멀티미디어 환경에서 클라이언트와 서버간 양방향 Push/Pull 전송기법

천주일, 이승룡

경희대학교 전자계산공학과

A Bidirectional Push/Pull Transmission Method Between Client and Server in Multimedia Environments

Jooihl Chon, Sungyoung Lee

Dept. of Computer Science, Kyung Hee University

요 약

멀티미디어 통신 시스템은 멀티미디어의 특성상 대용량 데이터를 처리하기 위한 버퍼 관리 기법이 요구된다. 버퍼에 있는 데이터를 전달하는 방식에는 데이터를 제공하는 서버가 데이터 흐름을 제어하는 Push 방식과 데이터를 제공받는 클라이언트가 데이터 흐름을 제어하는 Pull 방식이 있다. 그러나 RTP를 사용하는 기존의 멀티미디어 통신 시스템의 경우 서버와 클라이언트 사이에 Push 방식과 Pull 방식 중의 하나만을 제공하기 때문에 다양한 미디어의 재생을 지원하기에는 한계가 있고, 인터렉티브한 통신이 불가능하다. 더욱이, Push/Pull 두 가지 방식을 모두 지원한다 하더라도 단일화된 버퍼 구조를 제공하지 않아 메모리 사용의 낭비가 있다는 단점이 있다. 이를 해결하기 위하여, 본 논문에서는 RTP를 사용하는 멀티미디어 통신 환경에서 각각 단일화된 버퍼구조 구조를 가지고 있는 서버와 클라이언트 사이에 Push 방식과 Pull 방식의 전송방식을 모두를 지원할 수 있는 통합된 Push/Pull 전송 모델을 제안한다. 이 기법은 서버와 클라이언트 사이에서 Peer-to-Peer 개념으로 인터랙티브한 Push/Pull 전송을 지원할 뿐만 아니라 일관된 인터페이스를 지원하는 단일 버퍼 구조를 가짐으로써 시스템의 메모리 효율을 향상시킬 수 있다.

1. 서론

최근의 인터넷은 컴퓨터와 통신기술의 발전으로 인해 동영상, 음성 등과 같은 고 대역폭을 요구하는 멀티미디어 데이터의 온라인 서비스를 가능하게 하였다. 이러한 멀티미디어 통신 서비스에는 VOD(Video On Demand), AOD(Audio On Demand), 화상회의(Video Conference), 인터넷 방송(Internet Broadcasting) 등과 같이 매우 다양한 서비스가 포함되며, 멀티미디어 데이터를 처리하기에 충분한 고성능의 컴퓨터 하드웨어와 고속 통신망 기술을 요구한다.

버퍼에 저장되어 있는 데이터의 전달 방식은 데이터 흐름의 주도권을 누가 가지고 데이터를 전달하는가에 따라 두 가지 방식으로 구분된다. Pull 방식은 서버가 데이터 흐름의 주도권을 가지고 주기적으로 적절한 데이터를 클라이언트로 전송하는 방식으로서, 브로드캐스트 서비스에 적합하다[2].

이와는 반대로 Pull 방식은 클라이언트가 데이터 흐름의 주도권을 가지고 서버에 원하는 데이터를 요청하면 서버는 요청된 데이터를 클라이언트로 전달하는 요청/응답형식을 가지는 일종의 폴링(Polling) 방식으로서, 유니캐스트 서비스에 적합하다. 위와 같이 멀티미디어 통신 시스템의 경우 네트워크 서버와 네트워크 클라이언트 사이에 Push 방식 혹은 Pull 방식을 통해 버퍼에 있는 데이터가 전송된다[3]. 이 중 RTP를 사용하는 전송에서는 Push 또는 Pull중 한가지 방식만을 지원하거나, Push와 Pull 방식을 모두 지원한다 하더라고 단일한 인터페이스로 제공하지 않아 확장이 용이하지 않은 한계를 가지고 있다. 또한 서버의 일방적인 Push방식이나 클라이언트의 주도적인 Pull방식의 전송으로 인하여 서버와 클라이언트(객체와 객체) 사이에 인터렉티브한 통신이 어려운 한계를 가지고 있다. 따라서, 포괄적이고 통합적인 멀티미디어 통신 시스템에서는 서버와 클라이언트간에 Push와 Pull 방식 중 하나만 제공하는 경우에는 인터렉티브한 통신이 어렵고, 다양한 미디어 재생 장치를 지원할 수 없으며, 각각의 메모리를 유지함으로써

메모리 사용의 낭비를 피할 수 없게된다. 기존의 RTP를 확장한 전송 방식에서는 클라이언트 사이드에서만 여러 멀티미디어 디바이스와 Push/Pull 방식이 모두 지원되었다. 하지만 클라이언트에서 Receiving Buffer와 디바이스간의 전송만을 지원하기 때문에 서버와 클라이언트사이에 통신은 기존의 방법과 동일하다[4].

따라서 본 논문에서는 클라이언트에서 Receiving Buffer와 디바이스뿐만 아니라 RTP상에서 서버와 클라이언트간의 패킷 수신 버퍼가 Push 방식과 Pull 방식의 데이터 전송 방식을 모두 지원하여 사용자에게 제공하는 미디어의 QoS를 높일 수 있고 각 개체들 간의 상호 통신이 가능한 분산 환경에서의 RTP를 위한 Push/Pull 전송 기법을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 Push/Pull 방식의 전송 기법과 연관된 기존의 관련 연구를 살펴보고, 3장에서는 제안한 전송 기법의 모델을 제시하며, 4장에서는 제안한 전송 기법 알고리즘을 설명하고, 마지막으로 5장에서 결론을 내린다.

2. 관련연구

서버와 클라이언트 사이의 전송 기법과 관련된 연구는 매우 다양한 방면에서 많이 있었으나, RTP에서 데이터 전달 방식과 연관된 Push와 Pull 방식의 전송 기법에 관한 연구는 그리 많지 않다. 이번 장에서는 본 연구와 관련된 기존의 연구 작업에 대해 살펴보겠다.

Sun Microsystems사가 연구 개발하고 있는 JMF (Java Media Framework)에서는 *PushDataSource*와 *PullDataSource*라는 두 가지 인터페이스를 통해 각각 Push 방식과 Pull 방식을 위한 전송 기법을 제공한다. JMF는 멀티미디어 데이터 처리를 위한 단일한 구조와 프로그래밍 API를 제공하는 포괄적인 멀티미디어 플랫폼으로서, 자바를 이용한 네트워크 및 로컬에서의 멀티미디어 데이터의 재생 기능 등을 지원한다. 하지만 Push와 Pull 방식을 단일화된 인터페이스로 제공하지 않는 단점을 지니고 있어 실제 이를 이용한 프로그래밍 하기가 매우 까다롭다[6].

Brown University에서는 Push 방식을 이용해 데이터를 브로드캐스트 서비스하는 시스템에서 Pull 방식을 함께 부가적으로 지원할 수 있는 방법과 그에 따른 버퍼 관리 기법을 제안하였다. 이 연구에서 제안한 기법은 기존의 Push 방식의 브로드캐스트 시스템에 추가적으로 Pull 방식을 지원할 수 있는 서버와 클라이언트간의 독립적인 채널을 설정하고 이 채널을 통해 클라이언트의 요청을 전달함으로써 Pull 방식을 지원하였으며, 서버에 브로드캐스트 서비스 하고자 하는 데이터와 클라이언트에서 요청된 데이터를 멀티플렉싱하기 위한 Push/Pull MUX(Multiplexing) 버퍼를 두고 있다. 그러나 이것은 네트워크 서버와 클라이언트 사이에서 Push 방식과 Pull 방식의 동시 지원에 초점을 맞춘 것으로서, 본 연구에서 제안한 기법과는 기본 개념이 다르며 여기서 Pull 방식은 단지 기존의 Push 방식을 통한 브로드캐스트 서비스의 단점을 보완하기 위해서 제공된다[7].

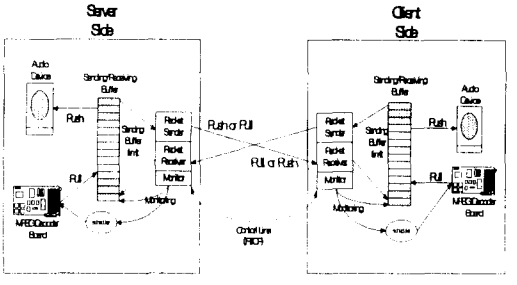
3. 제안한 Push/Pull 전송 모델

이번 장에서는 제안한 분산 환경에서 RTP를 위한 Push/Pull 전송 기법의 동작 구조에 대해 설명한다.

분산 환경에서 RTP를 위한 Push/Pull 전송 기법의 전체적인 동작 구조는 다음 [그림 1]과 같이 이루어지며, 실제 서버와 클라이언트 사이에 데이터가 전송되는 동작 구조를 표현한 것이다.

[그림 1]에서 서버 측의 패킷 송신자는 RTP의 초기 설정을 Push 전송 방식이나 Pull 방식을 결정하여 전송한다. 클라이언트에서의 패킷 수신자를 통해 수신되어 해석된 후 수신 버퍼로 전달되며, 수신 버퍼로 전달된 미디어 스트림 데이터는 설정된 미디어 재생 장치로 재생된다. 즉 Push/Pull 전송 기법은 초기에 서버에서 클라이언트에게 미디어 스트림 데이터를 전송할 때 Push 방식을 이용하여 전달하고, 클라이언트에서 버퍼를 모니터링하여 버퍼의 상황에 따라 서버에게 데이터를 요청하는 Pull 방식이 사용되는 기법이다.

RTCP 프로토콜은 하나의 RTP 세션에서 RTP와 같이 사용되며, QoS 정보를 모니터링하고, 진행중인 세션에 참가한 사용자들의 정보를 전달하는 것을 목적으로 한다. 즉 RTP는 전송기능만을 가지며 RTCP가 세션에 대한 제어 기능을 수행한다. 하지만 RTCP는 최소한의 세션 제어 기능만을 가지고 있다.



[그림 1] 분산 환경에서 RTP를 위한 Push/Pull 전송 기법

RTCP에는 APP 패킷이 있어 응용프로그램이 사용하는 패킷을 정의 할 수 있게 되어 있다. 본 논문에서는 클라이언트의 버퍼를 모니터링하기 위해 RTCP의 APP 패킷을 이용한다[8].

서버가 실행되면 서버의 전송 패킷의 알맞게 전송 패킷 스케줄러를 설정하고, 클라이언트에게 전송될 미디어의 전송속도를 전달한다. 클라이언트 프로그램은 전달받은 전송속도에 적합한 초기 버퍼

크기를 설정하게 된다. 버퍼의 크기는 전송되어지는 프래임과 버퍼링 시간의 곱으로 계산된다. 서버에 접속하면 서버는 설정되어 있는 전송 방식을 통하여 클라이언트에게 미디어 데이터를 전송한다. 클라이언트는 초기 실행 시 버퍼링 시간만큼 버퍼에 수신된 미디어 데이터를 버퍼링한 후 재생하게 된다. 재생되는 동안 클라이언트는 버퍼를 모니터링하여 클라이언트 버퍼의 상태를 RTCP의 APP 패킷을 이용하여 서버에 보낸다. 클라이언트는 버퍼의 상태에 따라 버퍼가 많이 비어 있으면 서버에게 더 많은 미디어 데이터를 요구하고, 버퍼의 한계에 다다르면 서버에게 미디어 데이터의 전송을 늦추도록 요구함으로써 버퍼링되는 데이터를 항상 일정하게 유지할 수 있다.

또한 Peer-to-Peer 개념으로 서버와 클라이언트가 서로 대칭적으로 구성되어 동시에 미디어의 통신이 가능하여 인터렉티브한 통신이 가능하다.

4. Push/Pull 전송 알고리즘

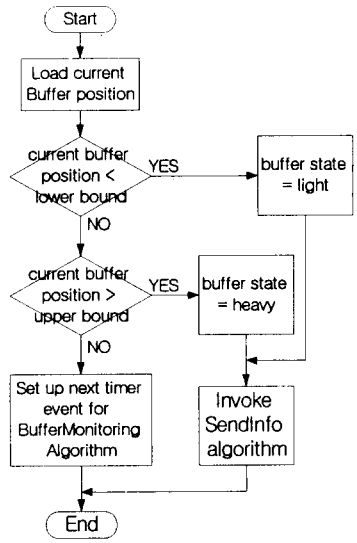
기존 멀티미디어 통신시스템을 위한 클라이언트에서의 Push/Pull 버퍼 관리 기법은 클라이언트 측에서만 수신 버퍼를 Push/Pull 전송 방식을 모두 지원하는 기법으로 서버에서 클라이언트에게 전송되는 방식은 Push 전송방식을 사용하고 있다. 본 연구는 이러한 전송 방식을 확장하여 클라이언트 사이드의 수신 버퍼가 Push/Pull 전송 방식을 모두 지원할 뿐만 아니라, 서버와 클라이언트 사이에도 Push/Pull 전송 방식을 모두 지원하도록 구성하였다. 다음 [표 1]은 제안한 Push/Pull 전송 기법을 위한 알고리즘을 정리한 것이다.

[표 1] Push/Pull 전송 기법을 위한 알고리즘

InitBuffer	버퍼를 초기화하는 알고리즘
DeinitBuffer	버퍼를 소멸시키는 알고리즘
StartBuffering	버퍼링을 시작하는 알고리즘
StopBuffering	버퍼링을 중단하는 알고리즘
AddData	버퍼에 데이터를 추가하는 알고리즘
PushData	데이터를 Push 방식으로 전달하는 알고리즘
PullData	데이터를 Pull 방식으로 전달하는 알고리즘
Buffer-Monitoring	Sending/Receiving 버퍼를 모니터링하는 알고리즘
SetBiterate	전송 속도를 조절하는 알고리즘
SendInfo	Pusll 방식을 이용하여 서버에 필요한 데이터를 요구하거나 전송 속도의 조정을 요청하는 알고리즘

위의 알고리즘 중 *InitBuffer, DeinitBuffer, StartBuffering, StopBuffering, AddData, PushData, PullData* 알고리즘은 기존의 RTP를 확장한 방법을 그대로 사용하였다[4].

BufferMonitoring 알고리즘은 Sending/Receiving 버퍼를 모니터링하여 현재 버퍼의 상태에 따라 오버플로우나 언더플로우 상황을 판단하여 이에 대처하는 역할을 담당한다. *BufferMonitoring* 알고리즘은 일정시간이 지나면 실행되어 버퍼 상황을 읽어온다. 버퍼 모니터링을 통하여 얻을 수 있는 값은 두 가지로 구분된다. 먼저, 버퍼 상태가 lower bound 아래로 버퍼링이 되어 있으면 클라이언트에서 멀티미디어 데이터를 재생하는데 언더플로우로 인하여 QoS가 줄어들 수 있다. 이때 버퍼의 상태를 서버에 보내 더 많은 데이터를 보내도록 *SendInfo* 알고리즘을 이용하여 서버에 요청하게 된다. 반대로 버퍼가 upper bound를 넘도록 버퍼링되면 오버플로어가 발생되어 데이터를 버리게 되는 상황이 발생하므로 서버에 정보를 보내어 전송되는 데이터의 양을 줄여서 오버플로우가 발생하지 않도록 대처하게 된다. *BufferMonitoring* 알고리즘을 순서도로 표현하면 다음 [그림 2]와 같다.



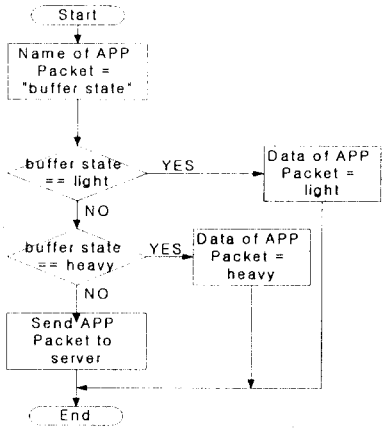
[그림 2] BufferMonitoring 알고리즘의 순서도

SendInfo 알고리즘은 RTCP의 APP 패킷에 서버의 정보나 클라이언트 버퍼의 정소를 전송하는데 사용되어 진다. APP 패킷의 이름에 따라 다음과 [표 2]와 같은 역할을 하게 된다.

[표 2] APP 패킷의 이름에 따른 역할

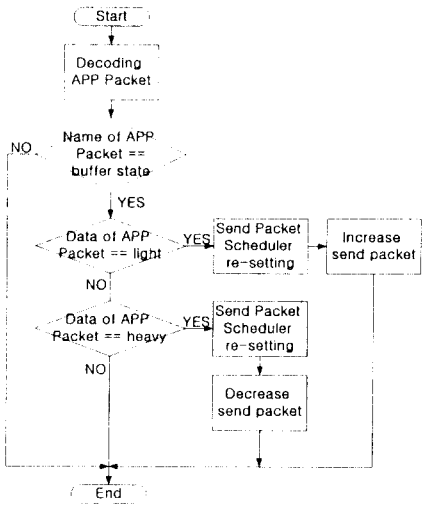
buffer state	클라이언트의 Receiving buffer의 상태를 나타낸다.
bitrate	서버에서 전송되는 미디어의 속도를 나타낸다. 이 값은 클라이언트의 Receiving buffer의 크키를 결정하는 요인으로 작용한다.

*SendInfo*는 *BufferMonitoring* 알고리즘에서 판단된 버퍼의 상황을 RTCP의 APP패킷을 이용하여 서버에 보낸다. 다음 [그림 3]은 *SendInfo* 알고리즘을 순서도로 나타낸 것이다.



[그림 3] *SendInfo* 알고리즘의 순서도

SetBitrate 알고리즘은 전송되어진 APP 패킷을 분석하여 클라이언트 수신 버퍼의 상태에 따라 전송되는 미디어 패킷을 조절하는 알고리즘이다. 다음 [그림 4]는 *SetBitrate* 알고리즘을 순서도로 나타낸 것이다.



[그림 4] *SetBitrate* 알고리즘의 순서도

먼저 클라이언트로부터 전송되어진 APP 패킷을 분석하여 버퍼 상태를 나타내는 것인지를 판단한다. APP 패킷은 응용프로그램에서 다른 용도로도 사용될 수 있기 때문에 버퍼 상태를 나타내는 것인지 먼저 판단을 하게 된다. 버퍼 상태를 나타내는 것이면, 현재 버퍼의 상태가 light상태인지 heavy 상태인지를 파악한다. light는 언더플로우가 발생할 수 있는 상황을 나타내고, heavy는 반대로 오버플로우가 발생할 수 있는 경우를 나타낸다. light 상태면 전송 패킷 스케줄러의 전송 패킷을 전송속도에 맞게 많이 만들도록 설정을 하고 전송되는 데이터 양을 증가시킨다. heavy 상태면 반대로 스케줄러의 설정을 적게 만들도록 설정하고 전송되는 데이터 양을 줄이게 된다.

5. 결론

본 논문에서 제안된 멀티미디어 환경에서 통합된 Push/Pull 전송 기법은 버퍼를 모니터링 하여 버퍼 오버플로어의 위험이 있을 경우 RTCP를 통하여 서버에게 전송하는 패킷의 양을 줄이라는 명령을 내리고, 언더플로어의 위험이 있을 경우 패킷의 전송량을 많게 요구함으로써 이를 방지할 수 있다. 또한, 서버와 클라이언트 각각에 일관된 인터페이스를 지원하는 단일 버퍼 구조를 가짐으로써 시스템의 메모리 자원의 낭비를 방지할 수 있다. 제안된 구조는 서버와 클라이언트가 동등한 역할을 수행하는 Peer-to-Peer 개념으로 설계되어 화상회의나 VOD 같은 인터렉티브한 멀티미디어 통신환경에 적합하다.

[참고문헌]

[1] M. Franklin, and S. Zdonik, "Data In Your Face: Push Technology in Perspective", In Proc. of the ACM SIGMOD International Conference on the Management of Data, Vol. 27, No. 2, pp. 516-521, June, 1998.

[2] S. S. Rao, H. M. Vin, and A. Tarafdar, "Comparative Evaluation of Server-push and Client-push Architectures for Multimedia Servers", In Proc. of the 6th International Workshop on Network and Operating System Support for Digital Audio and Video, April, 1996.

[3] J. P. Martin-Flatin, "Push vs. Pull in Web-Based Network Management", In Proc. of the Integrated Network Management VI, pp. 3-18, May, 1999.

[4] 정찬균, 이승룡, "통합 스트리밍 프레임워크의 설계", 제 11회 한국 정보처리학회 춘계 학술발표 논문집, pp. 319-322, 1999년 4월.

[5] 정찬균, 김형일, 홍영래, 임익진, 이승재, 이승룡, 정병수, "분산 스트리밍 시스템 설계", 제 4회 한국 멀티미디어 학회 추계 학술발표 논문집, pp. 338-343, 1999년 11월.

[6] Sun Microsystems, Java Media Framework API Guide, September, 1999, http://java.sun.com/products/java-media/jmf/index.html

[7] S. Acharya, M. Franklin, and S. Zdonik, "Balancing Push and Pull for Data Broadcast", In Proc. of ACM SIGMOD Conference, Vol. 26, No. 2, pp. 183-198, May, 1997.

[8] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, RTP: A Transport Protocol for Real-Time Applications, IETF RFC 1889, January 1996.