멀티미디어 데이터를 위한 피어-투-피어 전송모델

준회원 신 광 식, 윤 완 오, 정 진 하, 정회원 최 상 방*

Peer-to-Peer Transfer Scheme for Multimedia Partial Stream using Client Initiated with Prefetching

Kwang-sik Shin *Associate Member*, Wan-oh Yoon, Jin-ha Cheong,
Sang-bang Choi* *Regular Members*

요 약

DSL 이나 케이블 모뎀 등과 같은 광 대역 접속기술이 보급됨에 따라 멀티미디어 기반 컨텐츠를 선호하는 사용자 수가 증가되었다. 반면 서버 네트워크 자원은 늘어나는 사용자의 요청 수에 맞춰 끊임없이 확충 할 수는 없다. 그러므로 서버 네트워크 자원을 얼마나 효율적으로 사용하는가는 서비스의 품질을 결정짓는 중요한 요인이 된다. 본 논문은 클라이언트 자원을 이용하여 서버자원이용을 줄여주는 CIWP (Client Initiated With Prefetching) 모델을 기본으로 부분 스트림에 대해서는 P2P 모델을 적용함으로써 멀티캐스트 채널 이외의 추가적인 서버자원소비를 줄이는 새로운 모델을 제시하였다. 특히 threshold based multicast 알고리즘을 적용하여 모든 사용자는 서버로부터 대부분의 데이터를 전송받고 일부 앞부분의 데이터에 대해서만 다른 사용자를 통하여 받게 된다. 또한 사용자간의 데이터 전송은 선행 사용자의 서비스 시간을 초과하지 않도록 한다. 사용자간의 Peer-to-Peer 전송은 데이터전송 주체간의 네트워크 자원에 제한을 덜 받도록 ISP 단위로 그룹으로 나누어 그룹 내에서만 데이터 전송을 허용한다. 해석적 방법을 사용하여 추가로 요구되는 클라이언트 측 자원과 이를 통해 절감되는 서버 네트워크 자원을 분석하였다. 또한 시뮬레이션을 통해 지연시간과 전체 요청자중 지연 비율을 통해 성능 향상을 검증하였다. 결과적으로 제시된 모델은 서버 네트워크 자원이용을 비교할 때 35%의 대역폭 절감 효과를 얻을 수 있다.

Key Words : Multimedia communication, peer-to-peer, stream, video-on-demand, multicast.

ABSTRACT

Client requests have increased with the improvement of network resources at client side, whereas network resources at server side could not keep pace with the increased client request. Therefore, it is primary factor of the Qos that efficiently utilize network resources at server side. In this paper, we proposed a new model that peer-to-peer transfer scheme for partial multimedia stream based on CIWP which it decrease server network bandwidth by utilizing client disk resources saves additional server network resources. Especially, adapting Threshold_Based Multicast scheme guarantees to do that data transfer within clients never exceed service time of previous peer by restriction of which data size transferring from previous peer less than data size transferring from server. Peer-to-peer transfer within clients is limited in same group classified as ISPs. Our analytical result shows that proposed scheme reduces appling network resources at server side as utilizing additional client disk resource. Furthermore, we perform various simulation study demonstrating the performance gain through comparing delay time and proportion of waiting requesters. As a result, when we compared to Threshold_Based Multicast scheme, the proposed scheme reduces server network bandwidth by 35%.

* 인하대학교 전자공학과 컴퓨터구조및 네트워크연구실(sangbang@inha.ac.kr)
논문번호 : 040006-0106, 접수일자 : 2004년 1월 6일
* 본 논문은 정보통신부의 정보통신연구진흥원에서 지원하고 있는 정보통신 기초기술연구지원사업의 연구결과입니다.(04-기초-071)

Ⅰ. 서 론

인터넷이 생활화되고 대부분의 가정에 DSL 이나 케이블 모뎀과 같은 광 대역접속기술이 보급됨에 따라 사용자들은 기존의 텍스트 기반의 컨텐츠에서 멀티미디어 기반의 컨텐츠를 선호하는 경향을 보이고 있다. 특히 VOD 나 실시간 방송과 같은 스트리밍 미디어 서비스는 고 대역 네트워크 기술의 출현으로 인해 가장 촉망받는 서비스 사업 분야로 떠오르고 있다. 사용자 측의 네트워크 환경개선으로 인해 멀티미디어 서비스에 대한 사용자의 요청은 많아졌으나, 서버 네트워크 자원은 이에 맞춰 크게 개선되지 못하였다 (여기서 서버 네트워크 자원은 서버로부터 사용자가 속한 ISP까지 경로내의 모든 네트워크 자원을 포함한다). 서비스 도중 끊어지거나 일시적으로 정지하는 것과 같은 현상이 생기지 않도록 보장하기 위해서는 사용자의 요청이 들어오기 전에 서버 네트워크 자원을 충분히 확보하고 있어야 한다. 그러나 서버 네트워크 자원은 사용자 요청의 증가에 상응하도록 무한정 늘릴 수 없을 뿐만 아니라 서버 네트워크 자원은 상대적으로 매우 비싸기 때문에 한정된 서버 자원을 얼마나 효율적으로 이용하느냐가 미디어 서비스의 품질을 결정짓는 중요한 요인이 된다.

그동안 서버 자원의 효율적인 사용을 위한 다양한 연구가 진행되어 왔는데, 본 논문에서는 오브젝트 단위로 서버 자원을 할당해주는 멀티캐스트를 이용한 방법에 초점을 맞춘다. 멀티캐스트 방식은 다수의 사용자들이 하나의 비디오 스트림을 공유하는 것을 허용함으로써 서버 자원이용의 효율성을 높여준다. 예를 들면, 두 명의 사용자가 동시에 똑같은 비디오에 대해 요청을 보냈을 때 하나의 비디오 스트림을 멀티캐스트 해줌으로써 두 사용자를 동시에 서비스할 수 있으므로, 하나의 비디오 스트림 만큼의 대역폭 사용을 줄여줄 수 있다.

지금까지 연구된 멀티미디어 데이터의 멀티캐스트 방식은 크게 Server-push 와 Client-pull의 두 가지로 분류할 수 있으며 [1], 다시 Server-push는 서버에서 임의로 데이터를 전송하는 방식으로 SI (Server-Initiated)와 SIWP (Server-Initiated-With- Prefetching)으로 분류되고 Client-pull은 사용자의 요청을 받은 이후에 데이터를 전송하는 방식으로 CI (Client-Initiated)와 CIWP (Client-Initiated-With- Prefetching)으로 분류된다.

SI 모델에서 비디오 서버는 주기적으로 비디오 개체를 멀티캐스트 하고, 각각의 사용자는 적절한 멀티캐스트 그룹에 참가함으로써 서비스를 받는다. 이러한 방식은 주기적으로 멀티캐스트 채널이 생성되므로 최대 지연시간은 주기에 따라 보장된다 [6]. SI와 달리 SIWP 모델에서 비디오 개체는 몇 개의 조각으로 나눠지고 개개의 멀티캐스트 그룹을 통하여 주기적으로 멀티캐스트 한다. 사용자는 여러 멀티캐스트그룹으로부터 데이터를 받은 후 이후재생을 위하여 자신의 디스크에 저장해둔다. 비디오개체의 각 세그먼트가 주기적으로 멀티캐스트 되기 때문에 SI와 마찬가지로 최대지연시간이 보장된다 [7]. 한편 앞의 두 모델과 달리 클라이언트의 요청을 받은 이후에 서비스를 시작하는 CI 모델에서는 클라이언트가 특정 비디오에 대한 요청을 한 후 실제 멀티캐스트 될 때까지 기다린다. 채널이 확보되면 서버는 스케쥴링 정책에 따라 rm 사이에 요청한 클라이언트를 묶어 하나의 멀티캐스트 채널로 서비스해준다. 서버 자원이용은 줄여주지만, 최대 지연시간은 보장하지 못한다 [6], [8]. CI 모델에 클라이언트 디스크 이용을 추가시킴으로써 CIWP 모델에서는 동일한 비디오를 서로 다른 시간에 요청하는 두 사용자는 요청시간과 관계없이 멀티캐스트 채널공유를 통해 바로 서비스 받을 수 있다. 이미 전송된 부분에 대해선 별도의 유니캐스트 채널을 통해 전송 받고 멀티캐스트 채널을 통한 데이터는 이후재생을 위해 디스크에 저장해둔다 [9], [11].

SIWP 와 CIWP는 서버로부터 멀티캐스트채널을 이용하여 보내지는 데이터 중 서비스를 시작할 당시에는 필요하지 않지만 추후에 사용하게 될 데이터를 클라이언트의 디스크에 저장해두었다가 해당 부분을 재생할 때가되면 서버로부터 데이터를 전송 받는 것이 아니라 디스크에 있는 데이터를 읽어서 재생한다. 이처럼 SIWP와 CIWP는 클라이언트 자원을 이용해서 서버 자원이용을 줄여준다는 점에서 비슷한 특징을 갖는다.

본 논문은 CIWP 모델을 기본으로 하여 멀티캐스트 채널을 통한 데이터를 저장해두고 현재 사용자가 요청하기 이전에 이미 전송된 데이터에 대해서는 현재 사용자보다 앞서서 서비스를 받고 있는 사용자로부터 P2P 전송을 이용하여 데이터의 일부분을 받는 새로운 모델을 제시하였다. 여기서 P2P 모델이란 서버와 클라이언트간의 데이터 전송하는 기존의 네트워크 구조와 달리 각 클라이언트가 서버의 기능을 동시에 수행함으로써 별도로 서버를 필요로 하지 않는 클라이언트간의 데이터 전송 모델을 말한다 [11].

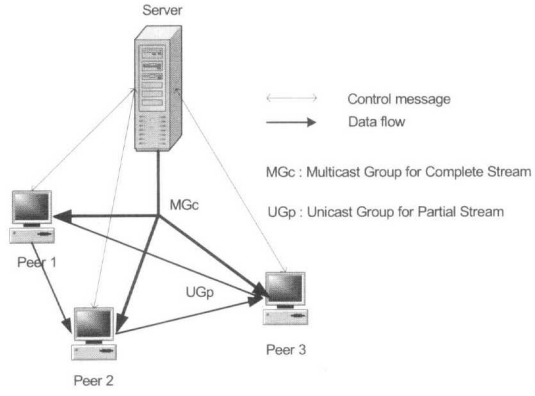


그림 1. 서버와 피어들 간의 제어 메시지통신 및 데이터전송

또한 사용자간에 너무 많은 데이터가 전송되어 로컬 네트워크가 과부하 되는 것을 막기 위하여 특정시간이 경과된 후에 들어오는 요청에 대해서는 새로운 멀티캐스트 채널을 생성하는 threshold based multicast 알고리즘을 한다. 따라서 모든 사용자는 서버로부터 대부분의 데이터를 전송 받고 일부 데이터에 대해서만 다른 사용자를 통하여 받게 된다[12]. 뿐만 아니라 T_i 값을 $L_i/2$이하로 제한함으로써 서비스를 제공하던 사용자가 자신의 서비스가 종료된 이후에 다른 사용자로부터 데이터 전송이 요구될 수 있는 잠재적인 문제도 미연에 방지하였다. 여기서 T_i 비디오 i에 대해 새로운 멀티캐스트 채널을 시작하기 위한 threshold 값이고, L_i는 비디오 i의 길이이다.

제안된 모델은 동일 지역권의 사람들은 비슷한 성향을 갖는다는 특성에 착안하여 클라이언트를 구역별로 각각의 사용자 그룹으로 분류하여 동일 그룹내의 사용자간에는 P2P (peer-to-peer)를 이용하여 데이터를 주고받음으로써 서버로부터 이미 지나간 부분의 데이터를 요구하지 않고도 서비스를 받을 수 있도록 하였다. 사용자간의 peer-to-peer 데이터 전송시 네트워크 자원에 영향을 적게 받기 위해 각 그룹은 ISP 단위로 나누워 진다. 본 논문은 서비스를 요청하는 사용자수가 일시에 폭증할지라도 각 그룹별로 멀티캐스트 채널을 공유하고 부분 스트림에 대해서 P2P 채널을 이용함으로써 서버 네트워크 자원을 늘리지 않고도 모든 사용자에게 양질의 서비스 해줄 수 있다.

해석적 방법을 사용하여 제안된 모델의 구현을 위

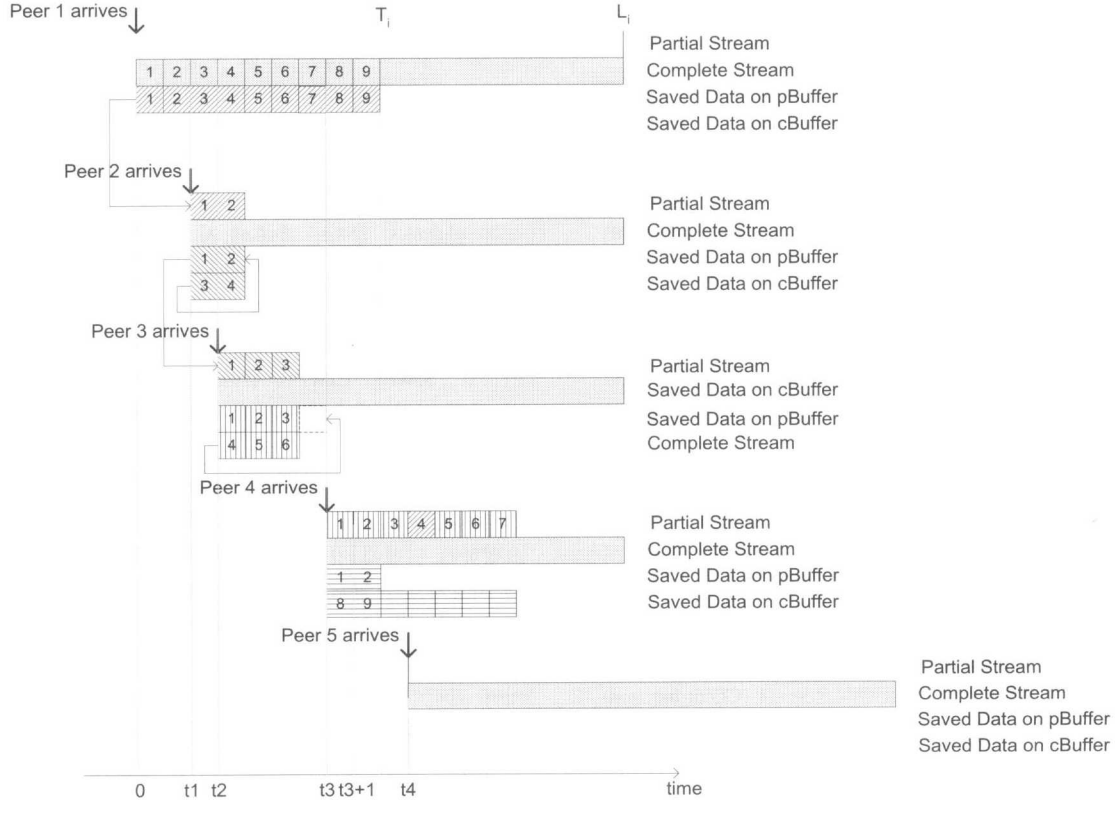


그림 2. 부분 스트림을 위한 Peer-to-Peer 전송모델의 시간진행에 따른 스트림 전송과 저장변화

해 요구되는 사용자의 디스크 자원 이용량과 이를 통해 절감되는 서버 네트워크 자원을 분석하였다. 분석결과 사용자의 디스크 자원 이용량은 다소 증가하지만, 서버 네트워크 자원에 대한 추가적인 요구가 많이 줄어든 것을 알 수 있었다. 이는 일반적으로 사용자 측의 디스크자원은 여유가 있고 가격 또한 상대적으로 저렴하기 때문에 바람직한 접근 방향이다. 다양한 시뮬레이션을 통하여 같은 대역폭을 갖는 네트워크 환경에서 제안된 모델을 기본 CIWP 모델, threshold based multicast 모델과 비교해볼 때 서비스 효율(서비스를 받기 위해 기다리는 비율과 요청직후 서비스를 받는 사용자의 비율) 과 지연시간이 상당히 개선됨을 알 수 있다.

본 논문은 다음과 같이 구성되어있다. 2절에서는 본 논문에서 제시한 모델의 기본동작에 대해 설명하고, 3절에서는 제시한 모델에 적용되는 알고리즘에 대해 자세히 살펴본다. 4절과 5절에서는 각각 해석적 분석과 시뮬레이션을 통해 제시한 모델의 성능을 평가하고, 6절에서 결론을 통해 본 논문을 마무리한다.

Ⅱ. 멀티미디어 부분 스트림을 위한 Peer-to-Peer 전송 모델

앞 절에서 설명한바와 같이, 본 논문에서 제시한 멀티미디어 부분 스트림을 위한 P2P 전송 모델은 멀티캐스트 채널을 통한 데이터전송을 위해서는 threshold based multicast 알고리즘을 적용하고, 부분 스트림 전송을 위해서는 피어 간의 P2P 전송을 이용 한다. 본 논문의 P2P 전송에서 사용하는 "피어"라는 용어는 사용자, 클라이언트와 동일한 의미로 함께 사용한다. 그림1은 비디오 서버와 각 피어 사이의 전반적인 제어용 메시지와 데이터의 흐름을 보여주고 있다.

여기서 MG_c는 세 피어가 공유하는 멀티캐스트 채널을 나타내고 UG_p는 피어1이 피어2에게 부분 스트림을 보내주는 유니캐스트 채널을 나타낸다. 서버는 피어1로부터 요청이 들어왔을 때, 해당 비디오에 대한 정보와 피어1의 디스크정보를 주고받은 후 서비스를 시작한다. 피어1에 대한 서비스가 진행되는 중에 피어2로부터 동일한 비디오에 대한 요청이 들어오면 서버는 제어채널을 통하여 현재진행중인 멀티캐스트채널 정보와 피어1에 관한 정보를 피어2에게 보내주고, 피어2에 관한 정보를 피어1에게 보내준다. MG_c를 이용하여 서버는 각 피어에게 비디오데이터를 멀티캐스트해주고, UG_p를 이용하여 피어1은 피어2에게 부분 스트림을 전송해준다. 이후 피어3이 동일한 비디오에 대해 요청할 경우 서버는 피어3에게는 멀티캐스트 채널정보와 피어2에 관한 정보를 보내주고, 피어2에게는 피어3에 관한 정보를 보내준다. 피어3의 요청이 들어온 시점에 따라 피어1로부터 데이터 전송이 필요한 경우가 있는데, 이 경우 피어2로부터 피어3으로 부분 스트림 전송이 끝난 후 피어1로부터 피어3으로의 부분 스트림 전송을 위하여 피어1에게는 피어3에 관한 정보를 보내주고 피어3에게는 피어1에 관한 정보를 보내준다. 다음에 이 경우에 대하여 자세히 설명한다.

그림 2는 임의의 비디오에 대해 시간진행에 따라 각 피어가 받는 데이터 스트림과 각 피어의 버퍼에 저장되는 데이터를 보여준다. 데이터의 각 부분은 구별하기 쉽도록 여러 개의 세그먼트로 구분하였다. 각 피어가 멀티캐스트채널을 통해 받은 스트림은 cBuffer에 저장하고 유니캐스트 채널을 통해 전송받은 부분 스트림은 pBuffer에 저장한다. 이후 또 다른 피어에게 부분 스트림을 전송해야하는 경우에는 pBuffer를 사용한다. 때문에 cBuffer에 있는 데이터를 전송해야 할 때는 cBuffer의 데이터를 pBuffer로 옮긴 후에 전송한다. 이는 cBuffer의 경우엔 일정시간이 지난이후에는 서버로부터 계속해서 멀티캐스트되는 스트림으로 인해 재생된 데이터는 버리고 새로운 데이터를 저장하면서 시간이 경과됨에 따라 지속적으로 버퍼가 갱신되기 때문이다. 두 버퍼를 디스크에 할당할 때는 cBuffer를 우선적으로 할당한 후 여분의 디스크가 존재하면 pBuffer를 할당한다. 그룹 내의 첫 번째 피어는 마스터피어가 되고 만일 첫 번째 피어가 할당할 수 있는 디스크자원이 T_i보다 작으면 $2*vLength$ 이상의 디스크를 할당할 수 있는 피어 중에서 요청순서가 빠른 순서로 $2*vLength$ 이외의 나머지 디스크의 총 합이 T_i가 될 때까지 마스터피어를 추가한다. $vLength$는 이미 멀티캐스트된 멀티미디어 스트림의 사이즈이다. 이때 잉여자원은 pBuffer에 함께 할당한다. 이때 참가한 모든 피어는 마스터 피어 그룹을 형성한다. 개개의 피어는 서비스를 요청하는 시간과 할당할 수 있는 디스크자원이 일정하지 않기 때문에 선행 피어가 미리 모든 세그먼트를 저장하면 디스크 낭비가 너무 크다. 이때 선행피어가 갖고 있지 않은 데이터 세그먼트를 제공하기위해 마스터피어가 필요하다. 이때 만일 그룹 내에 마스터피어 그룹이 완전히 설정되지 않아 제공하지 못하는 부분 스트림이 있다면 해당 피어는 서버로부터 직접 유니캐스트채널을

통하여 필요한 데이터 세그먼트를 받아야한다. 마스터피어가 아닌 각 피어는 pBuffer 와 cBuffer를 위해 각각 *vLength* 만큼의 공간을 할당하고 이전 피어로부터 들어오는 데이터는 pBuffer에 저장하고 서버의 멀티캐스트채널을 통해 들어오는 데이터는 cBuffer에 각각 저장한다. 그림2는 피어1이 마스터피어일 때 이후의 각피어의 버퍼에 저장되는 데이터와 서로간의 P2P 통신을 통해 전송되는 데이터를 세그먼트 단위로 보여준다. 시간 0일 때 피어1로부터 비디오 i에 대한 요청이 들어오면 서버는 L_i 사이즈의 데이터 스트림을 멀티캐스트한다. 피어1은 자신의 디스크상태를 확인한 후 T_i보다 많은 디스크를 할당할 수 있으므로 그룹 내의 마스터피어가 되고, T_i시간동안 데이터를 pBuffer에 저장한다. t1 시간이 경과된 후 피어 2로부터 동일한 비디오에 대한 요청이 들어오면 피어2는 t1 시간만큼의 cBuffer와 bBuffer를 할당한다. 그리고 나서 피어1로부터 세그먼트1과 세그먼트2 부분(비디오 데이터의 처음부터 t2 시간까지의 부분 스트림)을 전송받으면서 재생하고 동시에 pBuffer에 저장한다. 또한 이때 서버로부터 멀티캐스트되는 스트림은 cBuffer에 저장한다. t2 시간이 됐을 때, 피어3으로부터 요청이 들어오고, 피어2는 피어 3에게 세그먼트 1-3 부분을 전송해준다. 피어 3의 이후 동작은 t2 시간에서의 피어2의 동작과 동일하다. 그러나 t3 시간이 됐을 때 피어3의 동작은 다소 달라진다. 그림3은 t3 시간 이후부터 단위 시간별로 피어3의 버퍼에 저장되는 데이터와 피어4에게 전송하는 데이터의 변화를 보여준다. 그림3에서 버퍼가 가득 채워졌을 때 버려지는 세그먼트를 살펴보면 본 논문에서 제시한 알고리즘이 데이터 저장 전송 과정에서 갖는 데이터별 우선순위와 관리정책 알 수 있다.

1) 아직 재생되지 않은 데이터는 재생될 때까지 보존해야만 한다.
2) 이미 재생된 데이터는 뒷부분부터 삭제한다.
3) 전송된 데이터는 버퍼에서 삭제한다. (단 마스터피어는 T_i 시간이 될 때까지 버퍼에 있는 데이터를 그대로 유지한다.)

t3 시간되면 멀티캐스트채널을 통하여 세그먼트7이 들어온다. 이때 현재 여분의 버퍼가 없으므로 세그먼트7을 저장하기 위해서는 다른 세그먼트를 버려야만 한다. 버퍼내의 데이터를 살펴보면 먼저 세그먼트5와 세그먼트6은 아직 재생되지 않은 부분이므로 규칙 1)에 의해 재생될 때까지는 보존해야한다. 그러므로 현재 피어3에게 이미 재생된 데이터인

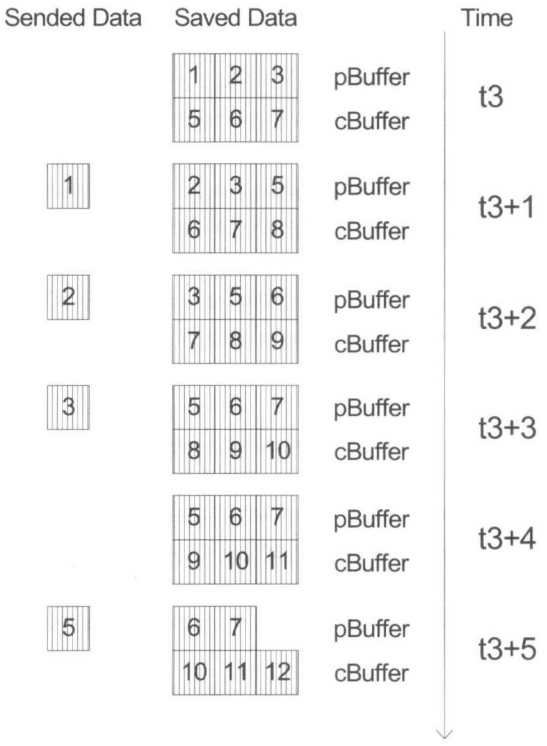


그림 3. t3 이후의 피어3의 송신 세그먼트와 버퍼에 저장된 데이터 변화

세그먼트1, 2 ,3 그리고 세그먼트4 중에서 선택해야하는데, 규칙 2)에 따라 이때 가장 늦은 부분인 세그먼트4를 버퍼에서 제거한다. 이는 모든 피어에게 공통적으로 필요한 데이터 앞부분은 선행피어로부터 릴레이 되도록 함으로써 마스터피어로 하여금 동일한 세그먼트를 중복 전송하지 않도록 한다. t3+1 시간이 되면 세그먼트 1을 피어 4에게 전송해주고, 규칙 3)에 따라 전송된 세그먼트1은 제거하고, 세그먼트5를 pBuffer로 옮기고 cBuffer 에는 세그먼트8을 저장한다. t3+2 시간과 t3+3 시간은 t3+1 일 때와 같은 원리로 동작한다. t3+4가 되면 피어3은 세그먼트4를 가지고 있지 않으므로 피어4에게 전송해주지 못하고 세그먼트11을 받기위하여 이때 재생한 세그먼트8을 삭제한다. 마스터피어는 피어 3을 대신하여 세그먼트4를 피어4에게 전송 해준다. 이후에도 계속하여 피어3은 피어4에게 세그먼트5, 6, 7을 전송해준다.

전체적으로 보면 각각의 피어간의 데이터전송은 선행피어로부터 릴레이하는 형식으로 진행되고, 선행피어가 갖고 있지 않은 데이터는 마스터피어가 제공하게 된다. 실제 피어간의 데이터 전송량을 보

면 피어1은 세그먼트1, 2, 4이고, 피어2는 세그먼트 1, 2, 3이며, 피어3은 세그먼트1, 2, 3, 5, 6, 7로 각각 3개, 3개, 6개의 세그먼트를 전송한 것을 알 수 있다. 만일 모든 피어가 pBuffer에 T_i의 데이터를 저장하고 각각의 피어에게 릴레이 해줬다면 피어1은 세그먼트1, 2만을 전송하고, 피어3은 세그먼트1, 2, 3, 4, 5, 6, 7을 모두 전송해야 하므로 각각 2개와 7개가 된다. 이는 제안된 알고리즘과 비교할 때 디스크자원도 더 많이 사용할 뿐 아니라 일부 피어에게 부하가 편중되는 부작용을 갖는다. 이것이 각 피어의 디스크자원이 T_i+*vLength*보다 작을 확률은 그리 크지 않지만 마스터피어이외의 각 피어가 pBuffer에 *vLength* 만큼만을 할당하는 이유이다.

Ⅲ. 제안된 서버 및 클라이언트 알고리즘

이번 절은 앞 절에서 설명한 데이터 저장 관리 정책과 이를 위한 서버의 역할을 각각의 기능을 수행하는 스레드 별로 나누어서 구체적인 알고리즘을 설명한다. 그림 4는 알고리즘에서 사용된 기호의 의미를 보여준다.

Notations
V_{Id} : identifier for requested video
L_i : size of requested video i
T_i : threshold value
MG_c : multicast group channel for complete stream
UG_p : unicast group channel for partial stream
t_{cur} : current time
t_i : latest starting time of multicasting of video i
vLength : size of lapse time from the starting point (equal to (t_{cur}-t_i))
pLength : size of sending or receiving partial stream
fLenght : size of freeing for pBuffer
mDisk : sum of residue disk size for master peer
freeDisk : a peer's available disk size
pBuffer : disk buffer for partial stream
cBuffer : disk buffer for complete stream
pBsize : size of allocated in pBuffer
cBsize : size of allocated in cBuffer
sPeer : sending peer for partial stream
rPeer : receiving peer for partial stream
seq : sequence number of sending (receiving) video stream
startP : starting pointer for updating pBuffer
segment # : data segment number located in xBuffer

그림 4. 알고리즘에서 사용된 기호

3.1 클라이언트 알고리즘

클라이언트의 셋톱박스는 메인 스레드, 멀티캐스트 수신 스레드, 부분 스트림 송신 스레드, 부분 스트림 수신 스레드, 버퍼 매니지먼트 스레드로 구성된다.

1) 주제어 스레드 알고리즘

주제어 스레드가 처음 서버에 Vid 요청을 보낸 후 서버로부터 *message*(MG_c, *vLength, Li, Ti, mDisk*)를 받으면 우선 사용가능한 디스크 용량(*freeDisk*)을 체크한다. 여기서 *mDisk* 는 현재 그룹 내의 각 마스터피어의 pBuffer 중 잉여디스크할당량의 총합을 나타내는데 *mDisk* 가 T_i보다 작을 때는 추가의 마스터피어가 필요함을 나타낸다. 만일 *mDisk*가 T_i 보다 작을 때 *freeDisk*가 2**vLength*보다 크면 cBuffer에 *vLength* 만큼, 그리고 bBuffer에 *min(freeDisk,-vLength, Ti-mDisk)* 만큼의 디스크 버퍼를 할당한다. 이 경우 마스터피어가 되므로 본인이 마스터피어임을 표시해 둔다. 한편 *freeDisk* 가 2**vLength* 보다는 작지만 *vLength* 보다 크면 cBuffer에 *vLength* 만큼, pBuffer에는 *freeDisk-vLength* 만큼 할당한다. 이때는 본인이 서비스 받는 데는 지장이 없지만 다음피어에게 충분한 데이터를 전송 할 수 없으므로 부족분은 마스터피어가 대신하여 제공해준다. 일단 디스크 버퍼가 할당되면 멀티캐스트 수신 스레드를 실행한다. 그리고 나서 메인 스레드는 계속해서 서버로부터 피어관련 메시지를 기다린다. 메인 스레드는 *info(xPeer, seq, pLength, Type)* 형태의 메시지를 받는다. 여기서 *xPeer*는 *Type*이 *R*일 때는 수신 피어, *Type*이 *S* 일 때는 송신 피어의 정보를 나타낸다. 또한 *seq* 는 부분 스트림 중 보내거나 받아야할 첫 번째 시퀀스 번호를 나타내며, *pLength* 는 *seq*부터 시작해서 보내거나 받아야 될 스트림의 사이즈를 나타낸다. *Type*이 *R* 일 때 메인 스레드는 관련 정보를 이용하여 *rPeer*에게 데이터를 보내기 위해 부분 스트림 송신 스레드를 실행시킨다. *Type*이 *S* 일 때는 부분 스트림 수신 스레드를 실행시켜 *sPeer*로부터 데이터를 수신한다. 메인 스레드는 pBuffer에 새로운 데이터가 들어오거나 데이터의 내용이 바뀌면 서버에게 *modify(startP, pLength, seq)* 메시지를 보냄으로써 이를 알린다. 할당된 자원은 서버로부터 *free(rPeer, fLength)* 메시지를 받거나 서비스가 끝날 때 반환된다. 서버로부터 *free(rPeer, fLength)* 메시지를 받은 피어는 할당된 pBuffer 중 처음 *fLength* 만큼의 디스크자원을 반환하고 서비스가 끝나면 남아있는 모든 pBuffer와 cBuffer를 반환한다.

2) 멀티캐스트 수신 스레드

멀티캐스트 수신 스레드는 메인 스레드에서 받은 메시지 중 MG_c 채널 이름을 이용하여 멀티캐스트 채널로 들어가 스트림을 받는다. 받은 스트림은 일단 메인 스레드에서 할당한 cBuffer에 저장을 하고 cBuffer가 모두 채워지면 재생을 시작한다. 재생이

된 데이터는 디스크 관리정책에 따라 pBuffer로 옮겨지거나 삭제되고, cBuffer는 멀티캐스트되는 스트

```
Main Control Thread
request (V_id);
receive message (MG_c, vLength, L_i, T_i, mDisk);
if (freeDisk > 2*vLength)
  cBsize = alloc (vLength, cBuffer);
  pBsize = alloc (vLength, pBuffer);
  if (T_i > mDisk)
    pBsize = realloc (vLength+(T_i-mDisk), pBuffer);
    master = TRUE;
else if (freeDisk > vLength)
  cBsize = alloc (vLength, cBuffer);
  pBsize = alloc (freeDisk-vLength, pBuffer);
else
  cBsize = alloc (freeDisk, cBuffer);
send alloc_status (cBsize, pBsize);
start multicast receiver thread (MG_c);
while (1)
  if (receive(cmd))
    if (cmd == info (xPeer, seq, pLength, Type))
      if (Type == S)
        start partial stream receiver thread (sPeer, seq, pLength);
      else if (Type == R)
        start partial stream sender thread (rPeer, seq, pLength);
    else if (cmd == bfree (rPeer, fLength))
      free (fLength,pBuffer);
    else if (cmd == progress (T_i))
      send signal (T_i, partial stream receiver thread);
    else if (cmd == progress (L_i))
      exit();
  if (ismodify (pBuffer))
    send modify (startP, seq, pLength);

Multicast Receiver Thread
join (MG_c);
while (1)
  if (MG_c != NULL)
    receive Data (0, MGc);
    if (cBuffer != NULL)
      if (isfull (cBuffer))
        play (cBuffer);
        start buffer management thread (cBuffer);
      else
        if (freeDisk > vLength)
          save (MG_c, cBuffer));
        else
          if ((t_cur-t_i) >= T_i)
            save (MG_c, cBuffer);
    else
      play (MG_c);
  else
    if (!isempty (cBuffer))
      paly (cBuffer);
    else
      exit();
```

그림 5. 클라이언트 알고리즘; 주제어 스레드 , 멀티캐스트 수신 스레드

림에 의해 지속적으로 갱신된다. 이때 *seq* 넘버가 T_i일 때의 스트림이 들어오면 부분 스트림 수신 스레드에게 이를 알린다. 만일 피어가 마스터피어라면 T_i까지의 데이터는 pBuffer에 저장해둔다. 서버로부

```
Partial Stream Receiver Thread
UG_p = connect (sPeer)
while (1)
  receive Data (pLength, UG_p)
  if (UG_p != NULL)
    play (UG_p)
    if (!isfull (pBuffer))
      if (t_cur-t_i < T_i)
        save (UG_p, pBuffer))
      else
        if (partial stream sender thread != NULL)
          save (UG_p, pBuffer)
      end if
    end if
  else
    start buffer management thread (pBuffer)
  end if
 else
    exit
 end if
end while
while (isnotempty (cBuffer))
 play (cBuffer)
end while
exit()

Partial Stream Sender Thread
create channel (UG_p)
send Data (seq, pLength, UG_p)

Buffer Management Thread
while (scan(segment #, xBuffer))
 if (isplayed (segment #))
  if (xBuffer == cBuffer)
   if (!isfull(pBuffer))
    send segment (segment #, pBuffer)
   end if
   delete(segment #)
  else if (xBuffer == pBuffer)
   if (issended(segment #))
    delete (segment #)
   else
    delete max (segment #)
   end if
  end if
 end if
end while
exit()
```

그림 6. 클라이언트 알고리즘; 부분 스트림 수신 스레드, 부분 스트림 송신 스레드, 버퍼 매니지먼트 스레드

터 데이터 전송이 끝난 후 MG_c 채널이 사라지면, 멀티캐스트 수신 스레드도 종료된다.

3) 부분 스트림 수신 스레드

부분 스트림 수신 스레드는 메인 스레드에서 *info(sPeer, seq, pLength, S)* 메시지를 받으면 *sPeer* 로부터 시퀀스 넘버 *seq* 부터 시작하는 *pLength* 길이만큼의 스트림을 받는다. *sPeer*로부터 받은 스트림은 일단 재생을 시작하고 pBuffer에 저장한다. 멀티캐스트 수신 스레드로부터 T_i일 때의 *seq* 넘버를 받았다는 신호가 들어오면 현재 부분 스트림 송신 스레드가 실행되고 있는지를 체크해서 다른 피어와

의 연결이 없으면 이후에 들어오는 스트림은 저장하지 않는다.

```
Notations
MGc' : new multicast group channel after threshold for video i
bufInfo : each peer's pBuffer information that contains  sequence number and length of each segment
socket : peer's information for network connection
mastercount : total number of master peers
t_p : starting time of transmitting each partial stream
peer[n] : information structure about current peer
peer[n-1] : information structure about latest peer located in same group
partList : structure of partial stream scheduling list

Scheduler
receive request (V_id)
if t_cur <= t_i
  send to client message (MG_c, 0, L_i, T_i, mDisk)
else if t_cur-t_i < T_i
  send to client message (MG_c, vLength, L_i, T_i, mDisk)
else
  send to client message(MG_c', 0, L_i, T_i, 0)
  send to dataserver videoList (MG_c, V_id)
receive alloc_stat (cBsize, pBsize)
add peer[n](MG_c, vLength, cBsize, pBsize, bufInfo, mastercount, socket)
if (tcur-ti > 0)
  if ((peer[n-1].pBsize - vLength) >= 0)
    add partList (seq, vLength, t_p, peer[n].socket, peer[n-1].socket)
  else if ((peer[n-1].pBsize - vLength) < 0)
    vLength = vLength-peer[n-1].pBsize
    add partList (seq, peer[n-1].pBsize, t_p, peer[n].socket, peer[n-1].socket)
    t_p=t_p+time(peer[n-1].pBsize)
    for (k=0; k<mastercount;k++)
      if (peer[k].mastercount != 0)
        if ((vLength-(peer[k].pBsize-peer[k].vLength))>0)
          vLength = vLength-(peer[k].pBsize-peer[k].vLength)
          add partList (last_seq (vLength-peer[n-1].pBsize)+1, (peer[k].pBsize-peer[k].vLength), t_p, peer[n].socket, peer[k].socket)
          t_p = t_p+time (peer[k].pBsize-peer[k].vLength)
        else
          add partList (last_seq (vLength-peer[n-1].pBsize)+1, vLength, t_p, peer[n].socket, peer[k].socket)
          vLength = 0
          break
    if (vLength != 0)
      add partList (last_seq (vLength-peer[n-1].pBsize)+1, vLength, t_p, peer[n].socket, dataserver.socket)
while (t_cur < L_i)
  if (t_cur == partList.t_p)
    send to partList.rPeer info(sPeer, partList.seq, partList.pLength, S)
    send to partList.sPeer info(rPeer, partList.seq, partList.pLength, R)

Data Delivery Thread
while (videoList (MG_c, V_id) != NULL)
  send Data (L_i, MG_c)
if (receive (info (...)))
  send Data(pLength, UG_p)
```

그림 7. 서버 알고리즘; 스케줄러, 데이터 전송 스레드

4) 부분 스트림 전송 스레드

부분 스트림 전송 스레드는 메인 스레드에서 *info(sPeer, seq, pLength, R)* 메시지를 받으면 *rPeer*에게 시퀀스 넘버 *seq*부터 시작하는 *pLength* 길이의 스트림을 *rPeer*에게 전송한다.

5) 버퍼 매니지먼트 스레드

버퍼 매니지먼트 스레드는 앞 절에서 언급한 디스크 관리 정책을 따른다. 먼저 세그먼트가 재생됐는지를 체크한 다음에 재생된 세그먼트가 없으면 스레드를 종료한다. 버퍼 매니지먼트 스레드는 cBuffer와 pBuffer에 대하여 다르게 동작한다. cBuffer의 경우엔 pBuffer에 여유가 있으면 세그먼트를 pBuffer로 보낸 후 제거되고 pBuffer가 가득차 있으면 세그먼트는 바로 버려진다. pBuffer에 대해선 재생된 세그먼트 중 전송된 세그먼트를 삭제하고 만일 전송된 세그먼트가 없으면 남아있는 세그먼트 중 세그먼트 번호가 가장 큰 것(세그먼트 중 가장 뒷부분)이 삭제된다.

3.2 서버 알고리즘

서버는 스케줄러와 데이터 전송 스레드로 구성되는데, 스케줄러는 피어로부터 요청을 기다리고, 요청된 비디오에 관련된 멀티캐스트 채널정보와 관련된 그룹 내의 피어정보를 관리하고 각 피어에게 보내주는 역할을 한다.

1) 스케줄러 알고리즘

스케줄러의 주된 기능은 피어로부터 요청이 들어왔을 때 해당 멀티캐스트채널을 알려주는 것과 각 피어간의 부분 스트림 송수신이 필요한 경우 적절한 송신피어를 선택하고 송신피어와 수신피어에게 각각 *info(xPeer, seq, pLength, Type)* 메시지를 보낸다. 만일 그룹 내에 필요한 데이터를 갖고 있는 피어가 없을 경우 서버가 직접 서비스해줘야 하는데, 이때 스케줄러는 데이터 전송 스레드에게 *info(rPeer, seq, pLength, R)* 을 보낸다. 임의의 피어로부터 Vid에 대한요청이 들어오면 해당 멀티캐스트채널이 형성되어있는가를 먼저 체크하고 채널이 존재한다면 현재 진행정도가 T_i를 넘었는지를 체크해서 T_i 이전이면 채널정보와 진행사항에 관련된 정보를 보내준다. 만일 Vid에 대한 멀티캐스트채널이 형성되어있지 않거나 이미 채널이 존재하지만 T_i 시간이 경과한 경우 새로운 멀티캐스트 채널을 만들고 해당피어에게 *message*(MG_c, 0, T_i, L_i, 0) 를 보냄으로써 새로운 그룹을 형성한 후 비디오 리스트에 요청을 추가한다. 그룹 내의 각 피어의 정보는 링크드 리스트로 관리하여 각 피어가 갖고 있는 부분 스트림을 찾기 위해 바로 이전피어를 검색하여 필요한 데이터 길이와 시퀀스 넘버를 찾는다. 그림에서는 이해를 편하게 하기위하여 실질적인 자료구조 대신 배열형태로 나타내었다. 부족한 부분은 마스터피어의 정보를 검색하여 데이터 길이와 시퀀스 넘버를 찾는다. 이때 그룹 내에 마스터피어가 여러 명일 때는 첫 번째 마스터피어부터 순차적으로 검색한다. 이렇게 검색된 정보는 부분 스트림 리스트 *partList(seq, pLength, t_p, rPeer, sPeer)* 로 기록하여 리스트에 따라 시간대별로 각각의 피어에게 메시지를 보낸다.

2) 데이터 전송 스레드 알고리즘

비디오 리스트를 체크해서 리스트에 서비스 해줘야할 항목이 있으면 리스트에 있는 채널 이름에 따라 데이터 전송을 시작한다. 데이터 전송이 끝나면 MG_c 채널에 대한 자원을 반환하고, 비디오 리스트에 아무런 목록이 없을 때까지 계속해서 서비스해준다. 또한 스케줄러로부터 부분 스트림 전송을 위한 메시지가 들어오면 해당 피어에게 스트림을 전송한다.

Ⅳ. 해석적 모델 및 분석

임의의 비디오 i 에 대한 요청만을 생각할 때 그 발생 분포는 평균 분당 요청율이 λi 인 포아송 분포를 갖는다고 가정한다. 또한 서버는 무한의 서버 자원을 가지고 있고, 요청이 들어오면 즉시 서비스된다고 가정한다. 초기분석에서는 임의의 시간동안의 요청에 대해 서버 측 자원과 클라이언트 측 자원의 이용 변화를 분석하여 본 논문에서 제시하는 모델의 성능을 분석한다.

S(t)는 0 부터 시간 t 까지 사용된 전체 서버 대역폭을 나타내고, N(t) 는 0부터 시간 t 까지 서비스 받은 클라이언트 수를 나타낸다. S(t) 가 주어지면 사용된 서버의 평균대역폭 $\overline{B} = \lim_{t\to\infty} S(t)/t$ 이다.

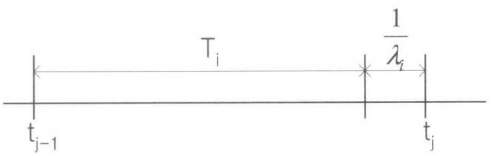


그림 8. j번째 구간 [tj-1, tj) 중에 서비스 받는 전체 클라이언트의 수

그림 8은 j번째 구간 [tj-1, tj) 중에 서비스 받는 전체 클라이언트의 수를 보여준다. 본 논문에서는 관심을 갖는 평균 서버 대역폭은 다음과 같은 식을 사용하여 얻는다.

$$E[t_j - t_{j-1}] = T_i + \frac{1}{\lambda_i} \quad (1)$$

$$\overline{B} = \frac{E[S]}{E[t_j - t_{j-1}]} = \frac{E[S]}{T_i + \frac{1}{\lambda_i}} = \lambda_i \frac{E[S]}{E[N]} \quad (2)$$

여기서 비디오 i 에 대한 threshold 값을 T_i 라 할 때 E[N]=1+$\lambda_i T_i$ 이다. 여기서 E[S]는 서버대역폭의 평균, E[N]은 서비스 받은 클라이언트 수의 평균을 나타낸다. 클라이언트 측 디스크는 충분하다고 가정한다. 임의의 T_i 시간동안의 요청에 대해 필요한 서버자원의 평균값을 구하기 위해 우선 그 기간 동안 k 번의 요청이 이루어질 확률을 다음과 같이 얻어진다.

$$P[K = k] = (\lambda_i T_i)^k e^{-\lambda_i T_i}/k! \quad (3)$$

이 경우 서버가 지연 시간 없이 서비스해 주기 위

하여 필요한 서버와 클라이언트 측 자원에 대한 분석을 다음 두절에 걸쳐서 설명한다.

4.1 서버의 평균 네트워크 대역폭($\overline{B_s}$)

서버 측에 네트워크 자원의 이용량의 평균을 비교함으로써 기존에 연구되어온 내용 중 threshold based multicast 와 비교할 때 본 논문이 제시하는 알고리즘의 효용성 여부를 보여준다. 여기서 b 는 실시간 재생을 위한 데이터 패킷의 전송률을 나타내고, L_i 는 비디오 i 의 데이터 스트림의 사이즈를 나타낸다. 본 논문에서는 적절한 성능을 얻기 위한 표현으로 T_i는 $L_i/2$ 보다 클 수 없도록 제한하였다 [12]. 다음은 클라이언트 당 서비스되는 평균 서버 대역폭에 대하여 설명한다.

1) Threshold_Based Multicast

먼저, 클라이언트 디스크는 충분히 크다고 가정한다. 비디오 i 가 시작된 후 t 시간 후에 클라이언트 요청이 들어온다면, 클라이언트는 부분 스트림이 재생되는 t 시간동안 멀티캐스트되는 스트림을 디스크에 저장하거나 멀티캐스트채널에 남아있는 L_i-t 시간동안의 데이터를 저장해야한다. 그러므로, 클라이언트는 적어도 min{t, L_i-t} 은 버퍼에 저장해야한다.

k 번 요청이 들어왔을 경우 총 서버대역폭의 평균 E[S|K=k] 는 다음 식에 의하여 얻어진다.

$$E[S \mid K=k] = (L_i + \frac{kT_i}{2})b \qquad (4)$$

위 식으로부터 총 서버 대역폭의 평균은 다음과 같다.

$$\begin{aligned} E[S] &= \sum_{K=1}^{\infty} \frac{(\lambda_i T_i)^k e^{-\lambda_i T_i}}{k!} E[S \mid K=k] \\ &= bL_i + \frac{b}{2} \sum_{k=1}^{\infty} \frac{(\lambda_i T_i)^k e^{-\lambda_i T_i}}{k!} kT_i \\ &= bL_i + \frac{\lambda_i b T_i^2}{2} \end{aligned}$$

(5)

그러므로, 평균 서버 대역폭 $\overline{B_s}$ 는 다음과 같이 얻어진다.

$$\overline{B_s} = \lambda_i b \frac{2L_i + \lambda_i T_i^2}{2(\lambda_i T_i + 1)} \qquad (6)$$

또한 $\overline{B_s}$ 는 $T_i = \frac{\sqrt{2L_i\lambda_i + 1} - 1}{\lambda_i}$ 일 때 최소값

$$\overline{B_s} = (\sqrt{2L_i\lambda_i + 1} - 1)b \qquad (7)$$

를 갖는다.

한편, 클라이언트가 할당받을 수 있는 디스크자원이 $L_i/2$보다 적은 경우 다음과 같이 변경한다. 먼저 $D \geq T_i$ 일 때는 위의 분석을 그대로 사용한다. 그러나, 만일 $D < T_i$ 이면 E[S]는 다음과 같이 변경되고

$$E[S] = (L_i + \frac{\lambda_i D^2}{2} + \lambda_i (T_i - D)(L_i - D))b$$

(8)

클라이언트 당 평균 대역폭은 다음과 같이 변경된다.

$$\overline{B_s} = \lambda_i b \frac{L_i + \frac{\lambda_i((L_i - 2D)^2 + T_i^2)}{2}}{\lambda_i T_i + 1}$$

(9)

변경된 각 항목에 대한 설명은 P2P for partial stream에서 자세히 설명한다.

2) P2P for partial stream

$$\begin{aligned} E[S \mid K=k] &= L_i b \\ E[S] &= L_i b \end{aligned} \qquad (10)$$

클라이언트 당 평균 서버 대역폭은 다음 식으로부터 얻어진다.

$$\overline{B_s} = \frac{\lambda_i b L_i}{(\lambda_i T_i + 1)} \qquad (11)$$

threshold based multicast에서와 달리 제시된 방법에서는 부분 스트림은 다른 피어로부터 공급받기 때문에 추가의 채널을 요구하지 않는다. $\overline{B_s}$는 T_i가 L_i에 근접할수록 작아지지만, 제시된 모델에서는 일부 피어가 다른 피어에게 데이터를 전송하기 위하여 본인의 서비스가 끝난 이후 불필요하게 접속을 유지해야하는 일을 막고 또한 각 피어 간의 과도한 데이터 전송으로 인한 과부하를 막기 위하여 T_i는 $L_i/2$이하로 제한한다. 그러므로, $\overline{B_s}$는 T_i가 $L_i/2$일 때 다음과 같은 최소 값을 갖는다.

$$\overline{B_s} = \frac{2\lambda_i L_i b}{(2 + \lambda_i L_i)} \qquad (12)$$

한편, 클라이언트가 할당받을 수 있는 디스크자원이 $L_i/2$보다 적은 경우 위의 분석은 다음과 같이 변경되어야 한다. 만일 $D \geq T_i$ 이면 위의 분석을 그대로 사용한다. 그러나, 만일 $D < T_i$ 이면 E[S]는 다음과 같이 변경된다.

$$E[S] = (L_i + \lambda_i (T_i - D)(L_i - D))b \qquad (13)$$

첫 번째 항목 L_i 는 0과 D 사이에 들어오는 요청에 대해 서비스하기 위한 멀티캐스트 채널을 나타낸다. 두 번째 항목의 $\lambda_i(T_i-D)$는 (T_i-D) 길이의 구간동안 들어오는 클라이언트 요청의 수이고, (L_i-D)는 각

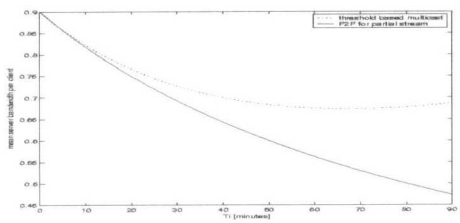


그림 9. Ti에 따른 클라이언트 당 평균 서버 대역폭

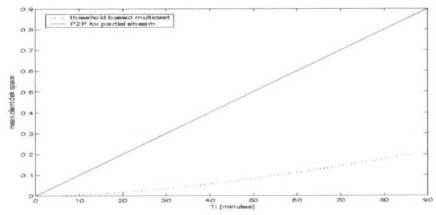


그림 10. Ti에 따른 평균 클라이언트 디스크 공간

요청에 대해 스케줄되는 부분 스트림의 길이이다. 그러므로 이 경우 클라이언트 당 평균 서버 대역폭은 다음과 같이 바뀐다.

$$\overline{B_s} = \lambda_i b \frac{L_i + \lambda_i (T_i - D)(L_i - D)}{(\lambda_i T_i + 1)} \quad (14)$$

분석을 간단히 하기 위해 요청이 들어오는 시간에 따라 첫 번째 마스터 피어를 제외한 다른 마스터 피어의 pBuffer에 추가로 저장되는 데이터가 달라지는데 이 부분에 대해서는 무시한다.

그림 9는 T_i 에 따른 클라이언트 당 평균 서버 대역폭을 보여준다. 그림으로부터 제안된 모델의 클라이언트 당 평균 대역폭이 크게 줄어드는 것을 알 수 있다.

4.2 클라이언트의 평균 디스크 공간($\overline{D_c}$)

threshold based multicast 역시 서버 측 네트워크 자원이용을 줄이기 위하여 클라이언트 측 디스크 공간을 이용한다. 본 논문에서 제시한 알고리즘은 threshold based multicast 에서 사용하는 클라이언트 디스크 자원보다 더 많은 양의 디스크 자원을 요구하는데 클라이언트 측 디스크 자원 이용량의 평균을 비교함으로써 서버 측 네트워크 자원을 줄이기 위해 추가로 소요되는 클라이언트 자원이 얼마만큼 증가하는지를 비교한다.

1) Threshold_Based Multicast

$E[D]$ 는 필요한 전체 클라이언트 디스크 공간을 나타내고, k 개의 클라이언트로부터 요청이 들어오는 경우 필요한 클라이언트의 평균 디스크 공간 E[C|K=k] 는 다음과 같다.

$$E[D \mid K = k] = \frac{kT_i}{2} b \quad (15)$$

전체 클라이언트의 평균 디스크 공간은 다음과 같이 얻어진다.

$$E[D] = \frac{\lambda_i b T_i^2}{2} \quad (16)$$

그러므로, 평균 클라이언트 디스크 공간 $\overline{D_c}$는 다음과 같다.

$$\overline{D_c} = \frac{\lambda_i^2 b T_i^2}{2(\lambda_i T_i + 1)} \quad (17)$$

2) P2P for partial stream

$$E[D \mid K = k] = (k+1) T_i b$$
$$E[D] = T_i b + \lambda_i b T_i^2 \quad (18)$$

그러므로, 평균 클라이언트 디스크 공간 $\overline{D_c}$ 는 다음과 같다.

$$\overline{D_c} = \lambda_i b T_i \quad (19)$$

그림 10은 T_i에 따른 클라이언트 당 평균 클라이언트 디스크 공간을 보여준다. 그림으로부터 제안된 모델은 미리 예상했듯이 기존의 모델보다 더 많은 디스크 공간을 요구하며 선형적으로 증가하는 것을 알 수 있다.

V. 시뮬레이션

앞장의 분석과정에서는 하나의 비디오에 대해 분석하였지만, 일반적으로 멀티미디어 서버는 다양한 선호도를 갖는 다수의 비디오에 대해 서비스한다. 본 장에서는 FCFS batching, CIWP, threshold based multicast 와 P2P for partial stream에 대해 지속적으로 들어오는 요청에 대해 서비스하기 위해 필요한 대역폭을 비교한다. 이를 위해 네트워크 대역폭은 무한대로 가정한다. 사용자로부터 서버로 들어오는 요청의 수는 1/λ의 간격으로 발생하는 포아송 분포를 따르면, 각각의 비디오에 대한 선호도는 zipf-like 분포를 따른다고 가정한다 [13]. 즉, 각 비디오 데이터가 선호도에 따라 정렬됐다고 하면 임의의 i 번째 비디오를 선택할 확률은

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_i} \quad (20)$$

여기서

$$f_i = \frac{1}{i^{1-\theta}},$$

$i = 1, \cdots, N$,

N은 서버에 있는 전체 비디오의 수,

θ는 skew factor

특히 $\theta = 0.271$은 비디오 대여점에서 조사된 선호도에 가장 근접한 분포를 나타내는 일반적으로 알려진 skew factor 값이다.

$$\lambda i = pi\lambda \qquad (21)$$

또한 클라이언트는 충분한 디스크 공간을 가지고 있으며, 일단 서비스가 시작되면 영화가 끝날 때까지는 연결을 끊지 않으며, 개개의 사용자는 고속 통신망을 이용한다고 가정한다. 서비스과정에서 점유되는 대역폭을 나타내는데 있어서 제어채널을 통한 데이터 전송은 상대적으로 매우 작은 대역폭만을 사용하므로 그 값은 무시한다.

표 1은 시뮬레이션 과정에서 사용되는 환경변수를 나타낸다. 서버에서 서비스하는 전체 비디오의 종류는 50가지이고, 분당 요청율 100을 갖는 포아송 분포에 따라 요청이 들어오며, 각 비디오의 길이는 70분에서 110분으로 유니폼 한 분포를 갖는다. 또한 비디오 선택에 있어서 각 비디오에 대한 선호도를 나타내는 skew factor는 0.271을 사용하고, 전체 네트워크는 2003년 5월 현재 국내에 가입되어 있는 ISP 수(81)로 구성되어있다고 가정한다 [한국인터넷 정보센터 제공]. 이때 동일 그룹에서 요청이 들어오는 분포는 zipf 분포($\theta = 0$인 zipf-like 분포와

표 1. 시뮬레이션을 위한 사용된 파라미터

parameter	Average	Range
Number of videos	50	N/A
Request rate (requests/min)	100	80-120
Video length (minutes)	90	70-110
Skew factor (selection movie)	0.271	N/A
Number of groups	81	N/A
Skew factor (request within the same group)	0	N/A

같은 경우)로 나타낸다[13]. threshold based multicast 모델과 제안된 모델에서의 threshold 값 T_i는 $L_i/2$로 가정한다.

5.1 클라이언트 요청에 따른 네트워크 자원 이용량

그림 11-a는 네트워크 대역폭이 무한대라고 가정했을 때 1000명의 사용자로부터 요청이 들어오는 동안 요구되는 네트워크 대역폭을 보여주고 있다.

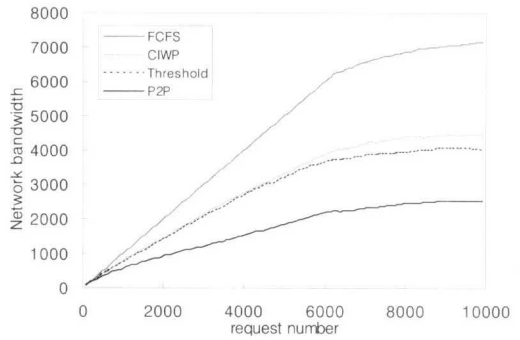


그림 11-a. 클라이언트 요청에 따른 서버 측 네트워크 대역폭

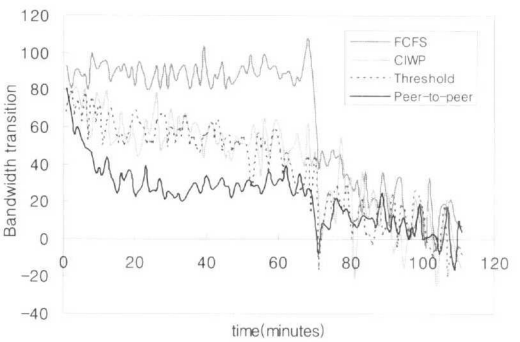


그림 11-b. 시간에 따른 대역폭 변이

Threshold_Based Multicast 와 제시된 모델에서의 T_i 값은 편의상 비디오 길이의 반($L_i/2$) 값을 사용하였다. basic CIWP와 threshold based multicast를 비교해보면 요청이 4000~6000 정도 들어오는 시기를 기점으로 필요한 대역폭이 차이가 생기는데, 분당 평균 100 개의 요청이 들어옴으로 이 부근이 40~60분 즉, $L_i/2$가 되는 시점이다. 그림 11-b는 시간 흐름에 따른 네트워크 대역폭의 변화를 보여준다. 한편의 비디오가 끝나는 시점인 100분에 근접할수록 변화 폭은 0에 가까워짐을 알 수 있다. 전반적으로 시간이 경과할수록 요구되는 대역폭이 줄어들지만 제안된 알고리즘의 경우 40~60분 사이에는 다소 증가된 모습을 볼 수 있다. 이는 T_i가 지나면 이미 서비스 중인 비디오라도 새로운 채널을 할당해야하므로 그 시기가 시작되는 시점에서 나타나는

현상이다. partial stream에 대해서는 서버는 직접관여하지 않기 때문에 threshold based multicast와 비교할 때 새로운 채널을 생성함에 있어서 제시된 모델이 T_i에 대해 보다 민감하게 반응하는 것이다.

5.2 네트워크 대역폭에 따른 평균 지연시간

그림 12는 서버로부터 사용자가 속한 그룹까지의 네트워크 대역폭에 따른 평균적인 지연시간을 나타내고 있다. 제안된 모델에서 서버는 오직 멀티캐스트 스트림만을 서비스하고 ISP 내에서의 클라이언트 요청에 대한 부분 스트림을 위해서는 동일 ISP 내에서의 피어에 의해 제공되기 때문에 기존의 다른 모델과 비교할 때 제안된 모델의 지연시간은 더 짧아진다. 그림 13은 일부 피어가 불시에 네트워크로부터 연결이 끊어졌을 때의 지연시간을 보여준다. 서비스를 받던 피어가 진행 중인 서비스를 중단하거나 다른 이유로 인해 연결이 종료된다면 그 피어로부터 부분 스트림을 전송 받던 다른 피어들은 영향을 받는다. 그러나 시뮬레이션 결과는 이러한 경우 전체 서비스 성능에 미치는 영향은 무시할 수 있을 만큼 경미하다는 것을 보여준다. 그림에서 그래프는 서비스를 받고 있는 클라이언트의 연결 종료에 따른 클라이언트의 평균 지연시간을 보여준다.

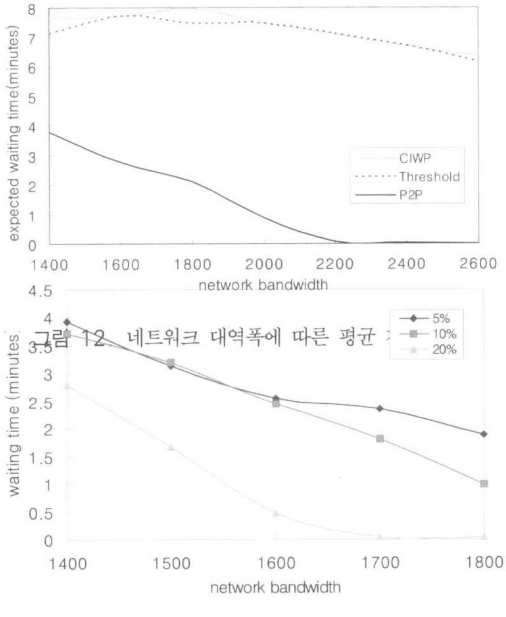


그림 13. 서비스도중 사용자의 일부 사용자의 연결이 끊길 경우 네트워크 대역폭에 따른 평균 지연시간

5.3 네트워크 대역폭에 따른 지연 비율

그림 14는 네트워크 대역폭에 따른 서비스를 요청한 10000명의 사용자 중에서 지연을 겪는 사용자의 비율을 보여준다. 제안된 모델은 다른 모델과 달리 여분의 대역폭이 포화된 상황에서도 동일 그룹 내에 동일한 비디오에 대한 서비스 요청이 있으면 큐를 거치지 않고 바로 서비스 받을 수 있으므로 보다 높은 서비스 율을 보여준다. 그림에서 제안된 방법은 클라이언트의 지연비율을 상당히 낮춰줌을 보여준다.

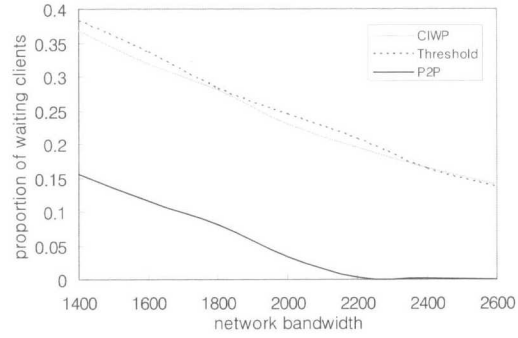


그림 14. 네트워크 대역폭에 따른 지연되는 피어의 비율

VI. 결 론

클라이언트 측의 고 대역 네트워크 기술의 개발로 인해 사용자들은 기존의 텍스트 기반 컨텐츠에서 멀티미디어 기반 컨텐츠를 선호하는 경향을 보이고 있다. 그러나 클라이언트 측 네트워크 환경은 점차 좋아지는 반면 서버 측 네트워크 자원은 이에 맞춰 개선되지 못하고 있다.

본 논문은 클라이언트 측 디스크 자원을 이용함으로써 서버 자원절감 효과를 얻고자 하는 CIWP 모델을 기본으로 하여 부분 스트림에 대해 P2P 모델을 접목시킨 새로운 모델을 제시하고 있다. 그리고 피어간의 과도한 데이터 전송으로 인한 과부하를 막기 위해 threshold based multicast 알고리즘을 도입하였다. 미리 정해둔 threshold 시간 T_i 이후에 들어오는 요청에 대해서는 새로운 멀티캐스트 채널을 생성하는 이 모델을 적용함에 있어서 T_i는 $L_i/2$를 넘지 않도록 제한하였다.

해석적 분석결과에 따르면 제안된 모델은 비록 추가적인 클라이언트 디스크 공간을 요구하지만 서

버 측의 네트워크 자원에 대한 요구는 현격히 줄여준다. threshold based multicast 모델과 비교할 때 제안된 모델은 다음과 같은 시뮬레이션 결과를 보여준다. 먼저 81개의 ISP로 구성된 네트워크 환경에서 분당 80-120의 요청 율을 갖는 10000번의 요청에 대하여 35%의 네트워크 대역폭절감을 보여준다. 또한 서버는 오직 멀티캐스트 스트림에 대해서만 서비스하고 동일 ISP내에서의 클라이언트 요청에 대한 부분 스트림은 ISP내의 피어로부터 제공되기 때문에 지연시간이 훨씬 짧아진다. 제안된 모델에서는 서비스 도중 일부 클라이언트의 접속이 불시에 끊기더라도 실제 성능에는 크게 영향을 끼치지 않을 보였고, 마지막으로 본 논문에서 제안된 방법은 전체 서비스를 받는 사용자들 중에서 지연을 갖고 서비스 받는 사용자의 비율을 현격히 줄여주었다.

참 고 문 헌

[1] P. J. Shenoy, P.Goyal, and H. M. Vin, "Issues in Multimedia Server Design," *ACM Computing Surveys*, Vol. 27, No. 4, Dec. 1995.

[2] R. Rejaie et al., "Multimedia Proxy Caching Mechanism for Quality Adaptive Streaming Applications in the Internet," *Proc. IEEE Infocom* 2000, IEEE Press, Piscataway, N.J., pp. 980-989, 2000.

[3] K.L Wu, P.S. Yu, and J.L Wolf, "Segment-Based Proxy Caching of Multimedia Streams," *Proc. 10th Int'l World Wide Web Conf.*, Elsevier Science, Amsterdam, pp. 36-44, 2001.

[4] S. Sen, J. Rexford, and D. Towsley, "Proxy Prefix Caching for Multimedia Streams," *Proc. IEEE Infocom* 99, IEEE Press, Piscataway, N.J., pp. 1310-1319, 1999.

[5] Zhi-Li Zhang, Yuewei Wang, David H.C. Du, and Dongli Su, "Video Staging: A Proxy-Server-Based Approach to End-to-End Video Delivery over Wide-Area-Networks," *IEEE/ACM Transactions on Networking*, Vol. 8, No. 4, Aug. 2000.

[6] A. Dan, P. Shahabuddin, and D. Sitaram, "Scheduling Policies for an On-Demand Video Server with Batching," in *Proc. ACM Multimedia*, pp. 168-179, Oct. 1994.

[7] L. Gao, J. Kurose, and D. Towsley, "Efficient Schemes for Broad-casting Popular Videos," in *Proc. NOSSDAV*, Cambridge, UK., July 1998.

[8] C. Charu, J. L. Wolf, and P. S. Yu, "On Optimal Batching Policies for Video-On-Demand Storage Server," in *Proc. IEEE ICMS*, June 1996.

[9] K. A. Hua, Y. Cai, and S. Sheu, "Patching: A Multicast technique for true video-on-demand services," *Proc. ACM Multimedia*, Sept. 1998.

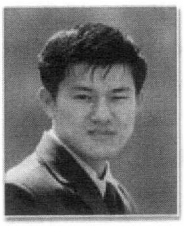
[10] S. W. Carter and D. D. E. Long, "Improving Video-On-Demand Server Efficiency through Stream Tapping," in *Proc. ICCCN*, Las Vegas, NV, pp. 200-207, Sept. 1997.

[11] Lance Olson, *MSDN Magazine*, Feb. 2001.

[12] Lixin Gao and Don Towsley, "Threshold-Based Multicast for Continuous Media Delivery," *IEEE Transactions on Multimedia*, Vol. 3, No. 4, Dec. 2001.

[13] R. Lienhart, M. Holliman, Yen-Kuang Chen, I. Kozintsev, and Minerva Yeung, "Improving Media Services on P2P Networks," *IEEE Internet Computing*, Jan.-Feb. 2002.

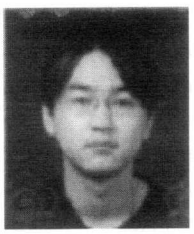
신 광 식(Kwang-Sik Shin) 준회원



2001년 2월 : 인하대학교 전자공학과 졸업
2003년 2월 : 인하대학교 전자공학과 석사
2003년 3월~현재 : 인하대학교 전자공학과 박사과정

<주관심분야> 멀티미디어 데이터 통신, 컴퓨터 네트워크, 시스템 프로그램밍

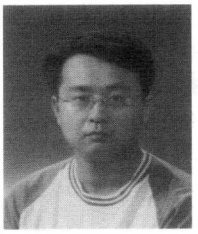
윤 완 오(Wan-Oh Yoon) 준회원



2000년 2월 : 경기대학교 전자공학과 졸업
2002년 2월 : 인하대학교 전자공학과 석사
2002년 3월~현재 : 인하대학교 전자공학과 박사과정

<주관심분야> 분산 처리 시스템, 병렬프로그래밍, 컴퓨터 아키텍처

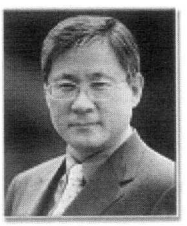
정 진 하(Jin-Ha Cheong) 준회원



1992년 2월 : 인하대학교 전자공학과 졸업
1994년 2월 : 인하대학교 전자공학과 석사
1994년 ~ 1999년 : 한미 기술연구소 (주)
2000년 3월~현재 : 인하대학교 전자공학과 박사과정

<주관심분야> 컴퓨터구조, Fault-tolerant computing, Parallel processing.

최 상 방(Sang-Bang Choi) 정회원



1981년 2월 : 한양 대학교 전자공학과 졸업
1981년~1986년 : LG 정보통신(주)
1988년 3월 : University of Washington 석사
1990년 8월 : University of Washington 박사
1991년~현재 : 인하대학교 전자공학과 교수

<주관심분야> 컴퓨터 구조, 컴퓨터 네트워크, 병렬 및 분산 처리 시스템, Fault-tolerant computing