라이브 미디어 스트리밍 서비스를 위한 하이브리드 P2P 오버레이 구조

(A Hybrid P2P Overlay Architecture for Live Media Streaming)

변 해 선 † 이 미 정 ††
(Haesun Byun) (Meejeong Lee)

요 약 본 논문에서는 라이브 미디어 스트리밍 서비스를 위한 하이브리드 P2P 오버레이 구조를 제안한다. 제안하는 구조는 가까운 지역에서 유사한 대역폭을 가진 피어들로 구성된 메시 오버레이와 메시 오버레이 구조를 기반으로 하여 세션 참여 안정성을 가진 피어들을 구성된 트리 오버레이로 이루어져 있다. 제안하는 방안에서는 메시 오버레이와 트리 오버레이를 유기적으로 활용함으로써 트리 오버레이 구조의 견고성과 메시 오버레이 구조의 통지연을 보완한다. 또한, 트리 오버레이에서 동일 지역 내 업링크 대역폭이 높은 피어를 미디어 소스 피어 가까이에 위치시키며 트리의 깊이를 줄이고 넓이를 넓혀서 스트림 전달의 지연을 최소화한다. 시뮬레이션을 통해, 제안하는 방안과 기존 연구의 성능을 확장성, 서비스 품질 등의 측면에서 평가해 보았다.

키워드 : 라이브 멀티미디어 스트리밍, 하이브리드 P2P 오버레이 네트워크, 트리 최적화 메커니즘

Abstract In this paper, we proposed a hybrid P2P overlay structure for live media streaming. The proposed structure consists of the mesh overlay organized by peers according to the geographical proximity and similar bandwidth range and the tree overlay formed by the peers for which the stability of participation is approved. The proposed scheme enhances the robustness of tree overlay and the long delay of mesh overlay by intelligently combining the utilization of the tree overlay and the mesh overlay. Furthermore, the peers with a large up-link bandwidth are located near to the media source peer. Therefore, it reduces the height of tree, and as a result, the stream transmission delay. Through simulation, we evaluated the performance of the proposed scheme in terms of scalability and quality of services.

Key words : Live media streaming, Hybrid P2P overlay network, Tree optimization mechanism

1. 서 론

1990년대 미디어 스트리밍 기술의 개념이 소개된 이후, 인터넷의 성장 및 다양한 유무선 단말의 개발로 UCC (User Created Contents) 등과 같은 콘텐츠를 인터넷을 통해 라이브로 서비스하는 라이브 미디어 스트리밍 서비스의 이용이 증가하고 있다. 콘텐츠를 전달하기 위한 서비스 구조가운데 전통적인 클라이언트-서버나 IP 멀티캐스트 구조는 확장성이 낮거나 네트워크상에 도입(deployment)이 어렵다는 문제를 가진다. 이에 대한 한 가지 대안으로 사용자 PC들을 연결하여 이들이 가진 콘텐츠 및 자원을 공유하는 P2P 오버레이 기반의 콘텐츠 전달 구조가 주목을 받게 되었다. 이러한 P2P 오버레이 서비스 구조는 자원 활용률이 높고, 확장성이 좋으며, 실용화 가능성이 높다는 장점을 가지나, P2P 통신의 본질적인 특성으로 말미암아 몇 가지 해결해야 할 과제가 있다[1-3]. P2P에서는 피어의 천(churn: 피어가 임의대로 접속/종료하는 현상)에 의해 오버레이 구조 변경이 빈번하게 발생하여 지속적인 스트리밍 서비스를 제공하기 어렵고, 피어의 단말 및 성능(Capability)이 이질

· 이 논문은 2009년도 정부(교육과학기술부)의 재원으로 한국과학재단의 자원을 받아 수행된 연구임(No. 2009-0060034)

† 학생회원 : 이화여자대학교 컴퓨터학과
ladybhs@ewhain.net
†† 정 회 원 : 이화여자대학교 컴퓨터학과 교수
lmj@ewha.ac.kr

논문접수 : 2009년 3월 6일
심사완료 : 2009년 9월 1일

정보과학회논문지 : 정보통신 제36권 제6호(2009.12)

적이어서 각각의 피어가 원하는 서비스 품질을 만족시키기 어렵다는 문제가 있다. 또한 이기적인 피어의 불공평한 자원 기여 문제도 해결해야 할 과제이다. 이와 함께, 라이브 미디어 스트리밍 서비스는 일반적인 요구기반 미디어 스트리밍 서비스보다 더 타이트한 재생 지연을 요구하며, 피어간 동기화 지연이 최소화 되어야 하는 등의 특성을 가지고 있다[4]. 이러한 특성을 고려하고 요구사항을 만족시키기 위해, P2P 오버레이 구조를 어떻게 형성할 것인지에 대한 P2P 오버레이 구조 연구[4-14], 형성된 구조 기반 하에 패킷을 어떻게 효율적으로 전달할 것인지에 대한 패킷 스케줄링 연구[14,15], 스트림 패킷의 에러 및 복구에 관한 연구[16,17] 등 지금까지 많은 방안들이 제안되었다. 본 논문에서는 이들 중 라이브 미디어 스트리밍 서비스를 제공하기 위해 오버레이 구조의 견고성을 향상시키고 피어 대역폭의 이질성을 다룰 수 있는 P2P 오버레이 구조를 형성하는 방안을 연구하고자 한다.

기존에 연구된 P2P 오버레이 구조로는 트리 구조[4-6], 메시 구조[7,8], 트리와 메시를 혼합한 하이브리드 구조가 있다[9-13]. 트리구조는 제어 메시지 오버헤드가 메시 구조에 비해 적으나, 피어의 천에 취약하며, 트리의 리이프(leaf) 피어들과 인터널(internal) 피어들 간 자원 기여가 공평하지 못하다는 단점을 가진다. 메시 구조는 전달 구조 구성이 용이하고 피어의 천에 견고하나, 피어들 간 제어 메시지 오버헤드가 크고 지연이 길다는 문제를 가진다. 하이브리드 구조는 각각 트리와 메시 구조에서 발생하는 단점을 보완하기 위해 다양한 형태로 이들을 혼합한 구조이다. 하이브리드 구조에 대한 대표적인 연구로는 mTreebone[12]이 있는데, 이 방안에서는 트리 구조를 통해 스트림을 전달받고, 플레이에 임박했는데 아직 받지 못한 세그먼트(missing segment)를 패치하기 위해 메시 구조를 사용한다. 트리 오버레이 구조의 보조 개념으로 사용하는 메시 오버레이 구조에서는 CoolStreaming에서와 같이 랜덤으로 메시 파트너 리스트를 선택하여 형성한다. 그리고 메시 오버레이에 있는 피어가 일정 기준 이상 세션에 참가하여 안정성을 확보하면 트리 오버레이에 참여하게 하는 방법으로 트리 오버레이를 확장한다. mTreebone에서 모든 피어들은 하나의 부모 피어, 랜덤 메시 파트너들과 커넥션을 유지한다. 또한 mTreebone에서는 트리 최적화 기법을 수행하는데, 자신과 메시 커넥션을 설립한 피어들을 대상으로 더 작은 대역폭을 가진 피어가 트리상에서 자신보다 더 위에 있다면 그 피어와 스왑을 수행한다. 또한 자신의 부모 피어의 활용 가능한 대역폭보다 소스 피어로부터 가까운 피어가 활용 가능한 대역폭을 가지고 있으면 그 피어를 부모 피어로 변경한다. 이는 트리의 깊이를 줄임으로써 소스로부터의 지연을 줄이는 효과를 가진다.

본 논문에서는 하이브리드 P2P 오버레이 구조를 제안한다. 제안하는 구조는 지리적으로 가까운 지역에 위치하고 유사한 대역폭을 가진 피어들로 구성된 메시 오버레이와 안정성을 확보한 피어들로 구성된 트리 오버레이로 이루어져 있다. 구체적으로, 메시 오버레이에서 세션 참여 안정성이 확인된 피어들을 트리 오버레이의 멤버로 조인시키는데 이때 트리의 최적화된 위치에 피어를 삽입하기 위해 메시 오버레이에서와 같이 피어의 지역성과 업링크 대역폭을 고려한다. 즉, 동일 지역 내에서 안정성을 확보한 피어들 가운데 업링크 대역폭이 높은 피어일수록 소스 가까이에 위치시키고, 이 피어들이 활용 가능한 업링크 대역폭만큼 자식 피어를 서비스하도록 하여 트리의 깊이(depth)를 줄인다. 이는 결과적으로 스트리밍 지연을 줄이는 효과가 있다.

제안하는 하이브리드 P2P 오버레이 구조는 트리 오버레이를 통해 시스템 확장성을 높이고, 메시 오버레이를 통해 스트림 전달 구조의 안정성을 보장하는 등 기존 하이브리드 방안들의 장점을 그대로 유지한다. 또한 세션 참여 안정성을 확보한 피어들만으로 트리 오버레이를 구성하고 트리의 깊이를 최소화하기 위해 트리 최적화 기법을 이용한 트리 재구성을 수행한다는 측면에서 mTreebone과 유사한 개념을 사용한다. 그러나 mTreebone의 트리 최적화 메커니즘에서는 각 노드가 최종적으로 최적화된 위치에 삽입될 때까지 주기적으로 최적화 위치를 찾는 동작을 반복하기 때문에 제어 메시지 오버헤드가 크다. 반면, 제안하는 방안에서는 부트스트랩 서버(bootstrap server)를 두어 P2P 오버레이에 참여하는 피어들의 위치와 정보를 유지하고 부트스트랩 서버에 의해서 트리 오버레이를 최적화하는 중앙 집중적 전달구조 최적화 방법을 취한다. 또한 큰 업링크 대역폭을 가진 피어들이 일부 서브 트리의 브랜치에 집중되어 배치되는 것을 사전에 제한하여 트리의 전체적인 대역폭 균형을 유지하고, 비슷한 서브 트리 깊이를 갖도록 함으로써 피어가 위치한 깊이의 레벨에 따라 일정 범위 내로 재생 동기화를 유지할 수 있도록 한다는 측면에서 mTreebone의 트리 오버레이와는 차별성을 가진다.

본 논문의 구성은 다음과 같다. 서론에 이어, 제 2장에서는 제안하는 하이브리드 P2P 오버레이 구조의 개념과 오버레이 구조의 설립 과정을 자세히 설명한다. 제 3장에서는 제안하는 방안에 대한 성능을 평가하고, 마지막으로 4장에서 결론을 맺는다.

2. 제안하는 하이브리드 P2P 오버레이 구조

이 장에서는 제안하는 하이브리드 P2P 오버레이 구

조의 주요 개념, 하이브리드 오버레이 구조 설립 과정 등을 자세히 설명한다.

2.1 하이브리드 P2P 오버레이 구조의 주요 개념

제안하는 방안에서는 효율적인 하이브리드 P2P 오버레이 구조를 설립하기 위해 피어의 ISP 및 지역성, 대역폭, 세션 참여 안정성을 고려한다. 먼저, 오버레이 구조 설립에 있어서 하부 물리적 네트워크의 불일치 문제를 고려하고, 미디어 스트림의 종단간 지연 제약사항에 만족하기 위해 피어의 ISP 및 지역성을 고려한다. 동일한 ISP에 속하는 피어들 중 지리적으로 가까운 피어들 간 커넥션을 설립하면 오버레이 관리를 위한 제어 오버헤드에 소모되는 대역폭과 지연 관점에서 P2P 시스템 성능을 향상 시킬 수 있다[10,13]. 둘째로, 자원 활용률을 극대화하고 서로 다른 성능을 가진 피어들의 이질성을 다룰 수 있는 하이브리드 P2P 오버레이 구조를 설립하기 위해 피어의 대역폭을 고려한다. 마지막으로, 견고한 하이브리드 P2P 오버레이 구조를 설립하기 위해 피어의 세션 참여 안정성을 고려한다. 여기서 세션 참여 안정성은 피어가 세션을 얼마나 오랫동안 유지하고 있는지를 기준으로 평가한다.

그림 1은 제안하는 하이브리드 P2P 오버레이 구조의 예를 보여주고 있다. 제안하는 오버레이 구조는 트리 오버레이 레벨과 메시 오버레이 레벨로 이루어져 있다. 트리 오버레이는 세션 참여 안정성을 확보한 피어들로 이루어져 있으며, 메시 오버레이는 트리 오버레이에 속하지 않는 피어들로서 동일한 ISP 및 지리적으로 가까운 지역에 있는 피어들로 이루어져 있다. 여기서 지리적으로 가까운 지역이란 도시, 우편번호 등과 같은 피어의 지역적 정보를 이용한다고 가정한다.

모든 피어들은 처음에 메시 오버레이에 조인하여 스트리밍 데이터를 교환하다가 세션 참여 안정성이 확인되면 트리 오버레이의 멤버가 될 수 있는 자격을 가진다. 그리고 자격을 획득한 메시 오버레이의 피어가 트리

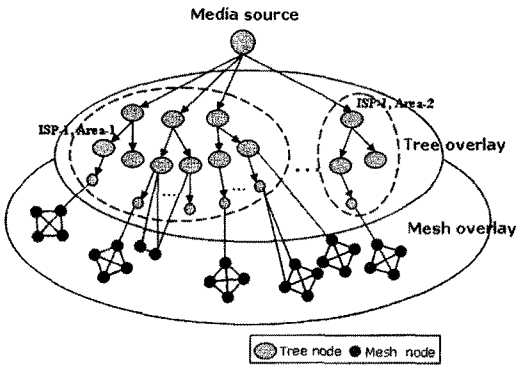

그림 1 하이브리드 P2P 오버레이 구조의 개념도

오버레이의 멤버로 조인할 때 업링크 대역폭이 높은 피어일수록 소스 피어 가까이에 위치시키고, 서브 트리 전반에 걸쳐 큰 업링크 대역폭을 가진 피어들을 골고루 포진시켜 트리의 모든 브랜치가 비슷한 깊이를 갖도록 한다.

제안하는 방안에서는 피어의 상태에 따라 라이브 미디어 스트림 전달 방법이 세 가지로 나눠진다. 먼저, 메시 오버레이에 속한 피어들은 필요한 세그먼트를 상대 피어에게 Pull하여 전달 받는다. 트리 오버레이에 속한 피어들에게는 부모 피어가 자식 피어에게 지속적으로 Push함으로써 스트림이 전달된다. 메시 피어에서 트리 피어로 전환하는 과정에 있는 트리 후보 피어들은 커넥션 변경이 발생하는 동안 세그먼트 중복 및 손실을 최소화하기 위해 트리 피어로 전환이 끝날 때까지 메시 피어들에게는 Pull하여 세그먼트를 전달받고, 새로운 부모 피어로부터 Push로 전달받는다. 그림 2는 제안하는 방안에서 라이브 미디어 스트림 전달 방법을 보인 그림이다. 트리 피어인 A와 B는 자신의 자식 피어들에게 세그먼트를 Push한다. 메시 피어인 D와 E는 서로 Pull하여 세그먼트를 요청하여 전달받는다. 트리 후보 피어인 C는 기존에 메시 커넥션을 설립하고 있던 D로부터 Pull에 의해 스트림을 전달받고, 새로운 부모 피어인 B로부터는 Push에 의해 스트림을 전달받는다.

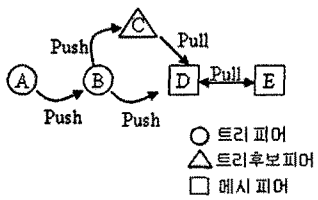

그림 2 미디어 스트림 전달 방법

2.2 메시 오버레이의 설립

라이브 미디어 스트리밍 서비스를 받고자 하는 피어는 하이브리드 P2P 오버레이 네트워크에 조인하기 위해 먼저 부트스트랩 서버에게 자신의 ISP, 지역정보, 업링크/다운링크 대역폭 정보를 포함한 메시 조인 요청 메시지를 보낸다. 부트스트랩 서버는 P2P 시스템에 접속/종료하고자 하는 피어를 관리하는 서버로서, 피어 정보 테이블을 유지, 관리하는 서버이다. 피어 정보 테이블은 동일 채널에 참여하고 있는 피어들의 ID, ISP, 지역 정보, 대역폭 정보, 피어의 상태(트리/메시), 트리 오버레이에서의 피어의 깊이, 부모 피어, 시스템 조인 시간 등의 정보를 저장하고 있는 테이블이다. 제안하는 방안에서 부트스트랩 서버는, 위에서 언급한 역할과 더불어, 새로운 피어를 서비스할 피어들을 결정하고, 최적화된 트리 오버

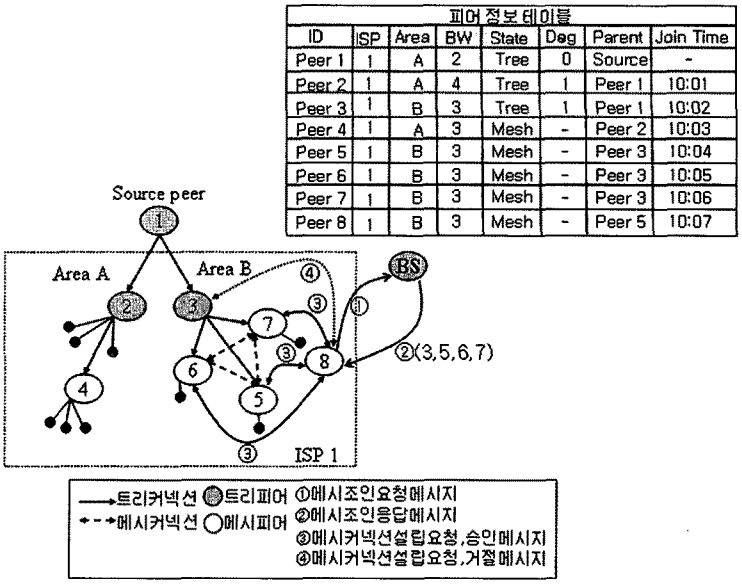

그림 3 메시 오버레이 참여 과정

레이를 구성하기 위해 트리를 관리하는 등 하이브리드 P2P 오버레이를 위한 중앙 서버와 같은 역할을 수행한다. 부트스트랩 서버는 조인 요청 피어로부터 받은 정보를 기반으로 조인 요청 피어와 동일 ISP, 지리적으로 가까운 지역에 위치한 피어들을 피어 정보 테이블에서 검색한 후, 그 정보를 가지고 조인 요청 피어를 서비스 해줄 피어들의 리스트인 서비스 피어 리스트를 만든다. 그리고 서비스 피어 리스트를 포함한 메시 조인 응답 메시지를 조인 요청한 피어에게 보낸다. 조인 요청한 피어는 부트스트랩 서버로부터 메시 조인 응답 메시지를 받으면 서비스 리스트에 포함되어 있는 일부 또는 전부의 피어들과 메시 오버레이 커넥션을 설립한다.

그림 3은 메시 오버레이에 새로운 피어가 참여하는 과정의 예를 보여주고 있다. 하이브리드 P2P 오버레이 네트워크에 조인하기 위해 피어 8이 메시 조인 요청 메시지를 보낸다①. 부트스트랩 서버는 피어 8과 동일 ISP, 지리적으로 가까운 지역에 위치한 피어들을 피어 정보 테이블에서 검색한 후 서비스 피어 리스트(피어 3,5,6,7)를 만들고, 서비스 피어 리스트를 포함한 메시 조인 응답 메시지를 피어 8에게 보낸다②. 피어 8은 부트스트랩 서버로부터 메시 조인 응답 메시지를 받으면 서비스 피어 리스트에 포함되어 있는 일부 또는 전부의 피어들과 메시 오버레이 커넥션을 설립한다③. 이 때 조인 요청 피어를 서비스할 충분한 대역폭을 가지고 있지 않은 피어는 메시 오버레이 커넥션 설립 요청을 거절한다④.

제안하는 방안에서는 조인 요청 피어가 메시 커넥션을 설립할 피어를 찾기 위해 메시 조인 요청 메시지를 플러딩하지 않고 부트스트랩 서버로부터 받은 서비스 피어 리스트 정보를 기반으로 메시 커넥션을 설립할 피어들의 정보를 알 수 있기 때문에 가십 기반 플러딩 기법보다 제어 메시지 오버헤드가 적다. 또한 지역적으로 가까운 지역에 위치한 피어와 메시 커넥션을 설립하기 때문에 랜덤하게 피어를 선택하는 기법보다 제어 메시지 및 세그먼트 교환에 있어서 지연을 줄이고 네트워크 자원 활용률을 향상시킬 수 있다.

2.3 트리 오버레이의 설립

메시 오버레이의 피어는 트리 후보 피어가 되는 기준 시간 동안 세션을 계속 유지하고 있으면 세션 참여 안전성을 확보한 피어로 간주하고 트리 오버레이의 멤버가 될 수 있는 자격을 가진다. 제안하는 방안에서는 이 피어를 '트리 후보 피어'로 정의하였다. 피어는 메시 오버레이에 조인할 때에 부트스트랩 서버로부터 트리 후보 피어가 되는 기준 값(단위 초, 이하 참여 안정성 판단 기준)을 전달받는다. 메시 피어는 세션 참여 시간이 참여 안정성 판단 기준을 초과하면 트리 후보 피어가 되며 부트스트랩 서버에게 트리 조인 요청 메시지를 보낸다. 부트스트랩 서버는 트리 조인 요청 메시지를 받으면 트리 후보 피어가 참여 안정성 판단 기준을 경과했는지를 검사한 후 인증된 피어이면 트리 후보 피어를 트리의 최적의 위치에 삽입하기 위한 메커니즘을 수행한다. 이 메커니즘은 크게 자식 피어 선택 메커니즘, 부모 피어 선택 메커니즘, 트리 최적화 메커니즘으로 이루어진다. 이들 메커니즘은 디그리 별로 수행되는데 즉, 트리의 첫 번째 디그리에서 자식 피어 선택 메커니즘을

수행하여 만족하는 피어를 발견하지 못하였다면 동일 디그리에서 부모 피어 선택 메커니즘을 수행한다. 부모 피어 선택 메커니즘에서도 만족하는 피어를 발견하지 못하였다면 두 번째 디그리에서 자식 피어 선택 메커니즘을 수행한다. 여기서 디그리란 트리 오버레이 네트워크상에서 소스 피어로부터 떨어져있는 홉 수를 의미한다. 이와 같이 자식 피어 메커니즘과 부모 피어 메커니즘은 각 디그리 별로 자식 피어 또는 부모 피어로 만족하는 피어를 발견할 때까지 반복한다. 자식 피어 선택 메커니즘에서 조건에 만족하는 피어를 발견한 경우 트리 최적화 메커니즘을 수행한다. 이들 메커니즘을 수행하는 목적은 트리의 넓이를 넓게 하고 깊이를 줄임으로써 스트림 전달 지연을 최소화하고 피어의 자원 활용을 최대화하기 위함이다.

먼저, 자식 피어 메커니즘은 트리 후보 피어의 자식이 될 피어를 선택하는 것이다. 부트스트랩 서버는 트리의 첫 번째 디그리에서 트리 후보 피어와 동일한 ISP 및 지리적으로 가까운 지역에 위치하고 트리 후보 피어보다 업링크 대역폭이 작은 트리 피어를 자식 피어로 선택한다. 자식 피어 선택 메커니즘을 통해 트리 후보 피어는 트리의 인터널 피어로 조인하게 되고, 큰 업링크 대역폭을 가진 피어일수록 소스 피어 가까이에 위치하도록 트리가 형성된다. 이로 인해 부모 피어 선택 메커니즘에서와 같이 트리의 넓이가 넓어지고 깊이를 줄이는 효과를 갖는다.

자식 피어 선택 메커니즘을 수행한 트리의 디그리에서 자식 피어 선택 메커니즘을 만족하는 피어를 발견하지 못한 경우 즉, 트리 후보 피어보다 더 작은 대역폭을 가진 피어가 없어서 트리 후보 피어가 트리의 인터널 피어로 들어갈 자리가 없다면 부트스트랩 서버는 부모 피어 선택 메커니즘을 수행한다. 부모 피어 선택 메커니즘은 트리 후보 피어를 서비스할 부모 피어를 선택하는 것이다. 부트스트랩 서버는 자식 피어 선택 메커니즘을 수행한 트리의 디그리에서 트리 후보 피어와 동일한 ISP 및 지리적으로 가까운 지역에 위치하는 트리 피어 가운데 트리 후보 피어를 서비스할 활용 가능한 대역폭을 가지고 있고, 트리 후보 피어보다 업링크 대역폭이 크거나 같은 피어를 선택한다. 여기서 활용 가능한 대역폭의 의미는 업링크 대역폭에서 트리 피어를 서비스하고 있는 대역폭을 제외한 대역폭 즉, 메시 피어를 서비스하고 있는 대역폭이나 사용되지 않는 대역폭은 활용 가능한 대역폭으로 가정한다. 부모 피어 선택 메커니즘을 통해 트리 후보 피어는 트리의 리이프 피어로 조인하게 되고, 활용 가능한 대역폭을 가진, 디그리가 가장 낮은 피어들부터 서비스할 피어를 채워 나가도록 트리가 형성되므로 트리의 넓이(width)가 넓어지고 깊이를 줄이는 효과를 갖는다.

한편, 자식 피어 선택 메커니즘을 통해 트리 후보 피어가 트리의 인터널 피어로 조인하는 경우, 트리 후보 피어의 업링크 대역폭이 자식 피어 선택 메커니즘에서 선택된 자식 피어의 업링크 대역폭보다 크기 때문에 선택된 자식 피어가 서비스 하고 있는 피어(또는 그 이하 서브 트리에 있는 피어)를 트리 후보 피어가 서비스 하도록 하는 것이 트리 후보 피어의 업링크 대역폭을 최대한 활용하는 방법이 된다. 이를 위해 부트스트랩 서버는 트리 최적화 메커니즘을 수행한다. 트리 최적화 메커니즘에서는 트리 후보 피어의 업링크 대역폭이 허용하는 피어의 수만큼 선택된 자식 피어가 서비스 하고 있던 피어(또는 그 이하 서브 트리에 있는 피어)를 트리 후보 피어가 직접 서비스 하도록 한다. 트리 최적화 메커니즘을 통해 부모 피어/자식 피어 선택 메커니즘에서와 같이 트리의 넓이가 넓어지고 깊이를 줄이는 효과를 얻을 수 있다.

표 1은 자식 피어 선택 메커니즘의 슈도코드를 보여주고 있다. 자식 피어 선택 메커니즘에서는 트리 후보 피어와 동일한 ISP 및 지리적으로 가까운 지역에 위치하며(line 12), 트리 오버레이 네트워크의 멤버이며(line 13), 트리 후보 피어보다 업링크 대역폭이 작은 피어를 선택한다(line 14). 동일한 디그리에 위치해 있는 둘 이상의 피어가 조건을 만족하는 경우(line 21), 가장 작은 업링크 대역폭을 가지는 피어가 트리 후보 피어의 자식 피어로 선택된다(line 22~25). 그리고 선택된 자식 피어의 부모 피어가 트리 후보 피어의 부모 피어가 된다.

자식 피어 선택 메커니즘에서는 트리 후보 피어가 트리의 인터널 피어로 조인하므로 두 번의 커넥션 변경이 발생한다. 첫 번째, 트리 후보 피어는 새로운 부모 피어(new_parent_peer)와 트리 커넥션을 설립한다. 둘째, 선택된 자식 피어(new_child_peer)는 트리 후보 피어에게 커넥션을 설립하고 기존 부모 피어와 트리 커넥션을 해제한다. 부트스트랩 피어는 새로운 부모 피어 주소를 트리 조인 응답 메시지에 포함하여 트리 후보 피어에게 보낸다. 또한 트리 후보 피어의 주소를 커넥션 변경 요청 메시지에 포함하여 자식 피어에게 보낸다.

자식 피어 선택 메커니즘을 수행한 디그리에서 트리 후보 피어보다 더 작은 대역폭을 가진 피어가 없는 경우, 즉, 트리 후보 피어가 트리의 인터널 피어로 들어갈 자리가 없다면 자식 피어로서 조건을 만족하는 피어를 찾지 못하게 되는데, 이 경우 부트스트랩 서버는 부모 피어 선택 메커니즘을 수행한다. 자식 피어 선택 메커니즘에서는 트리 후보 피어가 인터널 피어로 트리 오버레이에 조인하는 반면, 부모 피어 선택 메커니즘에서는 트리 후보 피어가 트리의 리이프 피어로 트리 오버레이에 조인한다.

표 1 부트스트랩 서버에서의 자식 피어 선택 메커니즘에 대한 슈도코드

```
1:  tree_cndt_peer_xxx      // 트리 후보 피어
2:  lowest_deg              // 가장 낮은 디그리
3:  orgi_parent             // 트리 후보 피어의 기존 부모 피어
4:  fast_join_time          // 트리 후보 피어가 접속한 시간
5:  new_parent_peer         // 새로운 부모 피어
6:  child_selection_process // 자식 피어 선택 수행 플래그
7:  lowest_up_bw            // 가장 작은 업링크 대역폭
8:  parent_selection_module // 부모 피어 선택 모듈 수행 플래그
9:
10: // 자식 피어 선택 메커니즘
11: for (peerlist begin(); peerlist end(); ++) {
12:     if (tree_cndt_peer_area == a_peer_area_in_peerlist &&          // 동일 지역에 위치하며
13:         a_peer_tree_state_in_peerlist == TRUE &&        // 트리 오버레이 네트워크의 멤버이며
14:         tree_cndt_peer_up_bw > a_peer_up_bw_in_peerlist {
15:                                                 // 트리 후보 피어보다 업링크 대역폭이 작은 피어들 중
16:         if (current_deg == a_peer_deg_in_peerlist) {        // current_deg에 속하는가?
17:           if (lowest_up_bw > a_peer_up_bw_in_peerlist)      // 업링크 대역폭이 가장 적은가
18:             lowest_up_bw = a_peer_up_bw_in_peerlist;        // lowest_up_bw 업데이트
19:             new_child_peer = a_peer_addr_in_peerlist;       // 자식 피어 업데이트
20:             new_parent_peer = the_parent_addr_of_the_peer_in_peerlist;
21:                                                             // 부모 피어 업데이트
22:           } end if
23:         } end if
24:         parent_selection_module = FALSE;        // 부모 피어 선택 모듈 플래그
25:     } end if
26: } end for
```

표 2 부트스트랩 서버에서의 부모 피어 선택 메커니즘에 대한 슈도코드

```
1:  tree_cndt_peer_xxx // 트리 후보 피어
2:  lowest_deg         // 가장 낮은 디그리
3:  orgi_parent        // 트리 후보 피어의 기존 부모 피어
4:  fast_join_time     // 트리 후보 피어가 접속한 시간
5:  new_parent_peer    // 새로운 부모 피어
6:
7:  parent_selection_module() { // 부모 피어 선택 메커니즘
8:    for (peerlist begin(); peerlist end(); ++) {
9:      if (tree_cndt_peer_area == a_peer_area_in_peerlist &&          // 동일 지역에 위치하며
10:         a_peer_tree_state_in_peerlist == TRUE &&        // 트리 오버레이 네트워크의 멤버이며
11:         a_peer_avail_bw_in_peerlist == TRUE &&          // 활용 가능한 대역폭을 가지고 있으며
12:         tree_cndt_peer_up_bw =< a_peer_up_bw_in_peerlist) {
13:                                         // 트리 후보 피어보다 업링크 대역폭이 크거나 같은 피어들 중
14:         if (current_deg == a_peer_deg_in_peerlist) {        // current_deg에 속하는가?
15:             if (orgi_parent == a_peer_addr_in_peerlist ) {  // 기존 부모피어와 동일피어인가
16:                 new_parent_peer = a_peer_addr_in_peerlist           // 부모 피어 업데이트
17:                 } else {
18:                 if (fast_join_time > a_peer_join_time_in_peerlist) {    // 조인시간이 빠른가
19:                     fast_join_time = a_peer_join_time_in_peerlist       // fast_join_time 업데이트
20:                   new_parent_peer =  a_peer_addr_in_peerlist            // 부모 피어 업데이트
21:                 } end if
22:             } end if
23:         } end if
24:     } end if
25:   } end for
26: } // module end
```

부모 피어 선택 메커니즘은 트리 후보 피어의 부모가 될 피어를 선택하는 것이다. 표 2는 부트스트랩 서버에서의 부모 피어 선택 메커니즘에 대한 슈도코드이다. 부트스트랩 서버는 트리 후보 피어와 동일한 ISP 및 지리적으로 가까운 지역에 위치하는 트리 오버레이의 멤버로서 트리 후보 피어를 서비스할 활용 가능한 대역폭을 가지고 있고, 트리 후보 피어보다 업링크 대역폭이 크거나 같은 피어를 선택한다(line 9~12). 이하 선택된 피어를 부모 피어로 부른다. 동일한 디그리에 위치해 있는 둘 이상의 피어가 조건을 만족하는 경우, 커넥션의 변경을 최소화하기 위해서 우선적으로 트리 후보 피어를 서비스하고 있던 피어가 조건을 만족하는지를 확인한다. 만약, 포함되어 있다면 그 피어를 부모 피어로 선택한다(line 18~19). 그렇지 않은 경우 하이브리드 시스템에 접속한 시간에 따라 먼저 접속한 피어를 부모 피어로 선택한다(line 21~24). 부트스트랩 서버는 부모 피어의 주소를 트리 조인 응답 메시지에 포함한 후 트리 후보 피어에게 보낸다. 트리 후보 피어는 트리 조인 응답 메시지를 받으면 부모 피어에게 트리 커넥션을 설립(또는 유지)한다.

그림 4는 자식 피어 선택 메커니즘을 통해 트리 후보 피어 4가 트리의 인터널 피어로 조인하는 예를 보인 것이다. 피어의 번호는 시스템에 조인한 순서라고 가정한다. 자식 피어 선택 메커니즘에 의해 트리 오버레이의 멤버이면서 트리 후보 피어보다 업링크 대역폭이 작은 피어인 피어 3이 자식 피어로 선택된다. 따라서 피어 3의 기존 부모 피어인 피어 1이 피어 4의 부모 피어가 되고, 피어 3이 피어 4의 자식 피어가 된다.

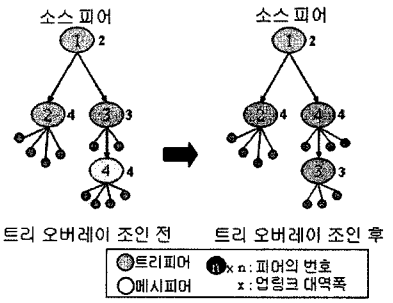

그림 4 자식 피어 선택 메커니즘 수행

그림 5는 부모 피어 선택 메커니즘을 통해 트리 후보 피어가 트리 오버레이에 조인한 예를 보인 것이다. 그림 4에서 피어 5,6,7,8이 트리 후보 피어가 되었을 때 부모 피어 선택 메커니즘에 의해 기존에 서비스를 받고 있던 트리 피어인 피어 4가 피어 5,6,8의 부모 피어로 선택되고, 피어 2가 피어 7의 부모 피어로 선택되어 기존 피어와 커넥션을 유지하고 트리 오버레이의 멤버가 된다.

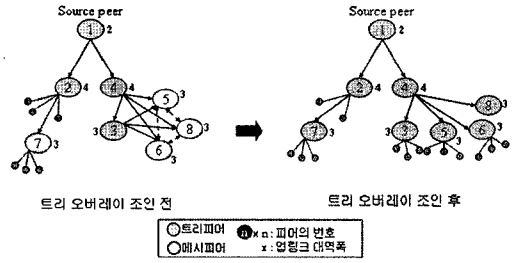

그림 5 부모 피어 선택 메커니즘 - 1

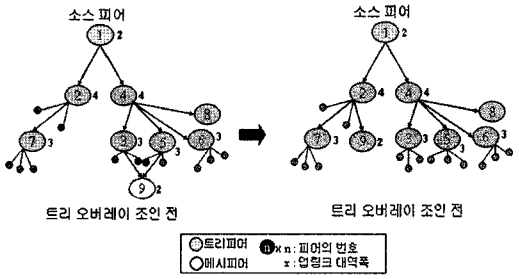

그림 6 부모 피어 선택 메커니즘 - 2

그림 6은 부모 피어 선택 메커니즘을 통해 트리 후보 피어가 트리 오버레이에 조인하는 또 다른 예를 보인 것이다. 그림 6에서 피어 9는 피어 3과 5로부터 서비스를 받았지만 부모 피어 선택 메커니즘에 의해 활용 대역폭을 가지면서 디그리가 가장 낮은 피어 2가 새로운 부모 피어로 선택된다. 따라서 피어 9는 피어 2의 자식 피어가 된다.

2.4 트리 최적화 메커니즘

앞에서 설명한 부모 피어선택 메커니즘에서는 트리 후보 피어가 트리의 리이프 피어로 조인하기 때문에 트리 오버레이가 최적화된 상태로 구성되는 반면, 자식 피어 선택 메커니즘에 의해 구성된 트리 오버레이는 데이터 전달의 지연을 최소화하기 위해 최적화 되어 있지 않다. 그림 7은 트리 후보 피어가 자식 피어 선택 메커니즘에 의해 트리 오버레이에 조인 한 후 발생되는 문제점을 보인 그림 (a)와 이 문제를 해결하기 위해 트리 최적화를 수행 한 후 형성된 트리 오버레이 구조의 예인 그림 (b)를 보인 것이다. 그림 7(a)에서 트리 후보 피어 7은 자식 피어 선택 메커니즘에 의해 피어 1의 자식 피어가 되고, 피어 4의 부모 피어가 된다. 트리 피어로 조인한 7은 4개의 피어를 서비스 할 수 있는 활용 가능한 대역폭을 가지고 있어서 피어 5와 6을 직접 서비스 할 수 있지만 그림(a)에서 보는 바와 같이 트리 구조는 최적화 되지 않는다. 부트스트랩 서버는 트리를 최적화하기 위해 트리 후보 피어에 의해서 디그리가 변경된 피어를 대상으로 트리 최적화 메커니즘을 수행한

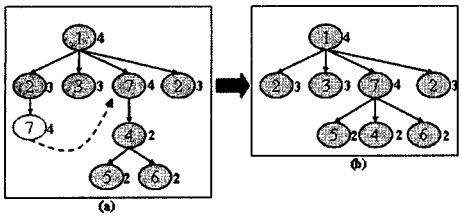

그림 7 지연을 최소화하기 위한 트리 최적화

다. 트리 최적화 메커니즘을 통해 피어 7은 피어 5와 6을 새로운 자식 피어로 커넥션을 설립한다.

표 3은 부트스트랩 서버에서의 트리 최적화 메커니즘에 대한 슈도코드를 보인 것이다. 부트스트랩 서버는 피어 정보 테이블에서 자식 피어 선택 메커니즘에 의해 선택된 자식 피어를 부모 피어로 두고 있는 피어들을 검색한다(line 12~13). 검색된 피어가 있으면(예:손자 피어) 그 피어들이 트리 후보 피어의 자식 피어가 되도록 연결을 설립하도록 한다. 또한 검색된 피어의 자식 피어(예:증손자 피어)가 있는지 다시 검색하고, 검색된 피어가 있으면 그 피어 역시 직접 트리 후보 피어의 자식 피어로 연결되도록 커넥션을 변경한다. 부트스트랩 서버는 트리 후보 피어의 활용 가능한 대역폭이 남아 있는 동안 이와 같은 과정을 반복하면서 트리의 깊이를 줄임으로써 전달 트리를 최적화 한다.

트리 최적화 메커니즘을 통해 활용 가능한 대역폭을 가진 피어가 서비스 할 수 있는 자식 피어의 수만큼 서비스하도록 하여 트리의 깊이를 줄이고 피어의 자원을 최대한으로 사용 할 수 있게 하는 효과를 얻을 수 있다.

2.5 버퍼 분할 및 Pull/Push 협동 스케줄링 통한 데이터 전달

2.3과 2.4에서 설명한 메커니즘에 의해 피어의 커넥션이 변경되는 경우, 피어 커넥션의 변경은 세그먼트의 손실 또는 중복을 발생시킨다. 이를 최소화시키기 위해 새로운 피어와 커넥션 설립이 완료되기 전까지 기존의 피어들과 메시 커넥션을 유지해야만 세그먼트 손실을 최소화 할 수 있으며, 새로운 부모 피어로부터 자신이 받아야할 세그먼트의 번호를 알려줘야만 새로운 부모 피어로부터 중복된 패킷을 수신하는 것을 예방할 수 있다.

그림 8은 제안하는 방안에서의 트리 후보 피어의 커넥션 변경 및 해제를 위한 순서도를 보여주며, 그림 9는 트리 후보 피어에서의 세그먼트 수신을 위한 Pull과 Push 포인터를 나타내고 있다. 그림 8에서와 같이, 제안하는 방안에서는 새로운 부모 피어와 커넥션을 설립

표 3 부트스트랩 서버에서의 트리 최적화 메커니즘에 대한 슈도코드

```
1:  tree_cndt_peer_xxx     // 트리 후보 피어
2:  degree_peer[1] = child_peer
3:                              // 자식 피어 선택 메커니즘에 의해 선택된 피어를 현재 디그리 피어로 설정
4:  num_of_child_peer= 1   // 현재 디그리에서 피어의 수 초기 값 설정
5:  // 트리 최적화 메커니즘
6:  while (tree_cndt_peer_avail_bw == TRUE) {  // 트리 후보 피어의 활용 가능한 대역폭이 있는 동안
7:    current_end = num_of_child_peer    // 현재 디그리에서 피어의 수를 current_end 값에 저장
8:    num_of_child_peer= 0               // 현재 디그리에서 피어의 수 reset
9:    for (int cur_num=1; cur_num<=current_end; cur_num++) { // current_end 만큼 반복
10:     cur_deg_peer[0] = degree_peer[cur_num];  // 현재 디그리 피어를 cur_deg_peer[0]에 저장
11:     for (peerlist begin(); peerlist end(); ++) {
12:       if (cur_deg_peer[0]==parent_peer_in_peerlist && a_peer_tree_state_in peerlist==TRUE) {
13:                                   // cur_deg_peer[0]를 부모 피어로 두고 있는 트리 피어를 검색
14:           if ( avail_bw_tree_cndt_peer > 0) { // 트리 후보 피어의 활용대역폭이 있으면
15:             connect the tree_cndt_peer with child[j] // 검색된 피어와 트리 후보 피어 커넥션 설립
16:             decrease avail_bw_tree_cndt_peer  // 트리 후보 피어 활용 가능한 대역폭 감소
17:             increase cur_deg_peer[0]  // 선택된 피어 활용 가능한 대역폭 증가
18:             num_of_child_peer++;      // 현재 디그리 피어의 child 피어의 수 증가
19:             degree_peer[num_of_child_peer] = a_peer_addr_in_peerlist;
20:                         // 현재 디그리 피어의 배열에 저장 설정
21:           } else {
22:             tree_cndt_peer_avail_bw == FALSE
23:           } end if
24:       }  end if
25:     }  end for
26:   } end for
27: } end while
```

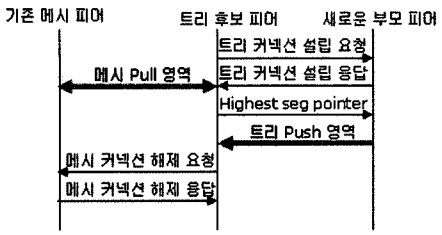

그림 8 트리 후보 피어의 커넥션 변경 및 해제

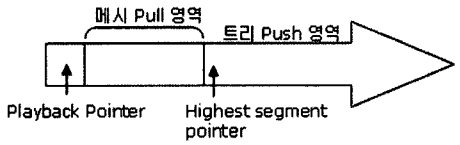

그림 9 세그먼트 수신을 위한 Pull과 Push 포인터

한 후에 Highest segment pointer를 포함한 메시지를 보내서 그 Pointer부터 세그먼트를 보내달라는 요청을 한다. 만약 Highest segment pointer 이전에 수신해야 할 세그먼트가 있다면(메시 Pull 영역), 기존의 커넥션을 통해 Pull(또는 push)하여 수신한다. Highest segment pointer 이전에 수신해야 할 모든 세그먼트를 수신한 후에 또는 Playback pointer가 Highest segment pointer를 넘어설 때에 기존 피어와 메시(또는 트리) 커넥션을 해제한다. 새로운 부모 피어는 Highest segment pointer부터 트리 후보 피어에게 세그먼트를 송신한다(트리 Push 영역).

3. 성능평가

제안하는 하이브리드 P2P 오버레이의 성능을 평가하기 위해 NS-2.26 툴을 이용하여 시뮬레이션을 수행하였다. 그림 10은 시뮬레이션을 위한 네트워크 모델을 보여주고 있다. 시뮬레이션 네트워크 모델은 GT-ITM 토폴로지 생성기를 이용하여 Transit-Stub 그래프 모델로 구성하였다. 구성된 토폴로지는 100개의 라우터 노드들이 백본을 형성하고 있으며, 각 라우터에 0~12개의 피어들을 랜덤하게 연결하였다. Transit-Stub 네트워크의 각 링크 대역폭은 100~155Mb로 구성하였다. 피어가 연결되는 링크로는 2.4Mb, 10M, 45Mb, 100Mb 대역폭을 랜덤하게 설정하였으며, 링크의 지연은 피어와 라우터, 라우터와 백본간에는 1~2ms으로 랜덤하게 선택되며, 백본 네트워크에서는 100~450ms의 범위로 설정하였다.

시뮬레이션에서는 트리 및 메시 커넥션을 유지하기 위한 메시지 전송 인터벌은 30초, 시뮬레이션 런타임은 1500초, 소스 피어에서의 멀티미디어 스트리밍 전송은 첫 번째 피어가 조인하는 시점부터 시뮬레이션 타임 1000초까지 스트리밍 서비스를 제공하며, 피어는 500초 이전에 uniform 하게 조인하며, 한번 접속하면 시뮬레이션이 끝날 때까지 참여 상태로 있다고 가정하였다. 또한 소스 피어는 최대 4개의 피어를 서비스 할 수 있는 것으로 가정하였고, 각 피어는 최대 1~4개 피어와 랜덤하게 커넥션을 설립할 수 있다고 가정하였다.

시뮬레이션에서는 제안하는 방안의 확장성, 서비스 품질 측면에서 성능을 평가하기 위해 피어의 수를 변화시

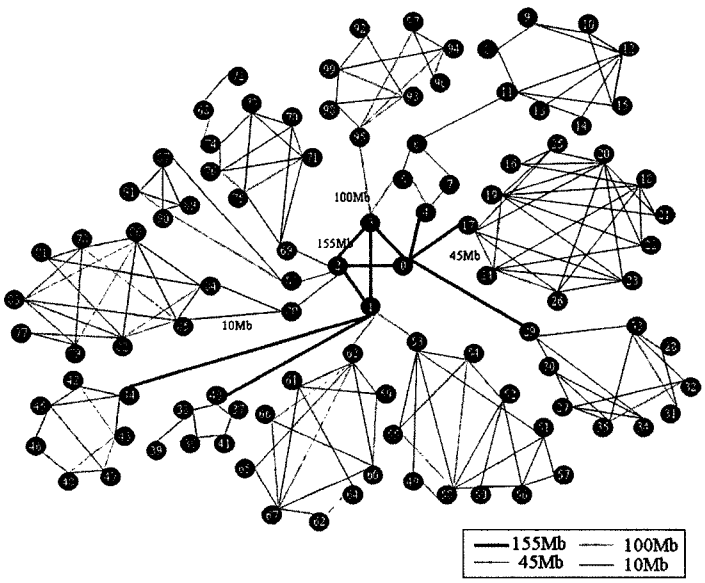

그림 10 네트워크 토폴로지

켜보면서 제어 메시지 오버헤드, 데이터 손실 율, 평균 처리율 등을 측정하였다.

***제어메시지 오버헤드**

제어 메시지 오버헤드는 피어가 받은 데이터양에 대한 메시지 양의 비율을 의미하며, 메시와 트리 오버레이에 접속하기 위한 메시 조인 요청/응답 및 트리 조인 요청/응답 메시지, 커넥션 변경/응답 메시지, 메시와 트리 오버레이를 유지하기 위한 커넥션 유지 메시지, 트리 최적화 관련 메시지 등을 모두 포함한다. 이 가운데, mTreebone과 제안하는 방안의 주된 차이를 보일 수 있는 제어 메시지의 요인은 트리 최적화를 위해 전달되는 메시지이다.

그림 11에서 보는 바와 같이, 제안하는 방안이 mTreebone 보다 제어 메시지 오버헤드가 적다.

mTreebone에서는 Treebone의 평균 depth를 줄이기 위하여 트리 최적화 메커니즘에서의 두 알고리즘인 High-Degree-Preemption, Low-Delay-Jump를 수행한다. 이들 알고리즘에서는 피어들 간 사용가능한 대역폭, 소스와의 거리(distance) 등의 정보를 교환한다. 또한 Treebone에 조인하는 피어가 최적의 자리로 삽입될 때까지 부모 피어와 자리 변경을 수행하면서 트리를 따라 올라간다. 반면, 제안하는 방안에서는 트리 오버레이에 조인할 때 여러 번의 자리 변경을 요구하지 않고 중앙 서버에 의해 최적의 자리로 삽입되기 때문에 mTreebone보다 자리 변경이 발생하는 횟수가 적어서 제어 메시지 오버헤드가 적다.

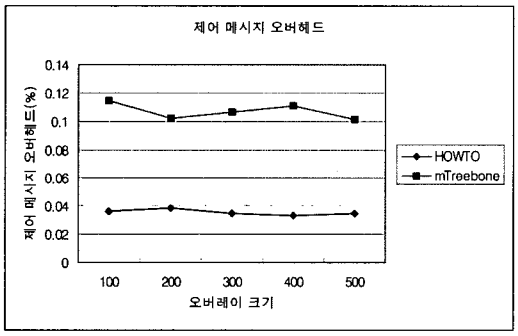

그림 11 제어 메시지 오버헤드

*** 평균 데이터 처리율**

그림 12는 평균 데이터 처리율을 보인 그래프이다. 그림에서 보는 바와 같이, 각 피어에서 수신한 mTreebone의 평균 처리율이 300Kbytes에 미치지 못하는 것을 볼 수 있다. 즉, 플레이 데드라인에 들어오지 못한 세그먼트의 수가 mTreebone에서 더 많다는 것을 의미하는데, mTreebone에서는 제안방하는 방안보다 부모피어 변경이 더 잦아서, 부모 피어를 변경하는 동안에 세그먼트 손실이나 지연이 발생할 수 있다.

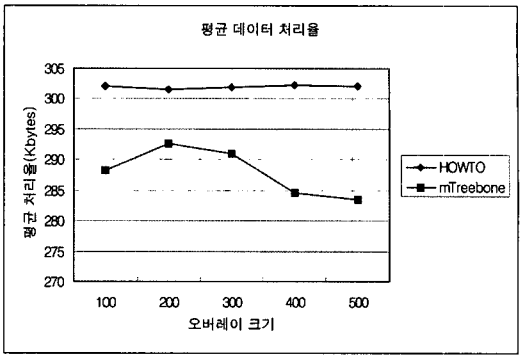

그림 12 평균 데이터 처리율

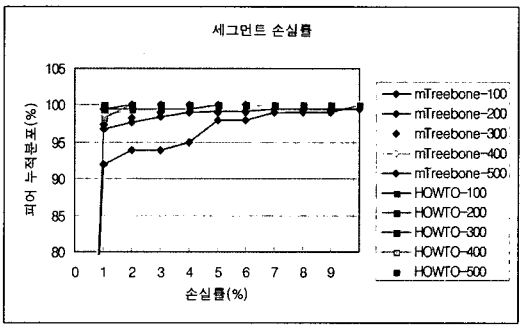

그림 13 세그먼트 손실율

그림 13은 세그먼트 손실율에 대한 누적 분포 함수를 보인 그래프이다. 그림에서 보는 바와 같이, HOWTO는 99%의 이상의 피어들이 손실이 거의 없고 일부 피어들이 2% 미만의 세그먼트 손실을 경험했다. 그러나 mTreebone에서는 97% 이상의 피어들은 손실이 거의 발생하지 않았으나, 일부 피어들이 3% 이상의 세그먼트 손실을 경험하였다.

4. 결 론

본 논문에서는 라이브 미디어 스트리밍 서비스를 위한 하이브리드 P2P 오버레이 구조를 제안하였다. 제안하는 구조에서는 지역성, 세션 참여 안정성, 피어의 특성을 고려하여 트리와 메시 오버레이를 구성하였다. 특히, 트리 오버레이에서의 전송 지연과 재생 지연, 시작 지연을 최소화하기 위해 피어가 트리 오버레이에 조인하는 과정에서 피어 선택 메커니즘 및 트리 최적화 방안을 통하여 효율적인 트리 오버레이가 구성되도록 하였다. 또한 트리 오버레이에 조인하는 동안 패킷의 손실 및 중복을 최소화하기 위하여 버퍼 분할 및 Pull/Push 스케줄링을 통하여 세그먼트 전달의 지속성을 보장하도

록 하였다. 시뮬레이션을 통하여 제안하는 방안의 제어 메시지 오버헤드, 평균 처리율, 세그먼트 손실률 측면에서 성능을 측정하였다.

추후, 피어의 비정상적인 탈퇴 시 하이브리드 오버레이 구조를 재구성하는 방안과 실시간 스트리밍의 중요한 요건 중 하나인 피어간의 스트리밍 동기화를 극대화할 수 있는 피어 선택 메커니즘을 연구할 것이다. 또한 스트리밍 세션의 길이가 정해져 있지 않는 라이브 미디어 스트리밍 서비스의 경우 부트스트랩 서버의 진단에 의해 가장 적절한 피어의 참여 안정성 판단 기준을 정의하는 알고리즘에 대해 연구할 것이다.

참 고 문 헌

[1] F. Thouin, and M. Coates, "Video-on-Demand Networks: Design Approaches and Future Challenges," *IEEE Networks*, pp.42-48, March/April 2007.

[2] W.-P. Ken Yiu, X. Jin, and S.-H. Gary Chan, "Challenges and Approaches in Large-Scale P2P Media Streaming," *IEEE Multimedia*, pp.50-59, April-June 2007.

[3] D.-E. Meddour, M. Mushtag, and T. Ahmed, "Open Issues in P2P Multimedia Streaming," Proceedings of MultiComm'06, pp.43-48, June, 2006.

[4] B. Li and H. Yin, "Peer-to-Peer Live Video Streaming on the Internet: Issues, Existing Approaches, and Challenges," *IEEE Communications Magazine*, June, 2007.

[5] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable Application Layer Multicast," *Proceeding of ACM SIGCOMM'02*, pp.205-220, September, 2002.

[6] D.A.Tran, K. A. Hua, and T. Do, "ZIGZAG: An Efficient Peer-to-Peer Scheme for Media Streaming," *Proceeding of IEEE INFOCOM'03*, April, 2003.

[7] X. Zhang, J. Liu, B. Li, and T.-S. Peter Yum, "CoolStreaming/DONet: A Data-Driven Overlay Network for Efficient Live Media Streaming," *Proceeding of IEEE INFOCOM'05*, March, 2005.

[8] S. Xie, B. Li, G. Y. Keung, and X. Zhang, "CoolStreaming: Design, Theory, and Practice," *IEEE Transactions on Multimedia*, Vol.9, No.8, December 2007.

[9] M. Zhou, and J. Liu, "A Hybrid Network for Video-on-Demand," *Proceeding of IEEE ICC'05*, pp.1309-1313, May, 2005.

[10] Q. Huang, H. Jin, and X. Liao, "P2P Live Streaming with Tree-Mesh based Hybrid Overlay," *Proceeding of IEEE ICPPW'07*, September, 2007.

[11] Hai Jin, Xuping Tu, Chao Zhang, Ke Liu, and Xiaofei Liao, "TCMM: Hybrid Overlay Strategy for P2P Live Streaming Services," *GPC 2007*, LNCS 4459, pp.52-63, 2007.

[12] F. Wang, Y. Xiong, and J. Liu, "mTreebone : A Hybrid Tree/Mesh Overlay for application-layer live video multicast," *Proceeding of IEEE ICDCS'07*, May, 2007.

[13] C. Xie, G. Chen, A. Vandenberg, and Y. Pan, "Analysis of hybrid P2P overlay network topology," *Elsevier Computer Communications 31*, pp.190-200, August, 2007.

[14] N.Magharei and R. Rejaie, prime, "PRIME: Peer-to-Peer Receiver-drIven Mesh-based Streaming," Infocom 2007.

[15] H. Chi, Q. Zhang, J. Jia and X.Shen, "Efficient Search and Scheduling in P2P-based Media-on-Demand Streaming Service," *IEEE Journal on Selected areas in communications*, vol.25, no.1, January, 2007.

[16] E. Setton, P. Baccichet, and B. Girod, "Peer-to-Peer live Multicast: A Video Perspective," *Proceedings of the IEEE*, vol.96, no.1, January 2008.

[17] V. Foder and G. Dan, "Resilience in Live Peer-to-Peer Streaming," *IEEE Communications*, June 2007.

변 해 선

2003년 이화여자대학교 과학기술대학원 컴퓨터학과 졸업(공학석사). 2003년~현재 이화여자대학교 과학기술대학원 컴퓨터공학과 박사과정. 관심분야는 시그널링 프로토콜, 차세대 네트워크, QoS 트래픽 엔지니어링, 가상사설망, 무선 네트워크

이 미 정

1987년 이화여자대학교 전자계산학과 졸업(학사). 1989년 University of North Carolina at Chapel Hill 컴퓨터학과(공학석사). 1994년 North Carolina State University 컴퓨터공학과(공학박사). 1994년~현재 이화여자대학교 공과대학 컴퓨터공학과 교수. 관심분야는 프로토콜 설계 및 성능 분석, 멀티미디어 전송을 위한 트래픽 제어, 인터넷 QoS, 트래픽 엔지니어링, 무선 이동 네트워크, Ad-hoc 네트워크