# Design and Evaluation of Data Access Strategies in a High Performance Multimedia-on-Demand Server *

**Divyesh Jadav Chutimet Srinilta Alok Choudhary P. Bruce Berra**
Department of Electrical and Computer Engineering
and CASE Center
Syracuse University Syracuse, NY 13244
divyesh, choudhar, csrinilt, berra @cat.syr.edu

## Abstract

*One of the key components of a multi-user multimedia-on-demand system is the data server. Digitalization of traditionally analog data such as video and audio, and the feasibility of obtaining network bandwidths above the gigabit-per-second range are two important advances that have made possible the realization, in the near future, of interactive distributed multimedia systems. Secondary-to-main memory I/O technology has not kept pace with advances in networking, main memory and CPU processing power. Consequently, the performance of the server has a direct bearing on the overall performance of such a system.*

*In this paper, we develop a model for the architecture of a server for such a system. Parallelism of data retrieval is achieved by striping the data across multiple disks. The performance of any server ultimately depends on the data access patterns. Two modifications of the basic retrieval algorithm are presented to exploit data access patterns in order to improve system throughput and response time. A complementary information caching optimization is discussed. Finally, we present performance results of these algorithms on the IBM SP1 and Intel Paragon parallel computers.*

## 1 Introduction

### 1.1 Motivation

A *Multimedia Information System* requires the integration of communication, storage and presentation mechanisms for diverse data types including text, images, audio and video, to provide a single unified information system [1]. The reason why multimedia data processing is difficult is that such data differs markedly from the unimedia data (text) that conventional computers are built to handle [2]. The chief differences are the need to provide real-time guarantees and the diverse data sizes.

In this paper, we focus on the most popular multimedia application, *video-on-demand* in a distributed environment. This term refers to making it possible for multiple viewers to interactively view The implications of such a system on the technology and the infrastructure needed are tremendous. The storage of even a modest hundred movies requires almost a terabyte of storage capacity in the server. Similarly, gigabyte/sec and terabyte/ sec bandwidth networks are necessary to carry the movies to the consumers.

In the absence of adequate hardware support, past and present interactive digital multimedia systems have been forced to make compromises such as providing single-user instead of multi-user support, small-window displays instead of full-screen display of video and image data, the use of lossy compression techniques and low audio/video resolution. Recent advances in underlying hardware technologies, such as the emergence of high-speed networks (eg: ATM) and powerful CPUs, however, obviate the need for such compromises.

In spite of these technological advances, there is one bottleneck that plagues the realization of such a system : the speed of data transfer from the secondary data storage to main memory. The data transfer time in the most popular form of secondary storage, magnetic disks, is still governed by the seek and rotational latencies of these devices. These latencies have not decreased commensurately with the advances in other areas of computer hardware. Multimedia information systems are inherently I/O intensive, and it is critical to reduce the ill-effects of this bottleneck. Techniques for doing so are the subject of this paper.

### 1.2 Related Work

[3] have proposed file system design techniques for providing hard performance guarantees. [5, 6] pro-

*This work is supported by Intel Corporation, NSF Young Investigator Award CCR-9357840, and by the New York State Center for Advanced Technology in Computer Applications and Software Engineering (CASE) at Syracuse University. The authors thank the Argonne National Laboratory for providing access to the IBM SP1, and the Caltech CCSF facilities for providing access to the Intel Paragon





Proceedings of the International Conference on Multimedia Computing and Systems (ICMCS '95)
0-8186-7105-X/95 $10.00 © 1995 IEEE

### 1.2 Related Work

[3] have proposed file system design techniques for providing hard performance guarantees. [5, 6] proposed a disk arm scheduling approach for multimedia data, and characterized the disk-level trade-offs in a multimedia server. [2, 7] have proposed a model based on constrained block allocation, which is basically non-contiguous disk allocation in which the time taken to retrieve successive stream blocks does not exceed the the playback duration of a stream block. Contiguous allocation of disk blocks for a media stream is desirable, for it amortizes the cost of a single seek and rotational delay over the retrieval of a number of media blocks, thus minimizing the deleterious effects of disk arm movement on media data retrieval. However, contiguous allocation causes fragmentation of disk space if the entire stream is stored on a single disk. Moreover, if a stream is stored on a single disk, the maximum retrieval bandwidth is restricted by the data transfer rate of the disk. [8] get around these problems by striping media data across several disks in a round robin fashion. The effective retrieval bandwidth is then proportional to the number of disks used. Our model is similar to this model in using data striping, round robin distribution of successive stream fragments and contiguous allocation within a given fragment. [7] categorize real time clients into 2 classes, those that require hard and soft performance guarantees, respectively. For the latter class, the worst case assumptions made in admitting new users are relaxed based on the observed server load to increase the number of users that can be supported. Most previous work has concentrated on minimizing rotational and seek overheads in retrieving data. Our approach is to increase the granularity of data retrieved so that the random effects of disk overheads form a smaller fraction of request service time. Moreover, little attention has been paid to the issue of tuning server performance based on user access patterns.

### 1.3 Our Research Contributions

In this paper, an integrated approach to the storage and retrieval of video data so as to maximize the number of users, while at the same time providing real-time service, is presented. Our model uses parallelism of retrieval to tackle the problem of the low speed of data transfer from secondary-storage to main memory. An algorithm (the *Remote Disk Stream Scheduling (RDSS) algorithm,* ) for server operation when sourcing a constant number of media streams is presented. Two modifications of the basic RDSS algorithm, the *Local Disk Stream Scheduling (LDSS)* and the *Local Memory Stream Scheduling (LMSS)* algorithms, are developed that exploit knowledge of data access patterns to improve system throughput.

We propose a complementary approach, *Gang scheduling*, that increases the number of streams that a server can support when there are multiple requests for the same media object, at the cost of increased response time for some clients. We discuss the trade-offs that this approach involves. Finally, we present experimental results on the IBM SP1 and Intel Paragon parallel computers.

The rest of this paper is organized as follows : Section 2 describes the architecture of the server. Section 3 describes the proposed scheduling policies. We present performance results in Section 4. Section 5 summarizes this paper.

## 2 The High Performance Server

### 2.1 Architecture

As explained above, multimedia applications strain the resources of a uniprocessor computer system for even a single-user mode of operation. When the server has to handle multiple requests from multiple users simultaneously, it is clear that the server must be considerably more powerful than a PC or workstation-type system. At the very least, the server should have terabytes of secondary storage, gigabytes of main memory, and be connected to a high-speed wide-area network. The server may also be required to perform fast compression of multimedia data. Hence it should have good floating-point and scalar arithmetic performance. A parallel computer is a good candidate to satisfy these requirements.

However, it must be noted that most parallel computers available till recently have concentrated on minimizing the time required to handle workloads similar to those found in the scientific computing domain. Hence, the emphasis was laid on performing fast arithmetic and efficient handling of vector operands. On the other hand, multimedia-type applications require fast data retrieval and real-time guarantees. Secondly, parallel computers have traditionally been expensive on account of their high-end nature and the comparatively small user community as compared to that of PCs. The advent of multimedia applications has brought the esoteric parallel machines in direct competition with volume-produced PCs and workstations.

We propose a *logical* model for a continuous media server, which is independent of the architectural implementation. The same model can be implemented on a parallel machine or a collection of PCs/workstations interconnected by high-speed links. In this paper, we have used the parallel computer approach to validate our work.

Accordingly, the architecture of the server is that of a parallel computer with a high-capacity magnetic disk(s) per node, with the nodes being connected by a high-speed interconnection network. Each node is a computer in its own right, with a CPU, RAM and secondary storage. In addition, each node has an interface with the interconnection network. This model allows one to stripe multimedia data across the disks of the server. This allows its retrieval to proceed in



Proceedings of the International Conference on Multimedia Computing and Systems (ICMCS '95)
0-8186-7105-X/95 $10.00 © 1995 IEEE

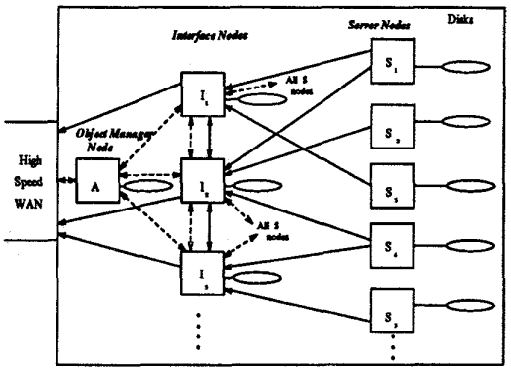


Figure 1: Logical Model of the Server. Example communication patterns are shown: dark lines indicate data, dotted lines indicate control information

parallel, thus helping the server to satisfy real-time requirements. In addition, the shrinking size and cost of RAM makes it possible to have hundreds of megabytes of main memory per node; memory capacity of this range is an advantage for buffering multimedia data during storage and retrieval.

## 2.2 Logical Model of the Server

Figure 1 shows a block diagram of the logical view of the proposed server. The physical server nodes are divided into three classes based on functionality : **Object Manager** ($A$), **Interface** ($I$), and **Server** ($S$) nodes. The three types of nodes are explained in greater detail below :

1. **The Object Manager node** is at the top of the server's control hierarchy. The Object Manager receives all incoming requests for media objects. It has knowledge of which Server nodes an object resides on and the workload of the Interface nodes. Based on this knowledge, it *delegates the responsibility* of serving a request to one of the Interface nodes. The Object Manager node also *logs data request patterns*, and uses this information to optimize server response time and throughput. This is explained in 3.2.

2. **Interface Nodes** are responsible for scheduling and serving stream requests that have been accepted. Their main function is to request the striped data from the server nodes, order the packets received from the server nodes, and send the packets over the wide area network to the clients. *Efficient buffer management* algorithms are vital towards achieving these functions. An interface node can also use its local secondary storage to source frequently accessed data objects.

| *Symbol* | *Description* |
|---|---|
| $R_{pl}$ | Required playback rate |
| $P_I$ | Size of packets sent by an $I$ node |
| $\delta_I$ | Duration of a packet sent by an $I$ node |
| $B_I$ | Buffer size at an $I$ node |
| $P_S$ | Size of packets sent by a $S$ node |
| $\delta_S$ | Duration of data in $B_I$ |
| $T_f$ | Period of issuing fetches to S |
| $S$ | Stripe factor |

Table 1: The parameters used in this paper

3. **Server Nodes** actually *store* multimedia data on their secondary storage in a striped fashion, and *retrieve* and transmit the data to an interface node when requested to do so. It is to be noted that the disk-per-node assumption is not literal : a node can have a disk-array instead for greater I/O throughput.

# 3 Scheduling Algorithms

## 3.1 Parameters Used and Scheduling Constraints

The data is stored at the server and transmitted in compressed digital form. For the purposes of this paper we assume the MPEG-1 compression standard [4]. The decompression of the data is done at the remote client's *multimedia terminal*. We assume that the wide-area network have the necessary bandwidth to support multimedia data rates and multiple clients. As mentioned earlier, the data is compressed and striped across the server nodes in a round-robin fashion. The number of nodes across which an object is striped is called the *stripe factor*. Since the stripe fragments on any given server node's disk are not consecutive fragments, it is not necessary to store them contiguously. Disk scheduling algorithms to optimize retrieval from the disk surface have been proposed [5], and can be used in our model. We are concerned with harnessing the parallelism provided by striped storage and investigating the buffering policies for the data. Table 1 shows the parameters used by our model.

$\delta_I$ is the time for which a packet sent by an $I$ node to a client will last at the client. Hence, this is also the deadline by which the next packet from the $I$ node must be received at the client. Its value is given by:

$$\delta_I = \frac{P_I}{R_{pl}} \quad (1)$$

Once the requested stripe fragments from the $S$ nodes have arrived at the destination $I$ node, the latter arranges them in the proper sequence and continues sending packets of size $P_I$ to the client no less than every $\delta_I$ seconds. The buffer at the $I$ node will last for $\delta_S$ time, before which the next set of stripe fragments must have arrived from the $S$ nodes.



Proceedings of the International Conference on Multimedia Computing and Systems (ICMCS '95)
0-8186-7105-X/95 $10.00 © 1995 IEEE

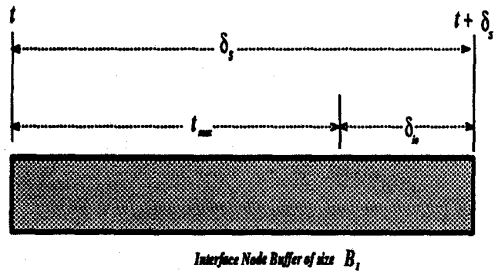


Figure 2: Time relationships of $I$-$S$ node traffic

The average time to retrieve $P_S$ bytes from a $S$ node is given by

$$\delta_{io} = \delta_{rq} + \delta_{avg_{seek}} + \delta_{avg_{rot}} + \delta_{tr_{P_S}} + \delta_{nw_{P_S}} \quad (2)$$

where $\delta_{rq}$ is the time delay for a request from an $I$ node to reach a $S$ node, $\delta_{avg_{seek}}$ and $\delta_{avg_{rot}}$ are the average seek and rotational latencies for the disks being used, $\delta_{tr_{P_S}}$ is the disk data transfer time for $P_s$ bytes, and $\delta_{nw_{P_S}}$ is the network latency to transport $P_s$ bytes from a $S$ node to an $I$ node.

Thus, if the playout of an $I$ node buffer is started at time $t$, then the *latest* time by which the requests for the next set of stripe fragments must be issued to the $S$ nodes is :

$$t_{max} = t + \delta_S - \delta_{io} \quad (3)$$

Figure 2 shows these relationships.

Note that equation 2 uses average seek and rotational latencies for disk accesses. Since these latencies are variable, there will be boundary conditions when the time to retrieve $P_S$ bytes is much more (less) than the average value. However, the effect of this deviation from the average value on the overall service time depends on the relative magnitudes of the other components of the service time. Our approach is based on the fact that when the granularity of data read from disk is increased, the effect of random disk seek and rotational overheads is reduced. While it is true that doing so increases buffering requirements, contemporary processors have large main memories, and using such processors is well worth the gain obtained in making disk service time more predictable. Another point in support of using average overhead values is that the worst case is rarely encountered in practice. Of course, if some clients require strict performance guarantees, then one can categorize users into those requiring hard and soft deadlines as in[9], and use the maximum values of the disk overheads for admitting users of the latter kind.

## 3.2 Exploiting Data Access Patterns

It is natural that certain objects in a database are accessed more frequently than other objects. For example, in this particular application, it is highly likely that the demand for newly released movies will be higher than that for older movies. We now present three different algorithms that address this issue. The first algorithm does not take frequency of data access into account, while the next two exploit this feature to reduce the response time to new requests.

### 3.2.1 Remote Disk Stream Scheduling Algorithm (RDSS)

In this algorithm, each video stream is scheduled by explicitly retrieving stripe fragments from the $S$ nodes. In this approach the I/O scheduler takes no advantage of the possibility that the same multimedia object is being used by multiple users simultaneously. Consequently, when many objects have this reference pattern, this policy will create excess interconnection-network and disk traffic. However, it is the simplest to implement.

### 3.2.2 Local Disk Stream Scheduling Algorithm (LDSS)

This algorithm and the next one depend on being able to detect that some objects are being accessed more frequently than others. This function can be performed by the object manager node (node $A$ in figure 1). Since all new requests for streams come to this node, it can log the object access patterns over a specified *time window*, $\Delta_t$. If any object is accessed at a rate above a threshold, $Th_{pop}$, then that object is classified as a *popular object*.

Having identified an object as being popular, when the next request for that object comes in, the stripe fragments are retrieved from the $S$ nodes in the usual way. However, in addition to sending packets of size $P_I$ to the client, the stripe fragments retrieved from the $S$ nodes are written to the *local disk at the corresponding $I$ node*. Thus, when the next request for the object comes in, the object can be streamed from the local disk(s) of the $I$ node. This has the benefit of reducing interconnection-network and ($S$ node) disk traffic, and also improving the overall response time of the system. Note that the overhead of storing the stripe fragments on local disk is marginal, since disk writes are non-blocking and can proceed in the background.

### 3.2.3 (Local) Memory Stream Scheduling Algorithm (LMSS)

This algorithm goes a step further in reducing system response time for popular objects. In this case, a popular object is stored on the $I$ node backing store as in the LDSS scheme. In addition, the first few packets of the object are stored in the *main memory of the $I$ node*, so that when a request comes in, it can be served immediately once it has been accepted.



Proceedings of the International Conference on Multimedia Computing and Systems (ICMCS '95)
0-8186-7105-X/95 $10.00 © 1995 IEEE

| *Symbol* | *Value* |
|---|---|
| $R_{pl}$ | 1.5 Mb/sec |
| $P_I$ | 64 KB |
| $B_I$ | $P_S * S$ |
| $P_S$ | 128 KB |

Table 2: Parameter values used for experiments

In both the LDSS and LMSS schemes, it is also necessary to keep track of when the frequency of access of a object falls below the threshold separating popular object and other objects. In that case, the disk space occupied by that object at the $I$ node can be used to store another popular object.

## 4 Results

We have evaluated the performance of our logical server model. We present results for the IBM SP1 and Intel Paragon below.

Due to storage space and availability of real-world data limitations, the disk access part was simulated. We have assumed gigabytes of disk space per node, and a disk data transfer rate of 10 Mbytes/sec. The time for one rotation of the disk was modeled as 11.1 ms, while the average seek time was modeled as 9.4 ms. Table 2 shows the values of the parameters defined in table 1 that we used for our experiments. The playback time for each stream varied between 4 and 5 minutes, depending on the time of arrival of the request for that stream.

### 4.1 Performance of the RDSS, LDSS and LMSS algorithms

We noted the performance of the algorithms for a server configuration of 6 interface nodes and 24 server nodes, and a stripe factor of 4. The composition of the requests was varied as follows : starting from requests for unique media objects, the percentage of requests for the same object was successively increased. Figure 3 shows the maximum number of streams that could be simultaneously supported using each policy on the SP1.

We observe that for a low percentage of requests for the same object, the *RDSS* algorithm outperforms the other two algorithms. This is so because in the latter two cases we allocate a dedicated $I$ node for the popular object. For a low percentage of requests for the popular object, the dedicated node is underutilized : it sources less streams than its full capacity, while a normal $I$ node in its place could have sourced the maximum number of streams that such a node can source. With increasing amounts of requests for the same object, however, the *LDSS* and *LMSS* algorithms outperform the *RDSS* algorithm as they reduce the load on the server nodes caused by frequently accessing the same object. Between the *LDSS* and *LMSS* algorithms, the latter clearly outperforms the former for

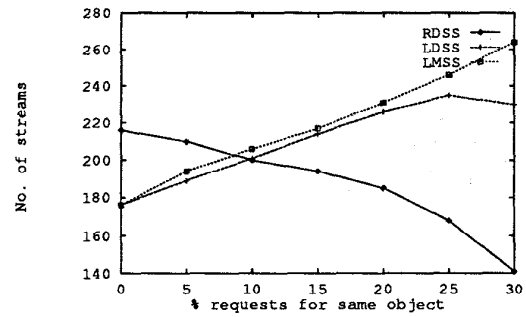


Figure 3: Maximum number of supported streams for varying number of requests for the same object (6 I nodes, 24 S nodes, IBM SP1).

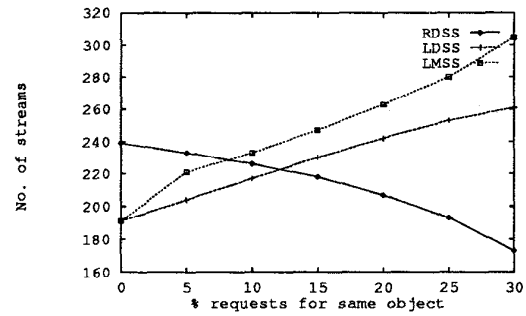


Figure 4: Maximum number of supported streams for varying number of requests for the same object (6 I nodes, 24 S nodes, Intel Paragon).

different values of the percentage of requests for a popular object. Lastly, the performance of the *RDSS* algorithm deteriorates rapidly as the percentage of requests for the popular object is increased, due to the corresponding increase in the load of the $S$ nodes on which the popular object is stored.

We ported our code to the Intel Paragon and repeated the same experiment as above. Figure 4 shows the results. The effect of varying the number of requests for the same object on the maximum number of streams that can be supported is similar as above.

### 4.2 Gang Scheduling

The *LDSS* and *LMSS* algorithms exploit the fact that some objects are more popular than others, and thus are requested more frequently. This fact is used to maximize the number of supportable streams of such objects by dedicating nodes to service requests for them.

In the first set of experiments, the servicing of a



Proceedings of the International Conference on Multimedia Computing and Systems (ICMCS '95)
0-8186-7105-X/95 $10.00 © 1995 IEEE

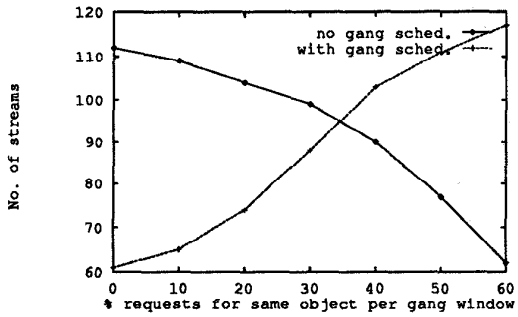


Figure 5: Number of supported streams for RDSS algorithm for varying number of requests for the same object per gang window (stripe factor 5, 2 $I$ nodes, 6 $S$ nodes, Paragon).

request is started as soon as the request has been admitted. The performance of all three algorithms can be improved by accumulating requests over an interval of time, and avoiding multiple fetches for requests received for the same object during that interval of time. We call this method *gang scheduling*. For instance, if during a *gang window* of 5 minutes, 10 requests are received for a certain object, then the server can start retrieving only one stream at the end of the gang window and source 10 client streams from the one stream.

For evaluating gang scheduling, we used a configuration of 2 $I$ nodes and 6 $S$ nodes, and a stripe factor of 5. We used a gang window of 1.5 seconds and 30 requests per gang window. Of course, in practice a longer window would be used. Without loss of generality, we use the window mentioned for the run time of 5 minutes. The values of the other parameters are the same as in table 2. Figure 5 shows the effect of varying the percentage of requests for the same object per gang window on the maximum number of streams that can be supported on the Paragon for the RDSS algorithm.

Gang scheduling involves an extra overhead of accumulating requests over the gang window and searching through the accumulated requests to identify repeated requests. Hence we observe from the figure that RDSS with gang scheduling is inferior to pure RDSS for low number of repeated requests per gang window. However, as the percentage of requests for the same object per gang window increases, RDSS with gang scheduling identifies the request pattern and outperforms pure RDSS. In effect, this method delays the servicing of some admitted requests in order to minimize the load on the server. Hence there is a tradeoff between the response time for clients and reduction in server workload. Consequently, the size of the gang window is a crucial parameter in making use of gang scheduling.

# 5 Conclusions

In this paper we have presented an I/O model for a server in a distributed multimedia system. Three algorithms that exploit knowledge of data access patterns were developed to maximize the number of streams that the server can source simultaneously. Our experiments showed that the *LMSS* algorithm outperforms the *LDSS* algorithm, which in turn outperforms the *RDSS* algorithm when an appreciable percentage of stream requests are for the same media object. We showed the utility of gang scheduling in further improving the performance of all three algorithms. In gang scheduling, a single stream between interface and server nodes is used to serve multiple clients. In conclusion, we reiterate that it is crucial to exploit user access patterns to maximize the throughput of a multimedia server.

# References

[1] P. B. Berra, C.-Y. Chen, A. Ghafoor and T. Little. Issues in networking and data management of distributed multimedia systems. In *proceedings of the First International Symposium on High Performance Distributed Computing*, September 1992.

[2] P. V. Rangan and H. Vin. Efficient storage techniques for digital continuous multimedia. *IEEE Transactions on Knowledge and Data Engineering* , Vol. 5, No. 6, August 1993.

[3] D. Anderson, Y. Osawa and R. Govindan. A file system for continuous media. *ACM Trans. on Computer Systems*, Vol. 10, No. 4, November 1992.

[4] D. Le Gall. MPEG: a video compression standard for multimedia applications. *Communications of the ACM*, April 1991.

[5] A. Reddy and J. Wyllie. Disk-scheduling in a multimedia I/O system. *Proceedings of the 1st ACM Intl. Conference on Multimedia*, August 1993, pg. 225.

[6] A. Reddy and J. Wyllie. I/O issues in a multimedia system. *IEEE Computer*, March 1994.

[7] P. Rangan, H. Vin and S. Ramanathan. Designing an on-demand multimedia service. *IEEE Communications*, Vol 30, No. 7, July 1992.

[8] S. Ghandeharizadeh and L. Ramos. Continuous retrieval of multimedia data using parallelism. *IEEE Trans. on Knowledge and Data Engineering*, Vol. 5, No. 4, August 1993.

[9] H. Vin, A. Goyal, et. al. An observation-based admission control algorithm for multimedia servers. *Proc. of the Intl. Conference on Multimedia Systems and Computing*, May 1994.



Proceedings of the International Conference on Multimedia Computing and Systems (ICMCS '95)
0-8186-7105-X/95 $10.00 © 1995 IEEE