Smart Multimedia File Objects

Michael S. Gingras and Jon B. Weissman

Division of Computer Science

The University of Texas at San Antonio

San Antonio, TX 78249

Abstract

In this paper, we describe a method, Smart File Objects (SFOs), for optimizing access of remote files across wide-area and wireless networks. The SFO is application-specific middleware designed to address the network bottleneck caused by high latency, low bandwidth networks. The SFO adaptively caches and prefetches data concurrently with application execution. We have applied the SFO concept to an MPEG player accessing remote Internet files. The SFO estimates the playback time using a set of derived cost models for the different components of the MPEG play process, and the communication overhead by timing the retrieval of remote video frames. This information allows the SFO to determine the amount of overlap between download and playback. The results indicate that playtime can be accurately estimated and both overall viewing time and viewing latency can be substantially reduced using this approach.

1.0 Introduction

Applications increasingly require access to remote data sources across wide-area and wireless networks. Remote file access suffers from the high latency, low bandwidth, and unpredictable performance of these networks. Most wide-area distributed file systems including

AFS [7], xFS [1], and Web browsers adopt a download model of file access in which the remote file is first brought down to the application's location. This "one-size-fits-all" approach is not appropriate for all network applications. In addition, the network bottleneck must be addressed in an application-specific manner. A Smart File Object (SFO) is application-level middleware that buffers file access over a wide-area or wireless network [8]. It adaptively prefetches and caches needed data concurrently with application execution to mitigate the impact of the network (Figure 1). The SFO is programmed to be *application-* and *network-aware* to obtain the best possible performance. An SFO is adaptive based on the application's patterns and frequency of data access as well as the bandwidth available to it.

This paper describes the creation of an SFO for an MPEG player. The use of the SFO required minimal changes to the player. The MPEG player used for the development of this SFO was *mpeg_play* version 2.3 written at UC Berkeley. It was selected because the source code was readily available and could be easily compiled on many UNIX platforms running X Windows. MPEG movies are typically accessed via a Web browser which first downloads the entire file and then executes the MPEG player. This process introduces a significant amount of delay before the application can begin. The SFO interfaces between the MPEG player and a remote file server such as a Web server, to overlap download and playtime as much as possible.

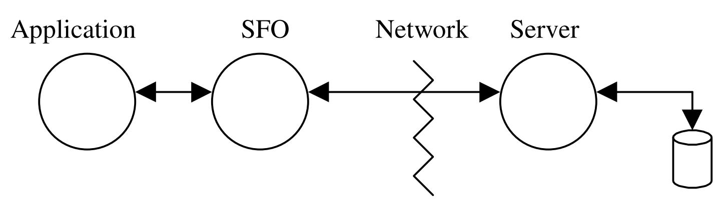

Figure 1: A generic SFO

The results demonstrate that for a wide-variety of MPEG files accessed across the Internet, the SFO was able to reduce playtime by 20% and viewing latency by 50% on average. In addition, accurate prediction of MPEG playtime is also possible, within a 10% error, even if the client machine is shared among competing users.

The most popular approach for overlapping playback and content retrieval for multimedia applications is streaming [6]. Our approach is different although the goals are similar (when compared to multimedia SFOs). Streaming approaches generally consider only network performance (not application demand) and require special protocols and file formats. The SFO is a more general approach that can be applied to a larger variety of network applications, including metacomputing applications [3]. While streaming requires a streaming server, the SFO can be implemented solely on the client side. This allows improved access to legacy data that has not been converted to a streaming format and server.

2.0 Approach

2.1 The SFO

The primary feature of the SFO is that it will concurrently download and display the MPEG file. While the initial part of the file is being downloaded, the SFO is measuring the rate at which it is receiving data and estimating the amount time the MPEG will play. Estimating playtime of an MPEG is complex. An important component of this paper is the model developed for predicting playtime. This prediction is used to calculate the minimum amount of the MPEG that must be downloaded first to fully overlap the download and display of the remaining part of the MPEG file. When the SFO has prefetched this minimum amount of the MPEG file, it creates a second process (child). The SFO (parent) process resets the file handle to the beginning of the file before it returns it to the player. The player begins decoding and displaying the MPEG at this time. The child continues to download the file until it is complete. The file becomes a form of interprocess communication with one process reading from the head of the file while the other process is writing to the end of the file (Figure 2).

The SFO provides two functions to the player: *SFOopen()* and *SFOread()*. The function *SFOopen()* replaces *fopen()* and has the same parameters with one exception, the file parameter can either refer to a local file or a URL for a file located on an http server. When *SFOopen()* is called with a local file name, it simply passes it on to *fopen()*. When *SFOopen()* is called with a URL, it establishes a TCP/IP connection with the http server, requests the MPEG file and stores the downloaded data in a local file.

The SFO must address the case in which data is requested before it is available. Since *fread()* will return an end of file, *SFOread()* must determine whether the true end of file has been reached or if the player has simply "out run" the download process. If the player has out run the download, *SFOread()* will block until more data has arrived. In this instance, the person viewing the MPEG will notice a pause. The job of the SFO is to try to prevent occurrence.

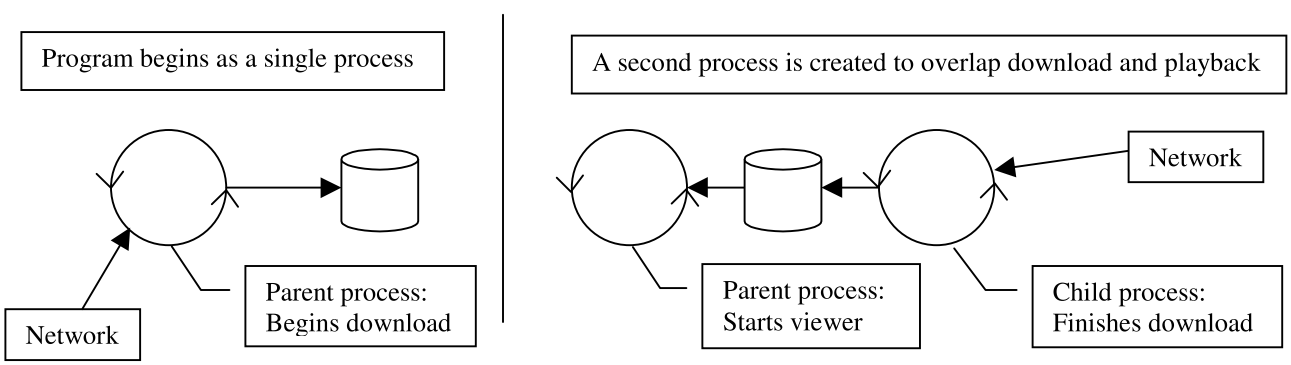

Figure 2: The SFO overlap process

2.2 Estimating Download Time

The SFO estimates the rate at which the rest of the file will be downloaded. It simply takes the total number of bytes downloaded and divides by the time taken. This has the advantage of smoothing out any communication spikes that may appear. Another strategy is to track a moving window of the last n frames, but this would require a method for spike removal that can be complex. In addition, more sophisticated network prediction methods have also been proposed in [9]. We plan to investigate these options in the near future. The current SFO implementation also ignores all information on how variable the download rate has been. This information could be used to determine how aggressive the SFO can be when determining when to start playing the MPEG.

2.3 Estimating MPEG Playtime

Estimating the time an MPEG will play is far more complicated. In order to produce a method for estimating playtime, the MPEG viewer was instrumented. We benchmarked 30 MPEGs with very different characteristics to construct a set of cost functions for predicting playtime. The data collected showed that most of the MPEG playtime falls into one of four steps: decoding, dithering, displaying or sleeping.

2.3.1 Decode Time

Estimating decode time requires a basic knowledge of the internals of an MPEG file. An MPEG is a series of pictures or frames stored in a compressed format. While some frames contain an entire picture, most frames contain only changes in the image relative to another frame or frames. The MPEG standard, ISO/IEC 11172 [4], defines three main frame types. Intra-coded frames (*I* frames) are complete pictures similar to a JPEG, that have two purposes. The first is to establish a baseline from which predictive frames are encoded. The second is to provide random access entry points into the video. Predictive coded frames (*P* frames) are encoded more efficiently, storing changes from the most recent reference frame. *P* frames can use the most recent *I* frame or *P* frame as their reference. Bidirectionally-predictive coded frames (*B* frames) offer the highest degree of compression by encoding the difference between a past reference frame and a future reference frame. *B* frames are never used as a reference for any other frame [4].

The order and number of each frame type is not defined by the ISO standard and is determined by the encoder. In practice, however, *I* frames are rarely more than 12 frames apart and at most three *B* frames are stored in a row [5]. These limits prevent degradation of the video as the delta stored in *B* frames are not exact but are approximations of the difference between three frames. Many MPEGs only use I frames. A typical series of frames is shown in Figure 3.

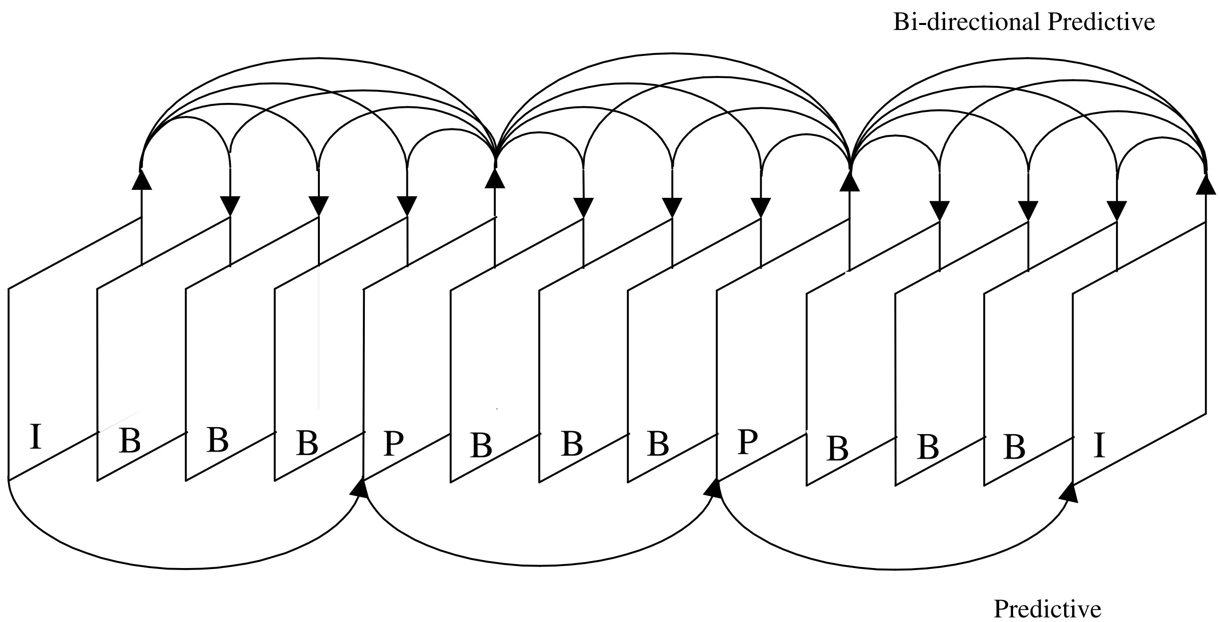

Figure 3: Typical Sequence of MPEG Frames

Frames are further divided into 16x16 pixel blocks called macroblocks. For efficiency, differences between frames are calculated and stored per macroblock rather than per pixel. Each macroblock is defined to be one of four types: intra-coded (*I*), forward-predictive (*P*), backward-predictive (*B)* and bidirectionally-predictive (*Bi*). Macroblocks do not have to be the same type as the frame in which they are contained, but the combinations are limited. For example, *I* frames can only contain *I* macroblocks, *P* frames can contain *I* and *P* macroblocks, but *B* frames can contain any type of macroblock (Figure 4). The SFO currently uses only frame-level information and does not consider the macroblock structure. However, macroblocks can provide more detailed information to the SFO if a more accurate prediction of decode time is needed.

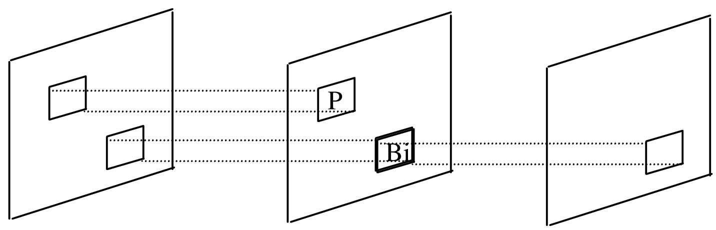

Figure 4: Macroblock Illustration

Frames are typically displayed at a rate of 25 to 30 frames per second. The frame rate is specified in the header of the MPEG file. For each frame type, there is a linear relationship between the frame length and the number of CPU cycles for decoding [2]. These relationships change for each hardware platform and decode routine. Calculating these relationships or cost equations required a significant amount of off-line data collection and processing before the SFO could be implemented. The size and decode time for thousands of frames were collected and separated by frame type. Linear regression was used to determine the equations that predict decode time for each of the three frame types as a function of frame length.

As the MPEG is downloaded, it is scanned for frame boundaries. For each frame in the MPEG, the frame type and length are determined. The frame length is input into the equation for the appropriate frame type and the result is the estimated time to decode that frame. The sum of the decode times for each frame will result in the estimated total time spent decoding the MPEG.

2.3.2 Dither and Display Time

The technique to estimate dither and display time is similar. Both dither and display are performed on the decoded picture so the frame type and length are irrelevant. Dithering is the process of mapping the color value assigned to a pixel to the nearest color in the current display pallet. Display is the process of illuminating the pixels on the screen. Estimating dither and display time is not as complicated as estimating decode time since the factors involved are constant throughout the MPEG and are readily available in the MPEG header. In particular, the height and width of the decoded picture in pixels is stored in the MPEG header. There is a linear relationship between this picture size and the amount of time required for dither and display. The picture size is constant for every frame in an MPEG, and therefore so is the display and dither time. The total dither or display time is the product of the time to dither or display one frame and the total number of frames in the MPEG.

2.3.3 Sleep Time

The viewer sleeps at the end of each frame if and only if it is necessary to achieve the frame rate. In other words, the frame rate is actually the maximum frame rate. If there is not sufficient CPU power to decode, dither, and display a given frame in 1/*frame rate* seconds, then the actual frame rate will be less than the specified frame rate. If the frame can be decoded, dithered, and displayed in under 1/*frame rate* seconds, then the viewer must sleep long enough to ensure it does not exceed the specified frame rate. The amount of time spent sleeping during a given MPEG is a function of the CPU load, picture size, frame pattern and frame rate. The minimum amount an MPEG will play is approximately the number of frames divided by the *frame rate* (minimum playtime). If the sum of decode, dither, and display exceeds this minimum playtime, than the viewer will not sleep. If the sum is less than the minimum playtime, the viewer will sleep an amount equal to the difference between the maximum frame rate and this sum.

2.3.4 Total MPEG Playtime

If the sum of decode, dither, and display is less than the minimum playtime, the best estimator of the actual playtime is the minimum playtime. Otherwise, the best estimator of the actual playtime is this sum. The faster the computer used for viewing, the more often the minimum time will be used. However, if the player is run on a slower computer or the load on the CPU is increased, then the sum of the components is a more likely estimator.

2.3.5 Effect of CPU Load

As the CPU load increases, the estimated playtime must change. The SFO samples the CPU load and applies it to the sum of the three component times. The measured CPU load is used to adjust the playtime equation by a linear factor. A more precise model for load would also include a model for disk I/O load and competing video demand. For the purpose of this SFO, modeling CPU load was sufficient.

4.0 Results

The SFO was tested on a suite of twelve MPEGs over a wide-area network. The MPEGs came from different sources and, therefore, have different frame patterns. The MPEGs include clips from movies, sports events, home movies, and computer-generated scenes in which the frame rate, resolution and picture size all vary. The actual playtimes varied from ten seconds to over five minutes. Nine of the twelve MPEGs were included in the set of 30 MPEGs used determine the cost equations. The remaining three MPEGs were not used prior to testing. The player was run on a Gateway 9100 laptop with a 266 MHz Pentium II CPU and 64 MB of RAM. The network connection between the client and the http server was made over the Internet between a commercial site and the University of Texas at San Antonio. Each of the MPEGs was played 35 times in five different scenarios (Table 1), and the output of each run for each MPEG was averaged together.

Download and Play	CPU Load	Number of Repetitions
Not Overlapped	1	10
Overlapped	1	10
Overlapped	2	5
Overlapped	3	5
Overlapped	4	5

Table 1: Number of tests conducted

CPU load was adjusted by starting a process that executed floating-point division continuously. CPU load of one means that only essential processes were running. CPU load of two means that one copy of the load-generating program was running, causing the viewer to have CPU access approximately 50% of the time. CPU load of three and four reflect the existence of additional copies of the load-generating program running.

The preliminary results indicate that the SFO was successful both at reducing total playtime and in reducing initial playtime latency. The results also suggest that the models for predicting MPEG playtime are accurate even if the CPU is shared. However, some inaccuracies occur for the decode component, but this will be addressed in subsequent work.

4.1 Total Playtime

On average, the SFO was able to reduce total viewing time by around 20% by overlapping download and play. More importantly, it was also able to reduce the playtime latency by 50% by prefetching as compared with downloading the entire file first. This means a much more pleasant viewing experience (Table 2).

However, in two of the 120 test runs, the player was forced to pause for data. In both cases, the user would have noticed several pauses. All of the delays were .5 second or less, except one pause lasted 2.5 seconds due to an unpredictable network spike.

	Total Download and Playtime			Time Before Playback Began		
Title	w/o Overlap	w/ Overlap	% Decrease	w/o Overlap	w/ Overlap	% Decrease
Blade	30.30	24.53	19.05%	12.16	6.38	47.49%
Blazer	363.81	335.29	7.84%	49.82	21.30	57.26%
Indy	49.00	34.62	29.35%	23.15	8.77	62.12%
lizard1	82.70	60.51	26.83%	35.18	12.99	63.08%
lizard2	57.74	44.23	23.41%	33.98	20.47	39.77%
Reichst	60.37	48.73	19.29%	20.17	8.52	57.75%
sci2	21.25	14.68	30.89%	11.31	4.75	58.04%
Ski	64.25	56.29	12.39%	13.68	5.71	58.22%
sts1	28.08	24.76	11.84%	7.68	4.36	43.26%
sts2	21.26	18.59	12.57%	9.91	7.24	26.97%
sts3	27.74	23.95	13.65%	9.60	5.82	39.43%
Ulysses	19.90	15.06	24.30%	8.85	4.01	54.67%
Average:			19.29%			50.67%

Table 2: Total Playtime Results

4.2 Total Playtime Prediction

We expect that applications such as MPEG will be run in a shared CPU environment, therefore a prediction of playtime as needed by the SFO must take into account machine load. We introduced artificial background machine load for the experiments. For a CPU load of one (no competing jobs), the SFO predicted the playtime with an error of 3% or less for eleven of the twelve MPEGs (Figure 3). The twelfth MPEG had an error of 7.25%. For CPU loads of two and three, the SFO was only slightly less accurate with a maximum error of 9.5.

When the MPEG viewer was not in competition for CPU time, only two of the twelve MPEGs were predicted using the sum of the component times. As the number of processes competing for CPU time increased, so did the number of MPEG that were predicted using the sum of the component times. Interestingly, the decrease in accuracy of prediction was not related to the fact that more MPEGs were being predicted by the sum of their components. The increased error was spread evenly between the two prediction methods.

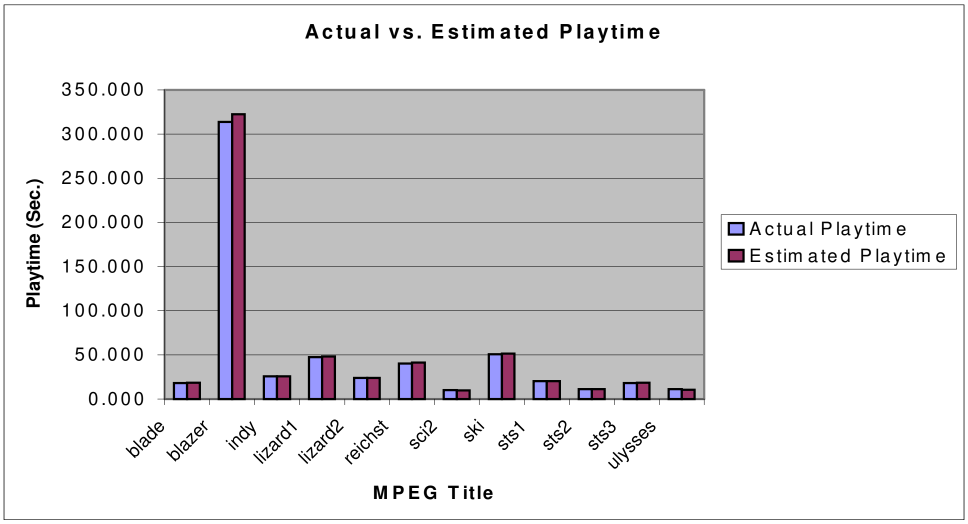

Figure 3: Total playtime prediction – CPU load of one

4.3 Component Time Prediction

Predicting total playtime is performed by predicting the cost of each component of the playtime process. We present the component time prediction results in this section.

4.3.1 Dither Time Prediction

The SFO was able to predict the time required to dither all of the frames in each MPEG with a maximum error of 7%. The cost equation used for estimating dither time was:

*estimated_dither_time = picture_size * 0.000000144324 * estimated_number_of_frames*

4.3.2 Display Time Prediction

For eleven of the twelve MPEGs, the SFO was able to predict the display time within 10%. The SFO generated an error of 14% for the twelfth MPEG. The cost equation used for estimating display time was:

*estimated_display_time = picture_size * 0.00000010283 * estimated_number_of_frames*

Display time is the most likely process to change radically when porting this SFO to other platforms because it depends on many different subsystems of the computer: the CPU, data bus, video card, and video bus.

4.3.3 Decode Time Prediction

Decode time is the most difficult component to predict. Below we show the component cost functions that were empirically derived for decode. The prediction is a function of the type of frame that is being decoded:

*estimated_frame_decode_time = frame_size * 0.0000014 + 0.001624* (for I frames)

*estimated_frame_decode_time = frame_size * 0.00000219 + 0.000533* (for P frames)

*estimated_frame_decode_time = frame_size * 0.00000406 + 0.000478* (for B frames)

We met with modest success in that the SFO predicted decode within 30% for only six of the twelve MPEGs and had a maximum error of 74%. Fortunately, the effects of this error rate were offset by the fact that decode is normally less than 30% of the total MPEG playtime. While it is not difficult to predict the CPU cycles required to decode a frame [2], it is more challenging to estimate the wall clock time required to decode a frame. The principle reason for the prediction inaccuracy is that the viewer mixes the decode process with disk I/O. An 80 KB buffer is filled whenever the decode process runs out of data irrespective of frame boundaries. This can cause variance in the frame decode time data. A more accurate decode time prediction requires that disk I/O be included in the model. This could be treated as a fifth step in the MPEG play process and have its own time estimate along with decode, dither and display.

The cost equations were computed by off-line benchmarking a set of MPEG files. It should be possible for the SFO to dynamically learn the rate at which MPEGs are played on its platform as opposed to having to collect data off-line to calculate the cost equations. As the SFO is used, it would gather statistics and produce cost equations on its own.

5.0 Summary and Future Work

We have described the application of the Smart File Object concept to an MPEG player to improve performance. The MPEG SFO was able to predict the playtime for any given MPEG file with a 10% error and use this information to effectively overlap the download and playback process. The result was a 50% reduction in viewer time latency and a 20% overall decrease in total viewing.

We are planning to explore future work in a number of areas: improving SFO performance and functionality, experimentation in wireless and QoS networks, server-side optimizations, and additional applications. Future performance and functionality improvements include minimizing disk I/O by the downloading the MPEG file in memory rather than on disk, making the viewer into a Netscape plug-in, and adding more end-user control. Memory caching would not only improve the performance of the viewer but should make also decode prediction more accurate. Additional end-user control would allow the user to express preferences such as smooth play or minimum latency. If the user wants smooth play, then the SFO could delay the beginning of playback in order to minimize the likelihood of pauses. If the user wants to preview an MPEG for content, a more aggressive start time will save time but surely cause pauses near the end of the MPEG. This may be perfectly acceptable if the user is just browsing the MPEG.

The SFO is designed to work on the client. A natural extension to this idea would be to put an SFO on the server. The server-side SFO would provide many opportunities for optimization not possible on the client side. Examples include dropping frames or compression.

If the client is falling behind on data requests, then the server-side SFO could drop frames and let the client repeat frames as necessary. Dropping a few frames each second would be less noticeable than a singe long pause waiting for data. Finally, we plan to develop SFOs for other applications and file types such as audio files and players and data-intensive metacomputing applications.

7.0 References

[1] T.E. Anderson, "Serverless Network File Systems*," Proceedings of the 15th ACM Symposium on Operating Systems*, 1998.

[2] A.C. Bavier, A.B. Montz, and L.L. Peterson, "Predicting MPEG Execution Times*,*" *Proceedings of the ACM SIGMETRICS '98,* June 1998.

[3] G. Bust, G. Crowley, J.B. Weissman, et al, "Modeling and Assimilation of Global Ionospheric Characteristics, proposal submitted to ONR MURI program, October 1998.

[4] ISO/IEC 11172-2, International Standard, Information Technology – Coding of Moving Pictures and Associate Audio for Digital Storage Media at up to about 1.5 Mbit/s Part 2:Video, Geneva, 1993.

[5] J.D. Murray, D. Russell, W. Vanryper, *Encyclopedia of Graphics File Formats,* Second edition, O'Reilly and Associates Inc., Sebastopol, 1996.

[6] H. Schulzrinne, A. Rao, and R. Lamphier," *Real Time Streaming Protocol* (*RTSP*), RFC 2326.

[7] M. Spasojevic and M. Satyanarayanan, "An Empirical Study of a Wide-Area Distributed File System," *ACM Transactions on Computer Systems*, Vol. 14, No. 2, May 1996.

[8] J.B. Weissman, "Smart File Objects: A Remote File Access Paradigm," to appear in the *Sixth Workshop on I/O in Parallel and Distributed Systems* (IOPADS), May 1999.

[9] R. Wolski, "Forecasting Network Performance to Support Dynamic Scheduling," Proceedings of *the Sixth IEEE International Symposium on High Performance Distributed Computing*, August 1997.