An Integrated Push/Pull Buffer Management Method in Multimedia Communication Environments

Sungyoung Lee, Hyun Woo Seung, Tae Woong Jeon

Department of Computer Engineering, Kyunghee University, Seoul, Korea

Department of Computer Science, Seoul Women's University, Seoul, Korea

Abstract. Multimedia communication systems require not only high-performance computer hardware and high-speed networks, but also a buffer management mechanism to process voluminous data efficiently. Two buffer handling methods, push and pull, are commonly used. In the push method, a server controls the flow of data to a client, while in the pull method, a client controls the flow of data from a server. These two buffering schemes can be applied to the data transfer between the packet receiving buffer, which receives media data from a network server, and media playback devices, which play the received media data. However, the buffer management mechanisms at client-side mainly support only one of the push and the pull methods. In other words, different types of playback devices separately use either but not both of the buffer methods. This leads to inefficient buffer memory usage and to inflexible buffer management for the various types of media playback devices. To resolve these problems, in this paper we propose an integrated push/pull buffer mechanism able to manage both push and pull schemes in a single buffer at client-side. The proposed scheme can support various media playback devices using a single buffer space, which in consequence saves memory space compared to the case where a client keeps two types of buffers. Moreover, it facilitates the single buffer as a mechanism for absorbing network jitter effectively and efficiently. The proposed scheme has been implemented in an existing multimedia communication system called ISSA developed by the authors, and has showed good performance compared to the conventional buffering methods in multimedia communication environments.

1 Introduction

A flexible buffer management mechanism is required to process buffer I/Os efficiently, to calculate an adequate buffer size according to the application service and the media type, and to control the buffer over/underflow effectively.

Two buffer handling methods, Server-push (or just push) and Client-pull (or just pull), are commonly used, depending on which side has control over the data flow. In the push method, a server controls the flow of data and periodically transfers appropriate data to clients. Although the push method is suitable for broadcast services, the server must control the data transmission speed (bit-rate) in order to avoid buffer over/underflow at the clients' side. The pull method is a kind of polling method where a client, having control over the data flow, requests data to a server and the server transfers the requested data to the client. Although the client can control buffer over/underflow, the pull method is suitable only for unicast services, not for broadcast services [1, 2, 3].

The two methods can be applied to data transfer not only between network server and clients, but also between the packet receiving buffer, which receives media data from the network server, and media playback devices, which play the received media data. However, most buffer management mechanisms at the clients' side are usually focused on network considerations such as the buffer size control according to end-to-end network situations, and mainly support only one of the push and the pull methods. Consequently, they have some limitations in supporting various media playback devices. Even though some of them support both methods, it is difficult to utilize a variety of devices since they do not provide a unified structure. In an integrated multimedia communication system such as ISSA (Integrated Streaming Services Architecture) [4], various media playback devices cannot be supported if the packet receiving buffer provides either of the push or pull modes. A flexible buffering mechanism is also needed to absorb end-to-end network jitter, in a network environment like Internet with frequent packet loss and unstable bandwidth.

In this paper, we propose an efficient and flexible push/pull buffer management mechanism for the client-side, which readily supports various media playback devices by providing a unified interface for both push and pull modes at the client's packet receiving buffer, and easily absorbs end-to-end network jitter. Using the proposed scheme, we can obviate the need to install an extra stub for buffer passing appropriate to each media device, furthermore better efficiency in memory use can be accomplished by also letting the buffering function absorb network jitter, instead of buffering only.

This paper is organized as follows. We sketch the design architecture of the proposed buffer management scheme in Section 2. The implementation and experimental results are described in Section 3. After briefly reviewing related work in Section 4, we present our conclusions in Section 5.

* This work was partly supported by Grant IFRP-9803-6 from the Ministry of Information and Communication of the Republic of Korea

© Springer-Verlag Berlin Heidelberg 2001
2 Design of the Proposed Scheme
In this section, we explain the operational architecture of the proposed scheme and the structure of the internal buffers, and summarize the buffer management algorithms.

2.1 The Design Model
Figure 1 illustrates the operational architecture of the push/pull buffer management scheme. It shows how the client-side push/pull scheme works with various kinds of media playback devices. The packets transmitted by the Packet Sender from the server through push or pull method are received and translated by the Packet Receiver of the client. Then, the media stream data are sent to the Receiving Buffer, and reproduced by the media playback devices through either the push or pull method initially set. To each of the media playback devices, an appropriate data transmission method is initially assigned. When the transmission session begins, the data stream is transmitted to the corresponding device, according to the method previously set. The Receiving Buffer on the client-side works regardless of the network media transmission methods, since the Packet Receiver receives data from the server through push or pull mode and sends the data to the buffer. The proposed scheme supports both push and pull mode in a single buffer.

2.2 Algorithms
The proposed buffer management scheme consists of 7 algorithms: two for buffer initialization and removal, another two buffering algorithms to absorb network jitter, and three algorithms for the data VO.

Figure 2 shows the procedure for the InitBuffer algorithm. It must first be determined which buffer passing mode (push or pull) the media device supports. Then, an appropriate buffer size is calculated using the following formula:

\[ \text{buffer size} = \text{bitrate of given media} \times \text{buffering time} \times \text{scale factor} \]

where bitrate of given media is the playback speed of the given media (bit/secs), buffering time is the buffering time in seconds for the media, and scale factor is used to calculate an actual buffer size. The scale factor is used to get a "safe" buffer size to prevent overflow of the receiving buffer, and is usually greater than 1.0. If the factor is 1.0, the actual buffer size is equal to the ideal buffer size. After the buffer size is calculated, memory allocation is executed for the buffer. Then, the StartBuffering algorithm is performed.

The DeinitBuffer algorithm, shown in Figure 3, is used to stop media playback and remove the buffer. If buffering is in process (buffering mode is start), the StopBuffering algorithm is invoked. Then, waiting until all media stream data in buffer are consumed, it frees memory for the buffer.

The StartBuffering algorithm begins by changing the buffering mode to start. Then, the current time is saved to calculate the playback delay time. The push/pull operations are locked until the total size of data in the buffer exceeds the buffering size.

The StopBuffering algorithm updates playback delay time by adding additional playback delay time. It also unlocks push/pull operations, and changes buffering mode to stop.