these sharing mechanisms, one can also use proxy servers to reduce network load. Proxy servers are an orthogonal technique to bandwidth sharing protocols, but one which is quite popular because it can be easily implemented and can be managed at low costs.

We partition our presentation in three topics. We first describe *client request oriented* techniques. Then, *periodic broadcast* mechanisms are presented, followed by a discussion on proxy-based strategies.

4.1 Client Request Oriented Techniques

The simplest approach for allowing the sharing of bandwidth is to batch new clients together whenever possible. This is called *batching* [2,20,21] and works as follows. Upon the request of a new media stream s_i by an arriving client c_k, a batching window is initiated. Every new client that arrives within the bounds of this window and requests the stream s_i is inserted in a waiting queue i.e., it is batched together with the client c_k. When the window expires, a single transmission for the media stream s_i is initiated. This transmission is shared by all clients, as in standard broadcast television. Batching policies reduce bandwidth requirements at the expense of introducing an additional delay to the users (i.e., client startup latency increases).

Stream tapping [15], patching [13,40], and controlled multicast [32] were introduced to avoid the latency problems of batching. They are very similar techniques. They can provide immediate service to the clients, while allowing clients arriving at different instants to share a common stream.

In the basic patching scheme [40], the server maintains a queue with all pending requests. Whenever a server channel becomes available, the server admits all the clients that requested a given video at once. These clients compose a new batch. Assume, that this new batch of clients requested a CM stream s_i which is already being served. Then, all clients in this new batch immediately join this on-going multicast transmission of s_i and start buffering the arriving data. To obtain the initial part of the stream s_i, which is called a *patch* because it is no longer being multicasted, a new channel is opened with the multimedia server. Data arriving through this secondary channel is immediately displayed. Once the initial part of s_i (i.e., the patch) has been displayed, the client starts consuming data from its internal buffer. Thus, in this approach, the clients are responsible for maintaining enough buffer space to allow merging the patch portion of the stream with its main part. They also have to be able to receive data in two channels.

Stream tapping, optimal patching [13], and controlled multicast differ from the basic patching scheme in the following way: they define an optimal patching window w_i for each CM stream s_i. This window is the minimum interval between the initial instants of two successive complete transmissions of the same stream s_i. The size of w_i can improve the performance of patching. If w_i is set too large, most of the server channels are used to send patches. On the other hand, if w_i is too small no stream merging will occur. The patching window size is optimal if it minimizes the requirements of server and network bandwidth. The algorithm works as follows. Clients which requested the stream s_i prefetch data from an on-going multicast transmission, if they arrive within w_i units of time from the beginning of the previous complete transmission. Otherwise, a new multicast transmission of stream s_i is initiated. A mathematical model which captures the relation between the patching window size and the required server bandwidth is proposed in [13]. In [32] an expression for the optimal patching window is obtained.

Figure 5 illustrates batching and patching techniques. In Fig. 5(a), three new clients requesting the stream s_i arrive within the batching window. They are served by the same multicast transmission of s_i. Figure 5(b) shows the patching mechanism. We assume that the three requests arrive within a patching window. The request r_0 triggers the initial multicast transmission of stream s_i, r_1 triggers the transmission of the patch interval $(t_1 - t_0)$ of s_i for r_1, and r_2 starts a transmission of the $(t_2 - t_0)$ missing interval of s_i for r_2.

A study of the bandwidth required by optimal patching, stream tapping, and controlled multicast is presented in [27]. It is assumed that the arrivals of client requests are Poisson with mean rate equal to λ_i for stream s_i. The required server bandwidth for delivery of stream s_i is given by [27] as: $B_{\mathrm{OP,ST,CM}} = (1 + w_i^2 N_i/2)/(w_i + 1/N_i)$, where $N_i = \lambda_i T_i$, T_i is the total length of stream s_i and w_i is the patching window.

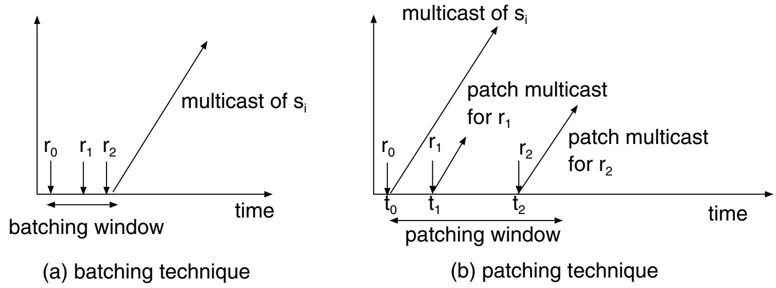

Fig. 5. Batching and patching techniques.

The expression presented above is very similar to the results obtained in [13, 32]. The value of the optimal patching window can be obtained differentiating the expression for $B_{\mathrm{OP,ST,CM}}$. It is equal to $(\sqrt{2N_i+1}-1)/N_i$. The server bandwidth for an optimal patching window is given by [32]: $B_{\mathrm{optimal\_window}} = \sqrt{2N_i+1}-1$.

A second approach to reduce server and network bandwidth requirements, called *Piggybacking*, was introduced in [4,35,48]. The idea is to change dinamically the display rates of on-going stream transmissions to allow one stream to catch up and merge with the other. Suppose that stream s_i is currently being transmitted to a client. If a new request for s_i arrives, then a new transmission of s_i is started. At this point in time, the server slows down the data rate of the first transmission and speeds up the data rate of the second transmission of s_i. As soon as the two transmissions become identical, they can be merged and one of the two channels can be released. One limitation of this technique is that it requires a specialized hardware to support the change of the channel speed dinamically.

In the hierarchical stream merging (HSM) techniques [7,25,27] clients that request the same stream are hierarchically merged into groups. The client receives simultaneously two streams: the one triggered by its own request and a second stream which was initiated by an earlier request from a client. With time, the client is able to join the latter on-going multicast transmission and the delivery of the stream initiated by it can be aborted. The merged clients also start listening on the next most recently initiated stream. Figure 6 shows a scenario where four client requests arrive for stream s_1 during the interval (t_1, t_3). The server initiates a new multicast transmission of s_1 for each new client. At time t_2, client c_2, who is listening to the stream for c_1, can join this on-going multicast transmission. At time t_3, client c_4 joins the transmission of c_3. Thus, after t_3, client c_4 can listen to the multicast transmission initiated at t_1. At t_5, client c_4 will be able to join the transmission started for client c_1. Bandwidth skimming (BS) [26] is similar to HSM. In this technique, policies are defined to reduce the user bandwidth requirements to less than twice the stream playback rate.

In [27], an expression for the required bandwidth of a server operating with the HSM and BS techniques was obtained. Suppose that the transmission rate needed by a client is equal to b units of the media playback rate ($b = 2$ for HSM and $b < 2$ for bandwidth skimming) and that the request arrivals are Poisson with mean rate equal to λ_i for stream s_i. The required server bandwidth for delivery of stream s_i can be approximated by: $B_{\text{HSM,BS}} \approx \eta_b ln(N_i/\eta_b + 1)$, where N_i is defined as above and η_b is the positive real constant that satisfies: $\eta_b[1 - (\eta_b/(\eta_b + 1))^b] = 1$

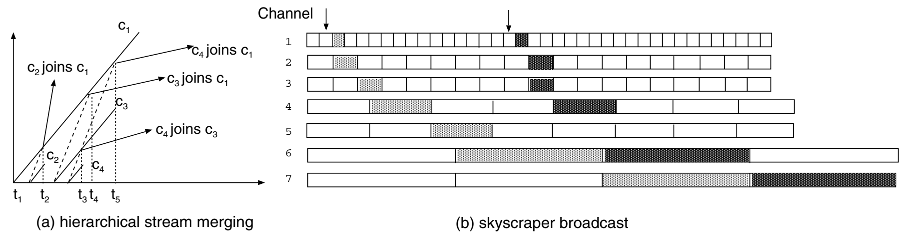

Fig. 6. Hierarchical stream merging and skyscraper broadcasting

Most of the studies in the literature have evaluated the required server bandwidth for the proposed sharing techniques. One key question is how the performance of a multimedia system is affected when the server bandwidth is limited by the equations previously presented. In a recent work [68] analytical models for HSM, BS and patching techniques are proposed to evaluate two performance metrics of a multimedia system: the mean time a client request is delayed if the server is overloaded (it is called the mean client waiting time) and the fraction of clients who renege if they are not served with low delay (it is called the *balking rate*). The models assumptions are: client request arrivals are Poisson, each client requests the entire media stream and all media streams have the same length and require the same playout rate. The model proposed to evaluate the balking rate is a closed two-center queueing network with C clients (C is the server bandwidth capacity in number of channels). The mean client waiting time is obtained from a two-center queueing network model with K users (There are one user per class and each class represents one stream.) Each user models the *first request for a stream s_i*, the others requests that batch with the *first* are not represented in the model. Results obtained from the analytical models show that the client balking rate may be high and the mean client waiting time is low when the required server bandwidth is defined as in the equations presented above. Furthermore the two performance parameters are very sensitive to the increase in the client load.

4.2 Periodic Broadcast Techniques

The idea behind periodic broadcast techniques is that each stream is divided into segments that can then be simultaneously broadcast periodically on a set of k

different channels. A channel c_1 delivers only the first segment of a given stream and the other $(k-1)$ channels deliver the remainder of the stream. When a client wants to watch a video, he must wait for the beginning of the first segment on channel c_1. A client has a schedule for tuning into each of the $(k-1)$ channels to receive the remaining segments of the video. The broadcasting schemes can be classified into three categories [39]. The first group of periodic broadcast techniques divides the stream into increasing sized segments and transmits them in channels of the same bandwidth. Smaller segments are broadcast more frequently than larger segments and the segments follow a size progression $(l_1, l_2, ..., l_n)$. In the Pyramid Broadcasting (PB) protocol [70], the sizes of the segments follow a geometric distribution and one channel is used to transmit different streams. The transmission rate of the channels is high enough to provide on time delivery of the stream. Thus, client bandwidth and storage requirements are also high.

To address the problem of high resource requirements at the client side, a technique called Permutation-based Pyramid Broadcasting (PPB) was proposed in [3]. The idea is to multiplex a channel into k subchannels of lower rate. In the Skyscraper broadcast technique [41] each segment is continuously transmitted at the video playback rate on one channel as shown in Fig. 6. The series of segments sizes is 1,2,2,5,5,12,12,25,25,52,52,... with a largest segment size equal to W. Figure 6 shows two client request arrival times: one just prior the third segment and the other before the 18^{th} segment broadcast on channel 1. The transmission schedules of both clients are represented by the gray shaded segments. The schedule is such that a client is able to continuous playout the stream receiving data in no more than two channels. The required maximum client buffer space is equal to the largest segment size.

For all techniques described above the required server bandwidth is equal to the number of channels and is independent of the client request arrival rate. Therefore, these periodic broadcast techniques are very bandwidth efficient when the client request arrival rate is high. A dynamic skyscraper technique was proposed in [24] to improve the performance of the skyscraper. It considered the dynamic popularity of the videos and assumed lower client arrival rates. It dynamically changes the video that is broadcast on the channels. A set of segments of a video are delivered in transmission clusters. Each cluster starts every W slots on channel 1 and broadcasts a different video according to the client requests. Client requests are scheduled to the next available transmission cluster using a FIFO discipline, if there is no transmission cluster already been assigned to the required video. A new segment size progression is proposed in [27] to provide immediate service to client. Server bandwidth requirements for transmitting a stream s_i considering Poisson arrivals with mean rate λ_i are given by [27]: $B_{Dyn\_Sky} = 2U\lambda_i + (K-2)/(1 + 1/\lambda_i WU)$, where U is the duration of a unit-segment, W is the largest segment size and K is the number of segments in the segment size progression.

Another group, the harmonic broadcast techniques, divide the video in equal sized segments and transmit them into channels of decreasing bandwidth. The third group combines the approaches described above. They are a hybrid scheme of pyramid and harmonic broadcasting.

4.3 Proxy Based Strategies

The use of proxies in the context of CM applications has several advantages. Server and network bandwidth requirements can be reduced and the client startup latency can be very low. In Sec. 4.1 and 4.2 the models to evaluate the scalability of the bandwidth sharing techniques are based on the assumption that client request arrivals are sequential (i.e., clients request a stream and playout it from the beginning to the end). The required server bandwidth for these techniques varies logarithmically with the client request arrivals (for stream merging) and logarithmically with the inverse of the start-up delay (for periodic broadcasting) [44]. In a recent work [44], tight lower bounds on the required server bandwidth for multicast delivery techniques when the client request arrivals are not sequential were derived. The results obtained suggested that for non-sequential access the scalability of these techniques is not so high as for sequential access. Thus, the use of proxies is a complementary strategy that can reduce resource requirements and client latency of large scale CM applications.

Provisioning a multimedia application with proxy servers involves determining which content should be stored at each proxy. Several studies are based on the storage of data accessed most frequently. Distinct approaches exist in the literature. One idea is to divide the compressed video in layers that can be cached at the proxy. An alternative is to cache a portion of a video file at the proxy. Recent work combines the use of proxies with bandwidth sharing mechanisms such as periodic broadcast and client request oriented techniques. In the last approach, not only the popularity of the video should be considered to decide which portion of the stream have to be stored at the proxy. The data staged at the proxy depends also on the mechanisms used to share the transmission of data. In most of the bandwidth sharing mechanisms, the server delivers the initial portion of a stream more frequently than the latter part. Therefore, the storage of a prefix can reduce more significantly the transmission costs than the storage of the suffix.

Caching of Video Layers. In a video layer encoding technique, the compressed video stream is divided into layers: a base layer and enhancement layers. The base layer contains essential low quality encoding information, while the enhancement layers provide optional information that can be used to improve the video stream quality.

The approach used in [72] is to divide a video stream in two parts and to store the bursts of the stream in a proxy. A *cut-off rate* C_{rate} is defined, where $0 \leq C_{rate} \leq P_{rate}$ (P_{rate} is the peak rate). The first part of the stream (the *upper part*) exceeds the *cut-off rate*, and the remaider of the stream is the *lower part.* The upper part is staged at a proxy server and the lower part is retrieved from the server. The stream transmitted from the server to the clients approaches to a CBR stream as C_{rate} decreases. Two heuristic algorithms are presented to determine which video and what percentage of it has to be cached at the proxy. The first stores hot videos i.e., popular videos, entirely at the proxy. The second stores a portion of a video so as to minimize the bandwidth requirements on the server-proxy path. Results shown that the second heuristic performs better than the first.

Another approach is presented in [59]. A mechanism for caching video layers is used in conjunction with a congestion control and a quality adaptation mechanism. The number of video layers cached at the proxy is based on the popularity of the video. The more popular is a video, the more layers are stored in the proxy. Enhancement layers of cached streams are added according to a quality adaptation mechanism [60]. One limitation of this approach is that it requires the implementation of a congestion control and of a quality adaptation mechanism in all the transmissions between clients and proxies and between proxies and servers.

Partial Caching of the Video File. The scheme proposed in [66] is based on the storage of the initial frames of a CM stream in a proxy cache. It is called proxy prefix caching. It was motivated by the observation that the performance of CM applications can be poor due to the delay, throughput and loss characteristics of the Internet. As presented in Sec. 3 the use of buffers can reduce network bandwidth requirements and allow the application to tolerate larger variations in the network delay. However, the buffer size is limited by the maximum startup latency a client can tolerate. Proxy prefix caching allows reducing client startup latency, specially when buffering techniques are used. The scheme work as follows. When a client requests a stream, the proxy immediately delivers the prefix to the client and asks the server to initiate the transmission of the remaining frames of the stream. The proxy uses two buffers during the transmission of stream s_i: the prefix buffer B_p and a temporary buffer B_t. Initially, frames are delivered from the B_p buffer while frames coming from the server are stored in the B_t buffer.

Caching with Bandwidth Sharing Techniques. When using a proxy server in conjunction with scalable delivery mechanisms several issues have to be addressed. The data to be stored at each proxy depends on the relative cost of streaming a video from the server and from the proxy, the number of proxies, the client arrival rate and the path from the server to the proxy (unicast or multicast enabled).

Most of the studies in the literature [23,58,16,36,71,5] define a system cost function which depends on the fraction of the stream stored at the proxy (w_i), the bandwidth required for a stream (b_i), the client arrival rate (λ_i) and the length of the stream (T_i). The cost for delivering a stream s_i is given by $C_i(w_i, b_i, \lambda_i, T_i) = B_{server}(w_i, b_i, \lambda_i, T_i) + B_{proxy}(w_i, b_i, \lambda_i, T_i)$ where B_{server} is the cost of the server-proxy required bandwidth and B_{proxy} is the cost of the proxy-client required bandwidth. Then, an optimization problem is formulated. The goal is to minimize the transmission costs subject to bounds on the total storage and/or bandwidth available at the proxy. The solution of the problem gives the proxy cache allocation that minimizes the aggregate transmission cost.

The work of [71] combines proxy prefix caching with client request oriented techniques for video delivery between the proxy and the client. It is assumed that the transmission between the server and the proxy is unicast and the network paths from the proxy to the clients are either multicast/broadcast or unicast. Two scenarios are evaluated: (a) the proxy-client path is unicast and (b) the proxy-client path is multicast. For the scenario (a) two transmission strategies are proposed. In the first a batching technique is used to group the client request arrivals within a window w_{p_i} (equal to the length of the prefix stored at the proxy). Each group of clients is served from the same unicast transmission from the server to the proxy. The second is an improvement of the first. It is similar to the patching technique used in the context of unicast. If a client request arrives at time t after the end of w_{p_i}, the proxy schedules a patch for the transmission of the missing part from the server. The $(T_i - t)$ (T_i is the length of the stream s_i) remaining frames of the stream are delivered from the on-going transmission of stream s_i. The client will receive data from at most two channels: the patch channel and the on-going transmission channel. For the scenario (b), two transmission schemes are presented: the multicast patching technique [13] implemented at the proxy and the multicast merging which is similar to the stream merging technique [25]. A dynamic programming algorithm is used to solve the optimization problem. Results show that the transmission costs when a prefix cache is used are lower compared to caching the entire stream, and that significant transmissions savings can be obtained with a small proxy size.

The work in [36] studies the use of proxy prefix caching with periodic broadcast techniques. The authors propose the use of patching to deliver the prefix from the proxy to the client and periodic broadcast to deliver the remaining frames (the suffix) from the server to the client. Clients will temporarily receive both the prefix from the proxy and the suffix from the server. Therefore, the number of channels a client needs is the sum of the channels to obtain the suffix and the prefix. A slight modification in periodic broadcast and patching is introduced such that the maximum number of simultaneous channels required by a client is equal to two. Proxy buffer allocation is based on a three steps algorithm aimed at minimizing server bandwidth in the path from the server to the proxy. Results show that the optimal buffer allocation algorithm outperforms a scheme where the proxy buffer is evenly divided among the streams without considering the length of each stream.

In [5] the following scenarios are considered: (a) the bandwidth skimming protocol is used in the server-proxy and proxy-client paths, (b) the server-proxy path is unicast capable and the bandwidth skimming technique is used in the proxy-client path, and (c) scenarios (a) and (b) combined to proxy prefix caching. In the scenario (a) the proxy can store an arbitrary fraction of each stream s_i. Streams are merged at the proxy and at the server using the closest target bandwidth skimming protocol [25]. In the scenario (b) the server-proxy path is unicast, thus only streams requested from the same proxy can be merged at the server. Several results are obtained from a large set of system configuration parameters. They show that the use of proxy servers is cost effective in the following cases: the server-proxy path is not multicast enabled or the client arrival rate is low or the cost to deliver a stream from the proxy to the client is very small when compared to the cost to deliver a stream from the server to the client.

5 Conclusions

In this chapter we have surveyed several performance issues related to the design of real time voice and video applications (such as voice transmission tools and multimedia video servers). These include issues from continuous media retrieval to transmission. Since the topic is too broad to be covered in one chapter we trade deepness of exposition to broadness, in order to cover a wide range of inter-related problems.

As can be seen in the material covered, an important aspect in the design of multimedia servers is the storage strategy. We favor the use of the random I/O technique due to its simplicity of implementation and comparable performance with respect to other schemes. This technique is particularly attractive when different types of data are placed in the server, for instance mixture of voice, video, transparencies, photos, etc. Furthermore, the same technique can be easily employed in proxies. To evaluate the performance of the technique, queueing models constructed from real traffic streams traces can be used. It is clear the importance of accurate traffic models to feed the overall server model.

A multimedia server should try to send the requested streams as smooth as possible (or as close as possible to CBR traffic) to minimize the impact of sudden rate changes in the network resource. Large buffers at the receiver imply better smoothing, but at the expense of increasing latency to start displaying a stream. The receiver playout buffer is also used to reduce the packet delay variability imposed by the network and to help in the recovery process when a packet loss occur. We have surveyed a few packet recovery techniques, and presented the main tradeoffs such as error correction capability and increase in the transmission rate, efficiency versus latency, etc. Modeling the loss process is an important problem and many issues remain open. Although some of the conclusions in the chapter were drawn based on the study of voice traffic the issues are not different for video traffic.

Due to the high speed of modern disk systems, presently the bottleneck to delivery the continuous media stream to clients is mainly at the local network where the server is attached, and not at the storage server. Therefore, an issue that has drawn attention in recent years is the development of algorithms to conserve bandwidth, when a large number of clients submit requests to the server. Since multicast is still far from been widely deployed, we favor schemes that use unicast transmission from the storage server to proxy servers. Between the proxy and the clients multicast is more likely to be feasible, and therefore multicast-based techniques to reduce bandwidth requirements are most likely to be useful in the path from the proxy to the clients.

To conclude, we stress the importance of developing multimedia applications and perform tests on prototypes, collect statistics, develop models based on the data obtained. Modeling tools are an important part of the evaluation process, and this includes not only simulation but analytical tools, traffic generators, etc. Several tools and prototypes have been developed in recent years. Our own tools include: video servers, implementing different storage techniques; voice transmission tool implementing FEC recovery mechanisms; distributed whiteboard (with multicast library), TANGRAM-II that includes a modeling environment with analytical as well as simulation solvers, traffic modeling environment and traffic generator and analyzer. (Most of the tools can be download from www.land.ufrj.br and/or www.dcc.ufmg.br.)

References

1. A. Adas. Traffic Models in Broadband Networks. *IEEE Communications Magazine*, (7):82–89, 1997.
2. C. C. Aggarwal, J. L. Wolf, and P. S. Wu. On optimal batching policies for video-on-demand storage server. In *Proc. of the IEEE Conf. on Multimedia Systems*, 1996.
3. C. C. Aggarwal, J. L. Wolf, and P. S. Wu. A permutation-based pyramid broadcasting scheme for video-on-demand systems. In *Proc. of the IEEE Conf. on Multimedia Systems*, 1996.
4. C.C. Aggarwal, J.L. Wolf, and P.S. Wu. On optimal piggyback merging policies. In *Proc. ACM Sigmetrics'96*, pages 200–209, May 1996.
5. J. Almeida, D. Eager, M. Ferris, and M. Vernon. Provisioning content distribution networks for streaming media. In *Proc. of IEEE/Infocom'02*, June 2002.
6. J. Apostolopoulos, T. Wong, W. Tan, and S. Wee. On multiple description streaming with content delivery networks. In *Proc. of IEEE/Infocom'02*, NY, June 2002.
7. A. Bar-Noy, G. Goshi, R. E. Ladner, and K. Tam. Comparison os stream merging algorithms for media-on-demand. In *Proc. MMCN'02*, January 2002.
8. S. Berson, R.Muntz, S. Ghandeharizadeh, and X. Ju. Staggered striping in multimedia information systems. In *ACM SIGMOD Conference*, 1994.
9. W. Bolosky, J.S. Barrera, R. Draves, R. Fitzgerald, G. Gibson, M. Jones, S. Levi, N. Myhrvold, and R. Rashid. The Tiger video fileserver. In *Proc. NOSSDAV'96*. 1996.
10. J-C. Bolot. Characterizing end-to-end packet delay and loss in the Internet. In *Proc. ACM Sigcomm'93*, pages 289–298, September 1993.
11. J-C. Bolot, S. Fosse-Parisis, and D. Towsley. Adaptative FEC-based error control for Internet telephony. In *Proc. of IEEE/Infocom'99*, pages 1453–1460, 1999.
12. J-C. Bolot and A. Vega-García. The case for FEC-based error control for packet audio in the Internet. *ACM Multimedia Systems*, 1997.
13. Y. Cai, K. Hua, and K. Vu. Optimizing patching performance. In *Proc. SPIE/ACM Conference on Multimedia Computing and Networking*, 1999.
14. S. Campos, B. Ribeiro-Neto, A. Macedo, and L. Bertini. Formal verification and analysis of multimedia systems. In *ACM Multimedia Conference*. Orlando, November 1999.
15. S. W. Carter and D. D. E. Long. Improving video-on-demand server efficiency through stream tapping. In *Sixth International Conference on Computer Communications and Networks*, pages 200–207, 1997.

16. S.-H.G. Chan and F. Tobagi. Tradeoff between system profit and user delay/loss in providing near video-on-demand service. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(8):916–927, August 2001.
17. E. Chang and A. Zakhor. Cost analyses for VBR video servers. *IEEE Multimedia*, 3(4):56–71, 1996.
18. A.L. Chervenak, D.A. Patterson, and R.H. Katz. Choosing the best storage system for video service. In *ACM Multimedia Conf.*, pages 109–119. SF, 1995.
19. T. Chua, J. Li, B. Ooi, and K. Tan. Disk striping strategies for large video-on-demand servers. In *ACM Multimedia Conf.*, pages 297–306, 1996.
20. A. Dan, D. Sitaram, and P. Shahabuddin. Scheduling policies for an on-demand video server with batching. In *Proc. of the 2nd ACM Intl. Conf. on Multimedia*, pages 15–23, 1994.
21. A. Dan, D. Sitaram, and P. Shahabuddin. Dynamic batching policies for an on-demand video server. *Multimedia Systems*, (4):112–121, 1996.
22. M.C. Diniz and E. de Souza e Silva. Models for jitter control at destination. In *Proc., IEEE Intern. Telecomm. Symp.*, pages 118–122, 1996.
23. D. Eager, M. Ferris, and M. Vernon. Optimized caching in systems with heterogenous client populations. *Performance Evaluation*, (42):163–185, 2000.
24. D. Eager and M. Vernon. Dynamic skyscraper broadcasts for video-on-demand. In 4^{th} *International Workshop on Multimedia Information Systems*, September 1998.
25. D. Eager, M. Vernon, and J. Zahorjan. Optimal and efficient merging schedules for video-on-demand servers. In *Proc. ACM Multimedia'99*, November 1999.
26. D. Eager, M. Vernon, and J. Zahorjan. Bandwidth skimming: A technique for cost effective video-on-demand. In *Proc. Multimedia Computing and Networking*, January 2000.
27. D. Eager, M. Vernon, and J. Zahorjan. Minimizing bandwidth requirements for on-demand data delivery. *IEEE Transactions on Knowledge and Data Engineering*, 13(5):742–757, September 2001.
28. F. Fabbrocino, J.R. Santos, and R.R. Muntz. An implicitly scalable, fully interactive multimedia storage server. In *DISRT'98*, pages 92–101. Montreal, July 1998.
29. D.R. Figueiredo and E. de Souza e Silva. Efficient mechanisms for recovering voice packets in the Internet. In *Proc. of IEEE/Globecom'99, Global Internet Symp.*, pages 1830–1837, December 1999.
30. C.S. Freedman and D.J. DeWitt. The SPIFFI scalable video-on-demand system. In *ACM Multimedia Conf.*, pages 352–363, 1995.
31. V.S. Frost and B. Melamed. Traffic Modeling for Telecommunications Networks. *IEEE Communications Magazine*, 32(3):70–81, 1994.
32. L. Gao and D. Towsley. Supplying instantaneous video-on-demand services using controlled multicast. In *IEEE International Conference on Multimedia Computing and Systems*, pages 117–121, 1999.
33. S. Ghandeharizadeh, R. Zimmermann, W. Shi, R. Rejaie, D. Ierardi, and T.-W. Li. Mitra: a scalable continuous media server. *Multimedia Tools and Applications*, 5(1):79–108, July 1997.
34. L. Golubchick, J.C.S. Lui, E. de Souza e Silva, and R.Gail. Evaluation of performance tradeoffs in scheduling techniques for mixed workload multimedia servers. *Journal of Multimedia Tools and Applications*, to appear, 2002.
35. L. Golubchick, J.C.S. Lui, and R. Muntz. Reducing i/o demand in video-on-demand storage servers. In *Proc. ACM Sigmetrics'95*, pages 25–36, May 1995.
36. Y. Guo, S. Sen, and D. Towsley. Prefix caching assisted periodic broadcast: Framework and techniques to support streaming for popular videos. In *Proc. of ICC'02*, 2002.

37. D. Heyman and D. Lucantoni. Modeling multiple ip traffic with rate limits. In J.M. de Souza, N. da Fonseca, and E. de Souza e Silva, editors, *Teletraffic Engineering in the Internet Era*, pages 445–456. 2001.
38. D.P. Heyman and T.V. Lakshman. What are the Implications of Long-Range Dependence for VBR-Video Traffic Engineering. *IEEE/ACM Transactions on Networking*, 4(3):301–317, June 1996.
39. Ailan Hu. Video-on-demand broadcasting protocols: A comprehensive study. In *Proc. IEEE Infocom*, pages 508–517, 2001.
40. K. A. Hua, Y. Cai, and S. Sheu. Patching: A multicast technique for true video-on demand services. In *Proceedings of ACM Multimedia*, pages 191–200, 1998.
41. K.A. Hua and S. Sheu. Skyscraper broadcasting: a new broadcasting scheme for metropolitan video-on-demand systems. In *Proc. of ACM Sigcomm'97*, pages 89–100. ACM Press, 1997.
42. J.Chien-Liang, D.H.C. Du, S.S.Y. Shim, J. Hsieh, and M. Lin. Design and evaluation of a generic software architecture for on-demand video servers. *IEEE Transactions on Knowledge and Data Engineering*, 11(3):406–424, May 1999.
43. P. Ji, B. Liu, D. Towsley, and J. Kurose. Modeling frame-level errors in gsm wireless channels. In *Proc. of IEEE/Globecom'02 Global Internet Symp.*, 2002.
44. S. Jin and A. Bestavros. Scalability of multicast delivery for non-sequential streaming access. In *Proc. of ACM Sigmetrics'02*, June 2002.
45. K. Keeton and R. Kantz. Evaluating video layout strategies for a high-performance storage server. In *ACM Multimedia Conference*, pages 43–52, 1995.
46. J. Korst. Random duplicated assignment: An alternative to striping in video servers. In *ACM Multimedia Conference*, pages 219–226. Seattle, 1997.
47. J.F. Kurose and K.W. Ross. *Computer Networking: A Top-Down Approach Featuring the Internet*. Addison-Wesley, 2001.
48. S.W. Lau, J.C.S. Lui, and L. Golubchik. Merging video streams in a multimedia storage server: Complexity and heuristics. *ACM Multimedia Systems Journal*, 6(1):29–42, January 1998.
49. R.M.M. Leão, E. de Souza e Silva, and Sidney C. de Lucena. A set of tools for traffic modelling, analysis and experimentation. In *Lecture Notes in Computer Science 1786 (TOOLS'00)*, pages 40–55, 2000.
50. Y. Mansour and B Patt-Shamir. Jitter control in QoS networks. *IEEE/ACM Transactions on Networking*, 2001.
51. A.P. Markopoulou, F.A. Tobagi, and M.J. Karam. Assessment of VoIP quality over Internet backbones. In *Proc. of IEEE/Infocom'02*, June 2002.
52. H. Michiel and K. Laevens. Traffic Engineering in a Broadband Era. *Proceedings of the IEEE*, pages 2007–2033, 1997.
53. R.R. Muntz, J.R. Santos, and S. Berson. A parallel disk storage system for real-time multimedia applications. *Intl. Journal of Intelligent Systems*, 13(12):1137–1174, December 1998.
54. B. Ozden, R. Rastogi, and A. Silberschatz. Disk striping in video server environments. In *IEEE Intl. Conference on Multimedia Computing and Systems*, 1996.
55. B. Ozden, R. Rastogi, and A. Silberschatz. On the design of a low-cost video-on-demand storage system. In *ACM Multimedia Conference*, pages 40–54, 1996.
56. K. Park and W. Willinger. *Self-Similar Network Traffic: an Overview*, pages 1–38. John Wiley and Sons, INC., 2000.
57. C. S. Perkins, O. Hodson, and V. Hardman. A survey of packet-loss recovery techniques for streaming audio. *IEEE Network Magazine*, pages 40–48, Sep. 1998.

58. S. Ramesh, I. Rhee, and K. Guo. Multicast with cache (mcache): An adaptative zero-delay video-on-demand service. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(3):440–456, March 2001.
59. R. Rejaie, H. Yu, M. Handley, and D. Estrin. Multimedia proxy caching mechanism for quality adaptive streaming applications in the Internet. In *Proc. IEEE Infocom*, pages 980–989, 2000.
60. Reza Rejaie, Mark Handley, and Deborah Estrin. Quality adaptation for congestion controlled video playback over the Internet. In *Proc. ACM Sigcomm'99*, pages 189–200, August 1999.
61. K. Salamatian and S. Vaton. Hidden Markov Modeling for network communication channels. In *Proc. of Sigmetrics/Performance'01*, pages 92–101, Cambridge, Massachusetts, USA, June 2001.
62. 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, 1998.
63. J.R. Santos and R. Muntz. Performance analysis of the RIO multimedia storage system with heterogeneous disk configurations. In *ACM Multimedia Conf.*, 1998.
64. J.R. Santos, R. Muntz, and B. Ribeiro-Neto. Comparing random data allocation and data striping in multimedia servers. In *Proc. ACM Sigmetrics'00*, pages 44–55. Santa Clara, 2000.
65. S. Sen, J. Rexford, J. Dey, J. Kurose, and D. Towsley. Online smoothing of variable-bit-rate streaming video. *IEEE Transactions on Multimedia*, 2000.
66. S. Sen, J. Rexford, and D. Towsley. Proxy prefix caching for multimedia streams. In *Proc. IEEE Infocom*, pages 1310–1319, 1999.
67. P.J. Shenoy and H.M. Vin. Efficient striping techniques for multimedia file servers. In *Proc. NOSSDAV'97*, pages 25–36. 1997.
68. H. Tan, D. Eager, M. Vernon, and H. Guo. Quality of service evaluations of multicast streaming protocols. In *Proc. of ACM Sigmetrics 2002*, June 2002.
69. D. Towsley, J. Kurose, and S. Pingali. A comparison of sender-initiated and receiver-initiated reliable multicast protocols. *IEEE Journal on Selected Areas in Communications*, 15(3):398–406, April 1997.
70. S. Viswanathan and T. Imielinski. Pyramid broadcasting for video on demand service. In *Proc. IEEE Multimedia Computing and Networking*, volume 2417, pages 66–77, 1995.
71. B. Wang, S. Sen, M. Adler, and D. Towsley. Optimal proxy cache allocation for efficient streaming media distribution. In *Proc. IEEE Infocom*, 2002.
72. Y. Wang, Z. Zhang, D. Du, and D. Su. A network-conscious approach to end-to-end video delivery over wide area networks using proxy servers. In *Proc. of IEEE Infocom 98*, pages 660–667, Abril 1998.
73. W.R. Wong. *On-time Data Delivery for Interactive Visualization Apploications*. PhD thesis, UCLA/CS Dept., 2000.
74. D. Wu, Y.T. Hou, and Y. Zhang. Transporting real-time video over the Internet: Challenges and approaches. *Proceedings of the IEEE*, 88(12):1855–1875, December 2000.
75. M. Yajnik, S. Mon, J. Kurose, and D. Towsley. Measurement and modeling of the temporal dependence in packet loss. In *Proc. of IEEE/Infocom'99*, 1999.
76. E. Steinbach Yi J. Liang and B. Girod. Real-time voice communication over the Internet using packet path diversity. In *Proc. ACM Multimedia 2001*, Ottawa, Canada, Sept./Oct. 2001.