

# Using Software Feedback Mechanism for Distributed MPEG Video Player Systems

Kam-yiu Lam<sup>1</sup>, Chris C.H. Ngan<sup>1</sup> and Joseph K.Y. Ng<sup>2</sup>

Department of Computer Science<sup>1</sup>  
City University of Hong Kong  
83 Tat Chee Avenue, Kowloon  
HONG KONG  
email: cskylam@cityu.edu.hk

Computing Studies Department<sup>2</sup>  
Hong Kong Baptist University  
224 Waterloo Road, Kowloon  
HONG KONG  
email: jng@comp.hkbu.edu.hk

## Abstract

When a distributed MPEG player system is transmitting MPEG videos over an open network, like the Internet, the system usually has no control on the network traffic. Thus, it has to look into the dynamic nature of the traffic and to adjust itself in order to provide the quality of service guaranteed to the video transmission. Transmitting multiple video streams and the different levels of quality of service (QoS) requirements for different clients further complicates the problem within the video server. In this paper, a priority feedback mechanism for a video server is proposed. With our pre-defined priority mapping functions and the feedback signal from each client, the video server will make adjustment in terms of priority to the server processes in order to support each client which possess a QoS requirement of its own. From our experiments, we find that when a video server is serving multiple clients, the use of a priority feedback mechanism can improve the overall video system performance.

## 1 Introduction

Because of the current advancement and development in multimedia systems and internet applications, a lot of work has been devoted to the design of distributed video player systems for displaying videos over an open network, like the Internet. Since MPEG is becoming a standard for video compression, distributed MPEG video systems will become very popular in the near future. Distributed MPEG video player systems are firm real-time systems. That is, video frames have to be delivered correctly and promptly. In such systems, MPEG compressed video frames are sent continuously from the video server to the clients where the frames are decompressed and displayed. Implicitly, each video frame is associated with a timing constraint on its display time [11]. If the video frames cannot be displayed on time, they are of no use to the clients and have to be dropped. Although the impact of dropping frames is not catastrophic, it may seriously affect the quality of services (QoS) of the video being displayed [7].

To provide a high QoS, the transmission rate of the video frames must be high and predictable. However, when transmitting MPEG videos over an open network, like the Internet, the system always has no control on

the dynamically changing and unpredictable network traffic. Transmitting multiple video streams and the different levels of quality of services (QoS) requirements from different clients further complicate the problem.

From previous studies, a predictable system can be attained by using sophisticated scheduling policies to manage the system resources such as the CPU in the video server, the network, the buffers and the CPU in the client [4, 9]. Although the simplest way is to use some static mechanisms for resource allocation [8], they may not be practical for many distributed multimedia applications, as the mechanisms require priori knowledge on the system environment and the requirements of each client. These required information may not be available. For example, in broadcasting news over the Internet, the client can be anybody who has connected to the news server. The server therefore has no prior knowledge about the clients. Furthermore, the requirements of the clients are not static. They may request videos with different sizes and resolutions and to playback the videos at different speeds. They may even change the playback speeds of the videos while they are being displayed. At the same time, the transmission delay of the frames in the network is unpredictable. It is difficult to predict the among of network jitter and latency in the Internet due to the differences in bandwidth among different links in the network and the dynamic workload of the network.

In order to provide a high QoS, the system has to be adaptive to the changing environment [3, 5]. Since the more controllable component in the system is the video server, a more practical way is to dynamically adjust the server to the changing environment so that the required QoS can be delivered to the clients, or at least the best QoS is provided under the current resources constraints.

To make the system adaptive to the changing environment, one recent suggestion is to use the *software feedback mechanisms* [1, 2, 3]. Although they have been shown to be effective in a single client MPEG video player system – only one client is connected to the server at each time, they are not suitable for multiple-clients systems in which the video server is connected to a number of clients with different requests on the videos. The reason is that the effectiveness of software feedback mechanisms is highly dependent on the length of the feedback cycle which is defined as the time interval between the time a client sends out the feedback signal and the time when the video server generates the response. In a conventional time-sharing environment, like Unix, this period of time depends on how many clients are servicing by the video server. It is unpredictable if the number of clients in the system is not a constant. To make the feedback mechanisms more effective, the length of the feedback cycle should depend on the current status of the clients. The client, in a “poorer” state than the other clients, should be served first so that the length of its feedback cycle can be shortened and hopefully the server will bring the client back to the “normal” status quicker.

In this paper, we have designed and developed a *priority feedback mechanism* for scheduling of server processes, which are serving the clients, in a multiple-client distributed MPEG video player system. Each client generates a feedback signal to the video server indicating its current status. With the QoS requirements and the feedback signal from each client, the video server will make adjustment in terms of priority to the server processes in order to support each client which possesses a QoS requirement of its own. In the design of priority feedback mechanism, one important consideration is that the additional overhead for priority

scheduling must be low [4]. Although different priority scheduling algorithms have been proposed, most of them are very complicated and require the assumptions on the priori knowledge of the processes. Thus, they are not suitable for distributed MPEG video player systems. Instead, a simple priority mapping function is more preferable. From our experiments, we find that when a video server is serving multiple clients such that every client is demanding a different quality of service, the use of the priority feedback mechanism can effectively improve the overall video system performance.

The rest of the paper is organised as follows. Section 2 describes the basic architecture of the distributed MPEG video player system. Section 3 introduces the software feedback mechanism. Section 4 discusses the priority feedback mechanism and the definition of the priority mapping functions. Section 5 discusses the implementation of the distributed real-time MPEG video player system. Section 6 presents the experiment set up and reports the results. Finally, Section 7 is the conclusions of the paper.

## 2 System Architecture

Figure 1 depicts the basic architecture of our distributed MPEG video player system. It consists of a video server, which is connected to a number of clients through an open network. The video server runs on a host as a typical Unix daemon process. It accepts requests from the clients and forks video server sessions (server processes). Each server process is responsible for servicing a client. It retrieves video frames from the secondary storage and put them into the buffer allocated to it. Then, it sends the video frames from the buffer to the client through the network according to the requested playback speed.

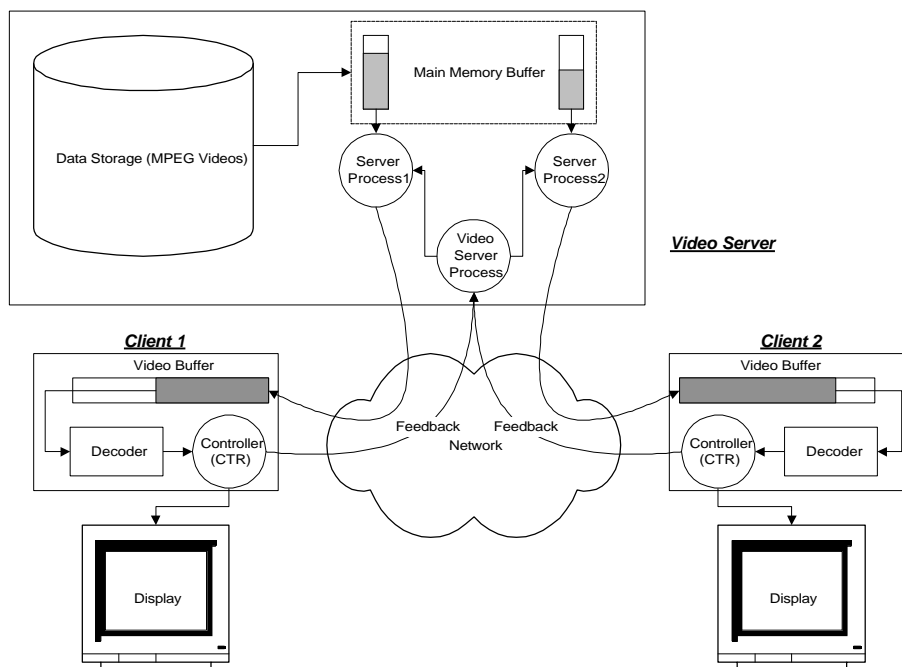


Figure 1: The basic architecture of a distributed MPEG video player system.

Each client consists of a set of collaborating processes: the controller (CTR), the video buffer and the decoder. These processes share the same memory. The video buffer is assigned to a client for temporary storage of the received video frames. During playback, the decoder retrieves the video frames from the video buffer and decompresses the frames. It passes the decompressed frames to the controller which provides an interface for various playback operations such as fast forward and backward, variable speed playback and random positioning.

### 3 Software Feedback Mechanism

Software feedback mechanism [2] is designed to monitor system status. Adjustments are made according to the observed system status to keep of the clients within the pre-specified goal. The basic principles of a software feedback mechanism are shown in Figure 2. It consists of two components: a *feedback filter* and a *control algorithm*. The internal state of the system under control is measured and a feedback signal is then generated. This signal is input into the feedback filter to eliminate transient noise. The control algorithm then compares the output of the feedback filter against a goal specification. Actions will be taken accordingly to adjust the system under control to keep its status within the specification.

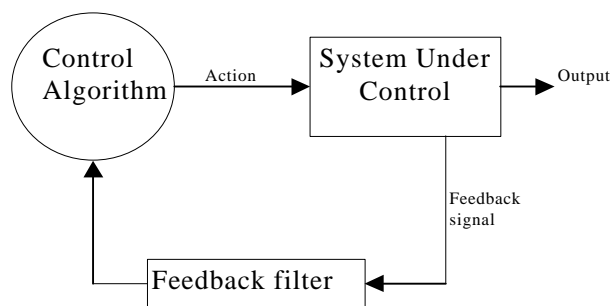


Figure 2: Software Feedback Mechanism

A variety of software feedback mechanisms have been designed to monitor the status of the clients in distributed MPEG video player systems [1]. For example, in applying the software feedback mechanism for buffer management, the buffer level in the client will be monitored. The observed buffer level is used as the feedback signal and input into a control algorithm which generates a signal to the server based on the specified threshold buffer level. According to the received signal, the server will make responses. For example, if the buffer of a client is lower than the threshold value, the server will send the video frames to the client faster to restore the buffer level to the threshold level. If the buffer level is high, the transmission rate of the video frames will be reduced. By ensuring sufficient number of frames in the buffer, it is expected that the display of the video will be smooth.

In the conventional Unix environment, the processes are scheduled in a time-sharing manner. Mostly, each process obtains the same amount of CPU time in each service cycle. If the status of the clients is not the same, e.g. one of the clients may be in a poorer status and has to drop many video frames, treating the server processes equally may result in a great degradation of the performance of the client. It is because the time

required to restore the status of the client to normal will be longer if the number of server processes is larger. For example, at time  $t_1$ , a feedback signal is generated at the time when the buffer level of the client is at 80% of the threshold buffer level. The server process receives the feedback at time  $t_2$  and generates a response at time  $t_3$ . The feedback cycle length is thus equal to  $t_3 - t_1$ . If the number of server processes (number of clients) in the system is large,  $t_3$  can be much greater than  $t_1$ . The buffer level may become even lower at the time when the response is generated. Therefore, it will take a much longer time to restore the level to the threshold value and a lot of frames will be dropped during this period.

The problem will be more serious if the clients are requesting different videos with different playback speeds and resolution. The impact of changes in the environment will be more serious on the client with higher playback speed. Consider a distributed MPEG video player system with a video server connected to  $n$  clients,  $C_1, C_2, \dots, C_n$ . Assuming that the playback speeds requested by the clients,  $C_1, C_2, \dots, C_n$ , are  $R_1, R_2, \dots, R_n$ , respectively and  $R_1 \gg R_2 > \dots > R_n$ . Under the impact of a transient network overload, all the clients are affected. Since the playback speed of  $C_1$  is much higher than the playback speed of  $C_n$ , it is likely that  $C_1$  will be more affected than  $C_n$  (and the other clients). Thus,  $C_1$  will be at a much poorer status than  $C_n$ . For example, at some instance, the buffer level of  $C_1$  is at 0.1 of the threshold level while  $C_n$  is at 0.7 of the threshold level. Since the same amount of time is allocated to each server process in each CPU service cycle, the time required to restore the buffer level in  $C_1$  to the threshold value will be much longer than the time required in  $C_n$ . During the “restoring” period, a lot of frames will be dropped in  $C_1$  and its performance will be more affected than the other clients.

## 4 Priority Feedback Mechanism and Priority Mapping Functions

### 4.1 Priority Feedback Mechanism

To improve the effectiveness of the software feedback mechanism, we introduce a priority scheme to the mechanism in which the priority of a server process will be dynamically adjusted according to the observed status of the corresponding client. To cater for different QoS requirements of the clients, in the assignment of priorities to the server processes, the mechanism will also consider the QoS requirement of individual client. We call this mechanism as priority feedback in which the server process will be assigned to a higher priority if it is servicing a client which is in a poorer status so that the CPU will serve it immediately.

In the design of such a dynamic priority scheduling algorithm, an important consideration is that the additional overhead in priority scheduling must be low. Otherwise, the normal processing of the system will be seriously affected. Although different priority scheduling algorithms have been proposed, most of them are very complicate and require the priori knowledge of the processes. Thus, they are not suitable for the distributed MPEG video player systems in dynamic environment [1]. Instead, a simple priority-scheduling algorithm is more preferable and therefore is adopted in our proposed priority feedback mechanism.

In the design of priority feedback mechanism, we have to determine what are the level of control and the feedback signals. We choose to apply the feedback mechanism in the highest level, between the server processes and the clients. The feedback signal is the QoS provided to the clients. In our implementation, we choose the number of frames being dropped within a fixed period of time<sup>1</sup> as the feedback signal. If the number is larger, the status of the client is poorer and it should receive “more services” from the video server than the clients which are dropping lesser frames. The number of frames being dropped in a fixed period of time  $x$  is input into a *priority mapping function* to calculate the priority of the server process,  $Priority(P)$ .

The calculation of the number of frames being dropped in a period is based on the feedback streams sent from the clients. In a client, its current status is monitored continuously. It sends a feedback stream to the server aperiodically (whenever its status is deviated from the normal) to tell the server what its current status is. The stream contains information on the frames to be dropped in the next  $n$  frames. For example, if the pattern of the feedback stream received from the client is “100111100011” (a zero indicates that the frame has to be dropped). The second, the third, and the eighth to the tenth frames need to be dropped by the server. After the server process has sent out the first frame, it notices that two frames – the 2<sup>nd</sup> and the 3<sup>rd</sup> frame – have to be dropped. Then, the number “two” will be input into the priority mapping function to calculate the priority of the server process for the period of sending the forth frame. After sending the forth frame, the priority of the process will be restored to normal, e.g. zero, as no frame has to be dropped before sending the next frame, the 5<sup>th</sup> frame.

## 4.2 Priority Mapping Function

We have designed three priority mapping functions to cater for different service requirements of the clients based on the requesting playback speed of the clients. They are:

- (1) *Linear* function;
- (2) *Increasing* function; and
- (3) *Decreasing* function.

In the *Linear* function, the priority mapping of a process is directly proportional to the number of frames being dropped:

$$Priority(P) = x + \text{number of frames to be dropped in the period}$$

where  $x$  is the default priority of the process (e.g., the priority of a process when it is at the normal status)

The problem of the *Linear* function is that it treats all the clients the same way according to their number of frames being dropped. If the play speed of the clients in a system is not the same, linear priority mapping will favour the higher play speed clients as they usually drop more frames as compared with the low play speed clients.

---

<sup>1</sup> Other QoS can be used.

In the *Increasing* function, the rate of increase in priority increases with the number of frames being dropped:

$$\text{Priority}(P) = (\text{number of frames to be dropped in the period})^2 / k$$

where  $k$  is a constant.

It has the effect of making the priority feedback mechanism less sensitive to the number of frames being dropped if it is small. It is suitable for the clients with high play speeds as they always have to drop frames (dropping a small number of frames is considered to be a normal status.)

The *Decreasing* function is the reverse of the increasing function in which the rate of increase in priority decreases with the number of frames being dropped.

$$\text{Priority}(P) = \sqrt{k \times (\text{number of frames to be dropped in the period})}$$

It has the effect of making the priority feedback mechanism highly sensitive to dropping frames if the number of frames being dropped is small. It is suitable for the clients with low play speed such that they seldom have to drop frames. Dropping even one frame means that their status is very poor, and the server processes have to be raised up to higher priorities.

By using different values of  $k$ , we can obtain different distributions for the mapping functions and control the intersection point of the curves. In order to cater for the different requirements of the clients, a mixed priority mapping method is used in which each client has its own priority mapping function, which may be increasing or decreasing. If the play speed required by the client is high, an increasing mapping function will be used for the client. On the other hand, if the required play speed is low, a decreasing mapping function will be used so as to make the performance of the client highly sensitive toward dropping frames. The definition of the threshold level can be based on the nature of the video such as its size and resolution. If the video is a large with high resolution, a lower value may be used. By using the mixed priority mapping method and different values for the threshold level, different requirements of the clients can be served separately.

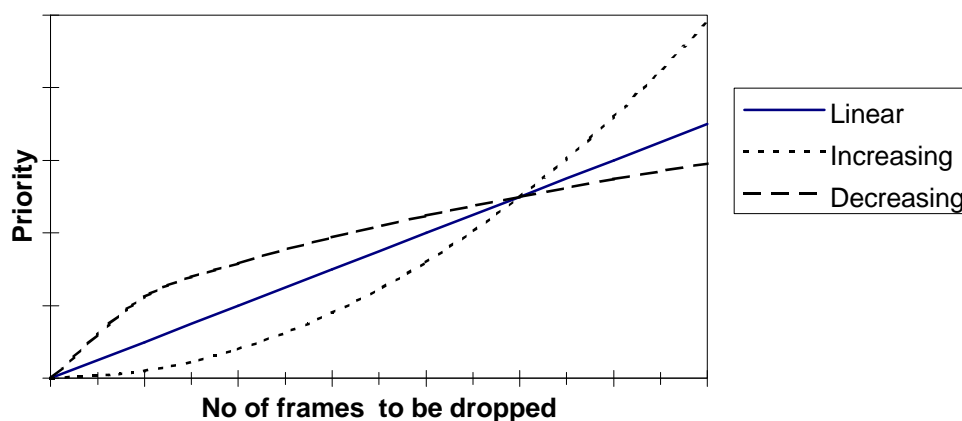


Figure 3: Priority Mapping Functions

## 5 System Implementation

Our distributed real-time MPEG video player system is developed based on the distributed MPEG video player system developed at the Oregon Graduate Institute (OGI) [1]. The main differences are in the video server in which we have implemented the priority feedback mechanism. The implementation of the priority feedback mechanism requires the real-time supports from the underlying operation system. In our system, we choose to use Solaris 2.5 [6]. The reason is that Solaris 2.5 is a general purpose Unix environment and it is a well-known and popular operating system. Although some real-time operating systems may provide better real-time supports for scheduling of processes, these systems are not common to our regular users. When we examine the real-time issues supported by Solaris 2.5, we find that although they are not as sophisticated as the other real-time operating systems, they provide sufficient support for our real-time requirements such as priority and quantum scheduling for real-time threads, fully pre-emptive kernel and bounded dispatch latency.

The player system is implemented in the C programming language. The client is developed based on the Berkeley MPEG decoder. The playback streams supported by the video player are MPEG-1 video [10]. The client sends feedback signals via UDP to the video server and retrieves video frames. Since messages and video frames are sent by UDP packets, big frames are chopped into pieces to comply with UDP packet size limit. The received video packets are stored in the client's buffer. The client buffer process reassembles the video packets back to the frames before passing them to the decoder process. To accommodate the Motif programming interface, a separate process from the controller is used to drive the user interface and the display the video frames.

The priority feedback mechanism is supported by the priority scheduling scheme provided in Solaris 2.5. In Solaris 2.5, 128 levels of priorities, from 0 to 127, can be assigned to the processes. By defaults, 60 priority levels, from 0 to 59, are defined for the real-time processes. A larger number of priorities mean a higher priority. When a process is running, it will only be pre-empted by a higher priority process. In the video server, all the server processes are defined as real-time processes. The priority feedback mechanism assigns a priority to the server process within these 60 priority levels.

## 6 Experiments

A series of experiments have been performed to investigate the performance of the priority feedback mechanism in a distributed real-time MPEG video player system. We have compared the performance of our distributed MPEG player system (with priority feedback) with the distributed MPEG video player developed from OGI (without priority feedback mechanism) which has been shown to perform well in various aspects [1].

### 6.1 Experiment Setup

The basic configuration of the system is as follows:

Video Server: Pentium PC (90 MHz)

Clients: 8 Pentium PC (90 MHz) – (all clients are on the same network)

Network: Ethernet (10Mbps). Server and clients are on the different sub-net

Video stream: A racing car clip with frame size of 320x240.

The total number of frames is 3596 at 30fps (about 2 minutes)

File size is 25,462,128 bytes

Average frame size ratio (I : P : B) is 2.88 : 2.73 : 1

The average frame size is 7431 bytes

Picture group pattern IBBPBBPBBPBB

## 6.2 Performance Measures

In the experiments, we measure the percentage of displayed frames (PD) by a client and the relative improvement of displayed frames (RI). They are defined as:

$$PD = \frac{\sum_{i=1}^n D_i}{n \times D_{total}}$$

where  $n$  is the number of clients.  $D_i$  is the number of displayed frames by client  $i$ , and  $D_{total}$  is the total number of frames in the video file.

$$RI = \frac{PD - PD'}{PD'}$$

where  $PD$  is the percentage of frames displayed with the priority feedback mechanism, and  $PD'$  is the percentage of frames displayed in the distributed MPEG video player system without any real-time issue (Non-RT).

Using the number of displayed frames alone is not a good measure of the QoS and it can be misleading [7]. Consider two playbacks of the same video for two clients. The client with more displayed frames does not necessarily has a better QoS. It is also dependent on how it drops (or displayed) the frames. In most cases, if the client drops the frames more even, its performance should be better as the video looks more smooth. Thus, another common QoS measure is the smoothness of display. One measurement for smoothness is the deviation of presentation jitter from the desired value of zero [1,7]. If we assume that the mapping of logical time (frame number) into the system time is precise and the delay from the client to the video output can be ignored, we can measure the jitter of displaying frames in terms of logical display time. Consider a video frame sequence of  $f_0, f_1, \dots, f_n$  and a playback sequence of these frames are  $f_{j_0}, f_{j_1}, \dots, f_{j_m}$ . At each logical display time  $j$  (where  $j > 0$  and  $j < n$ ), the logical time error is calculated as  $e_j = j - j_k$ , between the expected frame  $f_j$  and the actually displayed frame  $f_{j_k}$  where  $j_k < j$  and  $j_{k+1} > j$ . In [1], based on error sequence,  $E, (e_0, e_1, \dots, e_n)$ , the smoothness,  $S$ , of a playback is defined as:

$$S = \sqrt{\frac{\sum_{e \in E} e^2}{n}}$$

With this definition of smoothness,  $S$ , a lower value of  $S$  indicates a smoother playback. If all the frames can be display at their expected time,  $S$  will be zero. Based on the definition of smoothness,  $S$ , we define the relative improvement in smoothness, SRI, as:

$$SRI = \frac{S' - S}{S}$$

where  $S'$  is the smoothness in the Non-RT.

## 6.3 Experiment Results

### 6.3.1 Impact of Number of Clients

In this set of experiments, the play speeds of the clients are set to be different. One client requests a play speed of 30 fps and the others request a play speed of 12 fps. Table 1 and Table 2 summarise the results when the number of clients for low play speed (12 fps) is increased from 3 to 7.

| Play speed   | PD       |             |          |             | RI     |        |
|--------------|----------|-------------|----------|-------------|--------|--------|
|              | 30 fps   |             | 12 fps   |             | 30 fps | 12 fps |
| # of clients | With PFB | Without PFB | With PFB | Without PFB | ---    | ---    |
| 4            | 29.86%   | 27.86%      | 96.25%   | 94.51%      | 7.18%  | 1.84%  |
| 5            | 30.03%   | 28.19%      | 92.69%   | 90.68%      | 6.53%  | 2.22%  |
| 6            | 30.03%   | 28.19%      | 92.69%   | 90.68%      | 6.53%  | 2.22%  |
| 7            | 30.17%   | 29.19%      | 87.87%   | 85.64%      | 3.35%  | 2.6%   |
| 8            | 28.64%   | 25.56%      | 91.83%   | 78.97%      | 12.05% | 16.28% |

Table 1. The number of displayed frames and relative improvement when the number of clients with low play speed is varied.

| Play Speed   | Smoothness (S) |             |          |             | SRI    |        |
|--------------|----------------|-------------|----------|-------------|--------|--------|
|              | 30 fps         |             | 12 fps   |             | 30 fps | 12 fps |
| # of clients | With PFB       | Without PFB | With PFB | Without PFB | ---    | ---    |
| 4            | 1.574          | 1.670       | 0.528    | 0.662       | 6.10%  | 25.37% |
| 5            | 1.669          | 1.831       | 0.435    | 0.531       | 9.71%  | 22.01% |
| 6            | 1.648          | 2.324       | 0.670    | 0.875       | 41.02% | 30.60% |
| 7            | 2.002          | 2.469       | 0.998    | 1.109       | 23.33% | 11.12% |
| 8            | 2.34           | 3.180       | 0.683    | 1.350       | 35.90% | 49.40% |

Table 2. The smoothness and relative improvement when the number of clients with low play speed is varied.

Although the results are fluctuated due to the unpredictable network traffic during the experiments, it can be seen from Table 1 that the number of displayed frames in the client at high play speed (30 fps) with priority feedback is consistently larger than the one without priority feedback. Although the improvement in PD is

small, the relative improvement is much larger and is in the range of 3% to 12%. The small improvement in PD is due to the high play speed. At a high play speed, the number of dropped frames will be unavoidably large due to the higher demand on the system resources. The improvement with priority feedback is more significant if we look at the smoothness of the playback which is shown in Table 2. A relative improvement of up to 40% is observed in SRI with priority feedback. With the used of the priority feedback mechanism, the status of the high play speed client can be closely monitored. Whenever they are deviated from their normal status, responses will be generated and served immediately so that their status can be restored to normal in a short period.

As depicted in Table 1, the number of displayed frames (PD) in the clients with low play speed (12 fps) is also larger with priority feedback than without priority feedback. The relative improvement (RI) is also consistent although is marginal in most cases. From the results, we can see that the clients with low play speed also benefit from the priority feedback mechanism as well. Assigning a higher priority to the high play speed client, which drops more frames, does not affect the performance of the low play speed clients. If we look at Table 2, we can see that the improvement in smoothness is generally greater in the low play speed clients than in the high play speed client. Since their play speed is low, under most of the time, the frames can be displayed in time and do not need to be dropped. A small drop in number of frames can seriously affect their performance. With the priority feedback mechanism, the deviation from normal status can be detected and rectified immediately.

### 6.3.2 Impact of Various Play Speeds

In our first set of experiments, we emphasis on whether giving more attention to the high play speed player will affect the performance of the lower play speed clients. Hence, we have only one high play speed client and a maximum of 7 low play speed clients. In this set of experiments, the system has eight clients with play speed of 6, 12, 18, and 30 fps. For each play speed, we have two clients. The reason to have this set up is to investigate the problem on how the priority feedback mechanism will affect the clients at various play speeds. The experiment results are presented in Table 3.

| Play Speed | PD       |             | Smoothness (S) |             | RI     | SRI    |
|------------|----------|-------------|----------------|-------------|--------|--------|
|            | With PFB | Without PFB | With PFB       | Without PFB |        |        |
| 6          | 96.94    | 95.56       | 0.64           | 0.77        | 1.44%  | 20.31% |
| 12         | 88.89    | 76.28       | 1.03           | 1.47        | 16.53% | 42.72% |
| 18         | 48.68    | 47.96       | 1.84           | 1.95        | 1.50%  | 5.98%  |
| 30         | 28.46    | 26.10       | 2.67           | 3.02        | 9.04%  | 13.11% |

Table 3 shows the results with various play speeds of eight clients.

As can be seen in Table 3, consistently with the results from the first set of experiments, the results with priority feedback are consistently better than without priority feedback for all the clients in terms of both

number of frames displayed and smoothness of playback. The improvement in smoothness is up to 40%. This further confirms the believe that the priority feedback mechanism can improve the overall system performance and make the system more adaptive to the changing environment.

### 6.3.2 Comparing the Performance among Different Priority Mapping Functions

We repeat the above two sets of experiments with different priority mapping functions and the results are shown in shown in Figure 4 to 6. Figure 4 and Figure 5 show the smoothness in the high and low play speed clients respectively (the curve labelled as NPFB is performance without priority feedback mechanism). Consistent with the results in Section 6.3, the performance with priority feedback (linear, increasing, decreasing and mixed) is always better than the system without priority feedback. Among the four mapping functions, the mixed priority mapping method produced the best results most of the time as the clients with different service requirements are served with the priority mapping functions specially designed for them.

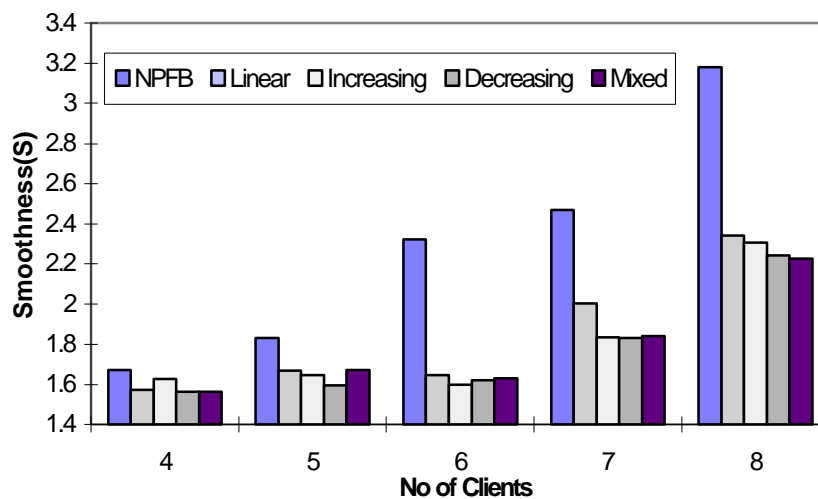


Figure 4: Smoothness of the High Playing Speed Client

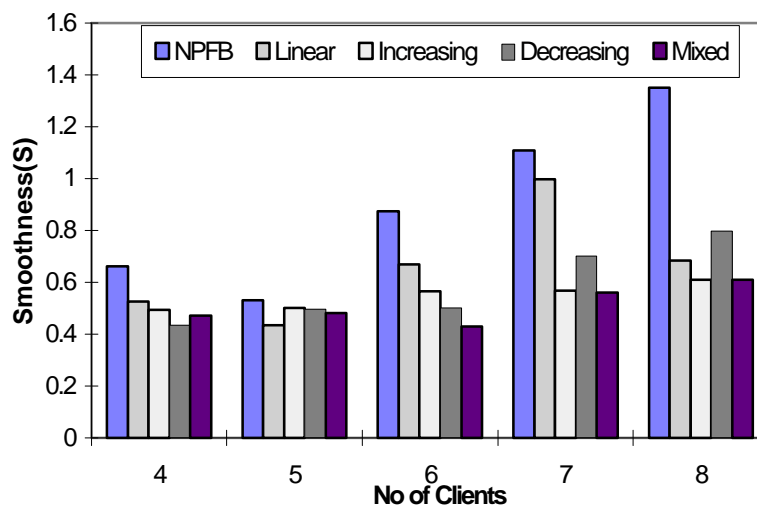


Figure 5: Smoothness of the Low Playing Speed Clients

In Figure 6, it is obvious that the higher the play speed, the poorer the performance. That is not avoidable and it shows at Figure 8. When we compare the performance within the same group (at the same speed), the system without the priority feedback (NPFB) always give a poorer performance, larger smoothness values. Among the different mapping functions, the mixed priority mapping method again produces the best results most of the time. It performs especially well at play speed between 12 – 30 fps.

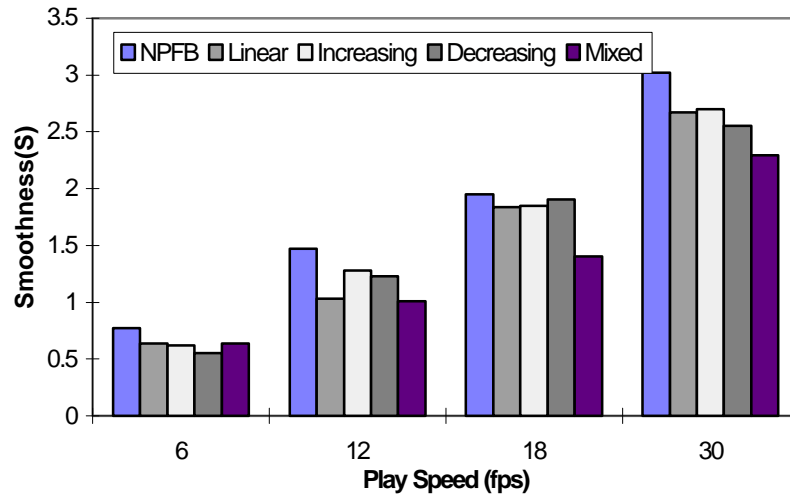


Figure 6: Smoothness at Various Play Speeds

## 7 Conclusions

In this paper, we have proposed a priority feedback mechanism to improve the performance of a distributed MPEG video player system in which the video server is connected to a number of clients and each client may request different videos at different play speeds. In the priority feedback mechanism, the priorities of the server processes are adjusted dynamically in a video server according to the observed status of the clients so that the client, which is in a poorer status, can receive more attention from the server. It is expected that by spending more time with the client, its status can be restored to the normal more quicker. Different priority mapping functions have been suggested for the priority feedback mechanism to cater for different service requirements of the clients.

Experiments have been performed to investigate the improvement of using the priority feedback mechanism and the performance of different priority mapping functions. Although the results are fluctuated due to unpredictable network traffic, significant improvement in the whole system performance are obtained with the priority feedback mechanism in terms of number of displayed frames and smoothness of playback. Amongst the proposed priority mapping functions, the mixed priority mapping gives the best overall performance.

## References

- [1] Shanwei Cen, Calton Pu and Richard Staehli, "A Distributed Real-time MPEG Video Audio Player", in Proceedings of the 5th International Workshop on Network and Operating System Support of Digital Audio and Video (NOSSDAV'95), April 18-21, 1995.
- [2] Calton Pu and R. Fuhere, "Feedback-Based Scheduling: a Toolbox Approach", in Proceedings of 4th Workshop on Workstation Operating Systems, October 14-15, 1993.
- [3] Veronica Baiceanu, Crispin Cowan, Dylan McNamee, Calton Pu and Jonathan Walpole, "Multimedia Applications Require Adaptive CPU Scheduling", In Proceedings of Workshop in Multimedia Resource Management, December 1-2, 1996.
- [4] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment", Journal of ACM, volume 20, number 1, pp. 46-61, 1973.
- [5] Crispin Cowan, Shanwei Cen, Jonathan Walpole and Calton Pu, "Adaptive Methods for Distributed Video Presentation", ACM Computing Surveys, volume 27, number 4, pp. 580-583.
- [6] Bill O. Gallmeister. Programming for Real World POSIX.4. O'Reilly & Associates, Inc., 1995.
- [7] A. Vogel, Brigitte Kerherve, Gregor von Bochmann and Jan Gecsei, "Distributed Multimedia and QOS: A Survey", IEEE Multimedia, volume 2, number 1, pp. 10-18, 1995.
- [8] Shuichi Oikawa and R. Rajkumar, "A Resource-Centric Approach to Multimedia Operating Systems", in Proceedings of Workshop in Multimedia Resource Management, December 1-2, 1996.
- [9] H. Tokuda, "Operating System Support for Continuous Media Applications", Multimedia Systems, Edited by J.F. Koegel Buford, Addison-Wesley Publishing Company.
- [10] L.A. Rowe, K.D. Patel and K. Liu, "MPEG Video in Software: Representation, Transmission, and Playback", in Proceedings of IS&T/SPIE 1994 International Symposium on Elec. Imaging Sci. & Tech., San Jose, 1994.
- [11] Ching-Chih Han, and Kang G. Shin, "Scheduling MPEG-Compressed Video Streams with Firm Deadline Constraints", in Proceeding of the 3rd ACM International Multimedia Conference and Exhibition, November 1995.