碩士學位論文

P2P 스트리밍 시스템 성능 평가를 위한 NS2 기반 시뮬레이터 개발

2007 年

漢城大學校 一般大學院

컴퓨터 工學科

컴퓨터 工學 專攻

金 惠 善

碩 士 學 位 論 文
指導敎授 黃 琦 太

P2P 스트리밍 시스템 성능 평가를 위한 NS2 기반 시뮬레이터 개발

NS2 based Simulator for Performance Evaluation of P2P Streaming Systems

2006年 12月 日

漢城大學校 一般大學院

컴퓨터 工學科

컴퓨터 工學 專攻

金 惠 善

碩 士 學 位 論 文
指導敎授 黃 琦 太

P2P 스트리밍 시스템 성능 평가를 위한 NS2 기반 시뮬레이터 개발

NS2 based Simulator for Performance Evaluation
of P2P Streaming Systems

위 論文을 컴퓨터工學 碩士學位論文으로 提出함

2006年 12月 日

漢城大學校 一般大學院

컴퓨터 工學科

컴퓨터 工學 專攻

金 惠 善

金惠善의 工學 碩士學位論文을 인정함

2006年 12月 日

심사위원장 정 인 환 (인)

심사위원 황 기 태 (인)

심사위원 김 일 민 (인)

목 차

표 목 차

그 림 목 차

요 약

인터넷 스트리밍 시스템은 스트리밍 소스를 공급하는 미디어 서버와 이로부터 미디어 스트림을 받아 분배하는 스트리밍 서버, 그리고 스트리밍 단말기들로 구성되며, 기존에는 하나의 스트리밍 서버에 다수의 단말기들이 직접적으로 연결되는 방식을 취하고 있다. 이러한 클라이언트-서버 형태의 중앙 구조에서는 스트리밍 서버에 대한 트래픽 집중으로 병목 현상이 발생하며, 한 스트리밍 서버의 용량에 따라 스트리밍 단말기의 개수가 제한되는 등 확장성 및 수용 능력의 한계라는 단점을 근본적으로 가지고 있다. 이러한 문제를 극복하여 확장성을 제공하기 위해 주로 파일 공유에 사용된 P2P 분산 아키텍처를 이용하는 P2P 스트리밍 시스템에 대한 연구가 최근 들어 진행되고 있다.

그러나 P2P 방식을 이용한 인터넷 스트리밍 시스템을 설계, 구현, 테스트하기 위해서는 실제 많은 컴퓨터들이 필요하며, 네트워크의 다양한 구성이나 트래픽 변화에 따른 실험을 하기에는 현실적인 어려움이 있다. 그러므로 본 논문에서는 다양한 연구 및 실험을 지원하는 P2P 스트리밍 시스템에 대한 시뮬레이터 *P2PStreamSim*을 개발하였다. P2P 스트리밍 시스템에 대한 모델을 정의하고, 시간 모델과 동작 모델을 분석하였으며 P2P 스트리밍 시스템의 성능 지수들을 정의하였다. 또한 네트워크 시뮬레이터를 지원하는 NS2 시뮬레이션 도구를 이용하여 P2P 스트리밍 시스템 시뮬레이터 *P2PStreamSim*을 설계 및 구현하였으며, 테스트 P2P 스트리밍 시스템을 사례로 적용하여 시뮬레이터의 동작을 검증하고 성능을 평가하였다.

제 1 장 서론

1.1 연구 동기

최근 인터넷상에서 Peer-to-Peer(이하 P2P) 컨텐츠 공유 프로그램이 활발하게 사용되면서 P2P에 대한 관심이 더욱 증가되고 있다. P2P 시스템은 독립된 피어(peer computer)들 사이의 자원을 공유하는 분산 시스템으로서 주로 파일 공유에 많이 사용되어 왔다. 여기서 피어란 클라이언트 또는 서버의 기능을 가진 단말기로서, P2P 시스템에 참여하는 모든 사용자 컴퓨터 혹은 장치가 피어가 될 수 있다. 그러므로 피어는 컨텐츠를 수신하여 필요에 따라 재생하거나 동시에 다른 피어에게 수신한 컨텐츠를 전달하는 릴레이 또는 서버의 역할을 한다[1,2].

한편 네트워크 속도의 향상 및 멀티미디어에 대한 컴퓨팅 처리 능력이 향상됨에 따라 화상 채팅, 비디오 회의, VOD(Video On Demand), E-learning 등 실시간 스트리밍에 대한 요구 및 응용 분야들이 많이 제시되고 지난 10년 동안 많은 연구들이 진행되어 왔다[3,4,5].

최근에는 인터넷 방송 등 여러 인터넷 스트리밍 시스템들이 개발 및 활용되고 있다. 인터넷 스트리밍 시스템은 스트리밍 소스를 공급하는 미디어 서버와 이로부터 미디어 스트림을 받아 분배하는 스트리밍 서버, 그리고 스트리밍 단말기들로 구성되며, 기존에는 하나의 스트리밍 서버에 다수의 단말기들이 직접적으로 연결되는 방식을 취하고 있다. 이러한 클라이언트-서버 형태의 중앙 구조에서는 스트리밍 서버에 대한 트래픽 집중으로 병목 현상이 발생하며, 한 스트리밍 서버의 용량에 따라 스트리밍 단말기의 개수가 제한되는 등 확장성 및 수용 능력의 한계라는 단점을 근본적으로 가지고 있다.

최근 들어 인터넷 스트리밍 시스템에 P2P 네트워크를 이용함으로써 이

러한 단점을 해결하는 연구들이 진행되고 있다. PeerCast[6]와 AnySee[7]는 P2P 스트리밍 시스템을 실제 구현하였으며, T. Hama[8] 등은 P2P의 피어가 이탈함에 따라 하위 피어들의 스트림 끊김 현상을 해결하는 연구를 수행하였다. 또한 CoopNet[9]는 스트리밍 서버의 병목 현상을 완화시키기 위한 연구를 하였다. 아직 P2P 스트리밍 시스템에 대한 체계적인 설계, 구현, 성능 평가 문제에 대한 해결 등이 미비한 상태이며 많은 연구과제들이 남아있다.

P2P 방식을 이용한 인터넷 스트리밍 시스템을 설계, 구현, 테스트하기 위해서는 실제 많은 컴퓨터들이 필요하며 네트워크의 다양한 구성이나 트래픽 변화에 따른 실험을 하기에는 현실적인 어려움이 있다. 그러므로 P2P 스트리밍 시스템을 설계 개발하기에 앞서 경비와 시간을 절감하여 설계의 정확성을 검증하기 위해 시뮬레이터의 개발이 선행적으로 필요하다. 이에 본 논문에서는 P2P 스트리밍 시스템의 시뮬레이터 ***P2PStreamSim*** 을 개발하였다.

1.2 연구 목적

본 논문은 P2P 스트리밍 시스템의 개발과 성능 평가를 위해 선행적으로 필요한 P2P 스트리밍 시스템의 시뮬레이터 ***P2PStreamSim***을 개발하는 데 목적이 있다. 본 논문의 세부적인 목적은 다음과 같다.

- **P2P 스트리밍 시스템 모델 정의**

  P2P 스트리밍 시스템의 구조적 모델, 동작 모델, 시간 모델, 성능 모델을 정의한다.

- **P2P 스트리밍 시스템 시뮬레이터 설계 및 구현**

  네트워크 시뮬레이션을 위한 기본 프로토콜과 체계를 제공하는 NS2(Network Simulator version 2)[10]를 이용하여 P2P 스트리밍 시스템 시뮬레이터 ***P2PStreamSim***을 설계 및 구현한다.

- **테스트 시스템 성능 평가 및 분석**

  시뮬레이터의 정확성 및 효용성을 검증하기 위해 테스트 시스템에 대한 성능 평가를 수행한다. 이를 위해 P2P 스트리밍 시스템에 대한 성능 지수를 파악하고, 시뮬레이션을 통해 실험 결과를 분석한다.

1.3 논문 구성

본 논문의 구성은 다음과 같다.

- 2장 연구 배경

  P2P 시스템과 NS2에 대해 기술한다.

- 3장 P2P 스트리밍 시스템 모델링

  P2P 스트리밍 시스템의 모델을 정의하고 시간 모델과 동작 모델을 분석하며, P2P 스트리밍 시스템에 대한 성능 지수를 분석 및 정의한 내용에 대해 기술한다.

- 4장 P2PStreamSim : P2P 스트리밍 시스템 시뮬레이터

  P2P 스트리밍 시스템을 위한 시뮬레이터 ***P2PStreamSim***을 설계 및 구현한 내용에 대해 기술한다.

- 5장 시뮬레이션 및 성능 평가

  테스트 P2P 스트리밍 시스템을 정의하여 시뮬레이션을 수행한 결과에 대해 기술한다.

- 6장 결론 및 향후 연구

제 2 장 연구 배경

2.1 P2P 시스템

2.1.1 P2P 시스템 개요

1999년 5월, 파일 공유를 위한 Napster[11] 소개로 시작된 P2P 네트워킹 기술은 분산 컴퓨팅과 인터넷 전화에 성공적으로 적용됨에 따라 현재 가장 관심 있는 인터넷상 새로운 통신방식으로 떠오르고 있다. P2P 응용에 대한 사용 증가세는 WWW 보다 빠른 성장을 보이고 있다. P2P 기술은 모든 형태의 분산 자원 접근에 사용될 수 있으며 인터넷 기반 응용에 새로운 가능성을 제시하고 있다.

P2P 시스템은 네트워크 환경에서 집중화된 서비스 개념 없이 분산 자원의 공유를 목적으로 동등한 자격을 가진 독립적인 피어로 이루어진 분산 시스템으로 정의된다[2].

P2P 방식은 서버의 개입 여부에 따라 순수(Pure) P2P와 혼합형(Hybrid) P2P로 구분되며 그림 2.1과 같다.

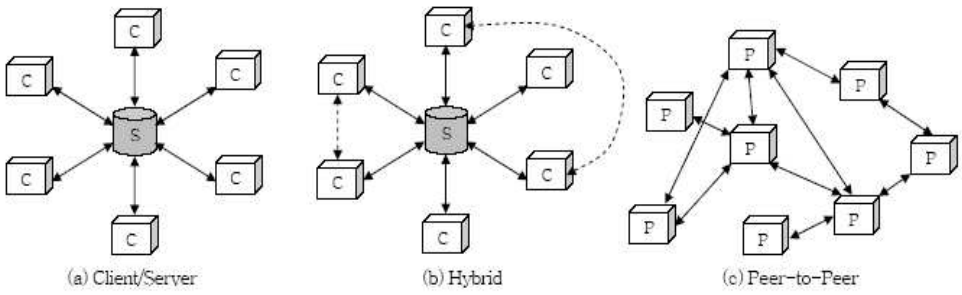

그림 2.1 시스템의 구조적 분류

순수 P2P(c)는 중앙 서버의 개입 없이 사용자들끼리 직접 연결되고, 혼합형 P2P(b)는 사용자간의 정보검색이나 편리성을 위해 서버가 개입하는 형태이다. 예를 들어 Gnutella[12]는 순수 P2P로 동작하며, Napster와 소리바다는 혼합형 P2P에 속한다. P2P 기반의 서비스에는 사용자 간의 직접적인 파일 공유, 상호 메시지 전달, 상대방의 CPU나 메모리 등의 PC 자원 공유 및 분산 컴퓨팅이 있다.

P2P 관련 표준화 활동은 IETF(Internet Engineering Task Force), ITU-T(ITU Telecommunication Standardization Sector)와 같은 국제 표준화 기구들과 Sun Microsystems와 같은 기업들을 중심으로 이루어지고 있다. IETF 조직 내의 p2prg(P2P Research Group)에서는 P2P 표준화 작업의 기초를 제공하기 위한 연구를 진행하고 있다. 그리고 Sun Microsystems에서는 현재 프로젝트 JXTA[13]를 통하여 공개 P2P 프로토콜 프레임워크를 개발하고 있으며, 어떠한 운영체제, 시스템에서도 운용될 수 있는 P2P 네트워크를 위한 프로토콜 표준을 제공하는 것을 목표로 하고 있다[14].

2.1.2 P2P 응용 분야

가. 정보 공유 분야

P2P 기반 자율 구성 조직 내에서 피어 및 자원의 존재 여부 정보(presense information) 제공은 P2P의 기본 기능이다. 이를 통하여 다른 피어와 직접 연락하고 필요한 자원(웹 서비스, 정보, 스토리지, 프로세서 사이클 등)에 대해서 질의가 가능하다. P2P 기반 인스턴트 메시징 시스템은 본 기능을 활용한 대표적인 서비스이다.

Skype[15]는 중앙 국설 교환기 없이 presence 서비스를 이용하여 자신의 buddy list에 등록된 피어 간 인터넷 전화 서비스를 제공하고 있다.

나. 파일 공유 분야

파일 공유는 가장 대표적인 P2P 응용 서비스로 인터넷 트래픽의 70% 이상이 파일 교환으로 발생된다고 추정된다. P2P 환경에서 파일을 다운로드 받은 클라이언트 피어는 다른 피어에게도 해당 파일을 제공함으로써 서버 역할을 겸하게 된다.

P2P 파일 공유 서비스는 원하는 파일을 검색하는 방식에 따라, 플러딩(flooding) 요구 모델(Gnutella), 중앙 디렉토릭 모델(Napster), 문서 라우팅 모델(Freenet)등으로 분류된다.

Gnutella에서는 인접 피어로 검색 요구를 플러딩 시켜 원하는 파일을 찾는 방법으로써, 사전에 피어 검색 범위를 정의함으로써 검색 요구 메시지의 플러딩이 제한된다. Gnutella 모델에서는 검색 피어 범위에 따라 검색 메시지가 지수 함수적으로 증가하므로 큰 규모의 망에서는 비효율적이다.

Napster는 중앙 서버에 의하여 인덱스 서비스가 제공되는 혼합형 P2P 시스템이다. 피어가 Napster 망에 접속 시 해당 피어가 제공 가능한 파일이 중앙 서버에 등록된다. 따라서 검색 요구 시 중앙 서버는 해당 파일을 보유한 피어 리스트를 제공한다.

Freenet[16]에서는 파일이 의도적으로 별도의 피어에 저장되는 방식이다. 본 방법은 대규모 망에 적용 가능하나, 검색 과정이 상대적으로 복잡하다는 단점이 있다.

다. 대역폭 효율화 분야

기존 클라이언트-서버 형태의 중앙 구조에서는 서버의 트래픽 집중으로 인한 병목 현상 및 큐잉 지연 문제점을 내포하지만, P2P 구조에서는 각 피어로 분산된 트래픽 경로를 이용 가능하게 함으로써 효율적인 부하 분산이 가능하다.

다수의 피어에 파일이 복제된 상태에서 P2P 방식으로 효율적인 부하 분

산을 이용하는 개념은 미디어 스트리밍 및 VOD(Video On Demand) 분야에 많이 적용된다. 예로써, PeerCast[6], Peer-to-Peer-Radio[17], SCVI.net[18] 등이 있다.

라. 데이터 스토리지 분야

망 내 컴퓨터 클러스터로 구성된 P2P 저장 네트워크를 이용하여 각 컴퓨터에서 제공 가능한 저장 공간을 사용함으로써, 기존 저장 공간 활용 및 관리 측면에서 효율화가 가능하다. 이를 이용한 시스템으로는 PAST[19], Pasta[20], OceanStore[21] 등이 있다.

마. 컴퓨팅 파워 공유 분야

P2P를 이용하여 각 컴퓨터에 분산된 여러 프로세서 사이클을 묶음으로써 최고의 컴퓨팅 파워를 제공할 수 있다. 이와 같이 P2P 기반으로 가상 조직 내 분산된 컴퓨팅 자원을 공유하는 기술이 그리드 컴퓨팅(grid computing)이며, 이를 응용한 최초의 프로젝트가 SETI@home[22] 이다.

2.2 NS2(Network Simulator version 2)

2.2.1 NS2 소개

콜롬비아 대학에서 개발된 시뮬레이션 테스트베드인 NEST를 기반으로 UC 버클리에서 1988년에 REAL 네트워크 시뮬레이터를 개발하였고, 이것을 기반으로 1989년 LBNL(Lawrence Berkely National Laboratory)이라는 네트워크 연구 그룹은 NS1이라는 네트워크 시뮬레이터를 발표하였다. 결국 1995년 VINT(Virtual InterNetwork Testbed) 프로젝트의 일환으로 DARPA(the Defense Advanced Research Projects Agency)에서 자금 지

원을 받아 NS1 시뮬레이터가 완성되었다. 1996년 시뮬레이션 언어인 Tcl(Tool Command Language)[23] 대신 MIT에서 개발하여 발표한 OTcl(Object Tcl)을 사용하여 NS1의 기능을 더욱 향상 시킨 것이 NS2 이다[24,25].

NS2는 TCP, UDP, HTTP 등과 같은 TCP/IP 프로토콜 패밀리와 라우팅 프로토콜, 그리고 멀티캐스팅 프로토콜 등과 같은 다양한 인터넷 프로토콜을 시뮬레이션 할 수 있다. 그리고 Ad Hoc 네트워크, 이동 통신망의 기지국(Base Station) 모델, WLAN, Mobile-IP 관련 프로토콜, 위성 네트워크(Satellite Network) 등과 같은 무선 네트워크까지 지원할 수 있는 매우 적용 범위가 넓은 네트워크 시뮬레이터이다.

2.2.2 NS2 구조

그림 2.2는 사용자 관점에서 본 NS2 구조이다.

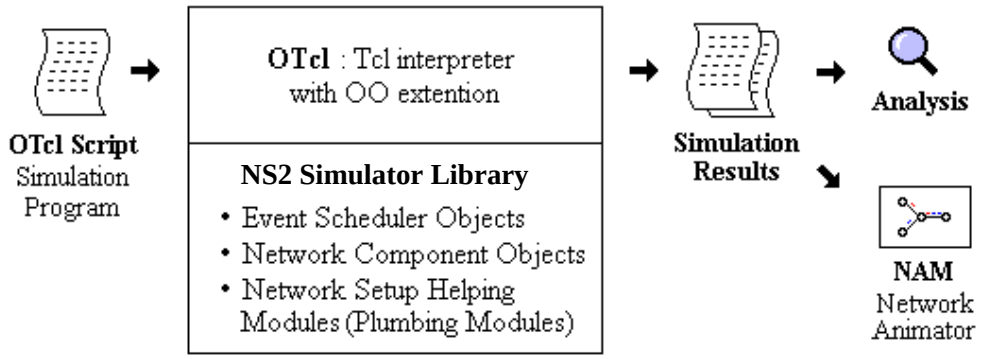

그림 2.2 사용자 관점에서 본 NS2 구조

먼저 사용자는 OTcl 스크립트 언어를 사용하여 사용자 인터페이스와 네트워크 토폴로지 구성에 대한 시뮬레이션 프로그램을 작성한다. 시뮬레이션 프로그램은 OTcl 인터프리터와 NS2 시뮬레이터 라이브러리(이벤트 스

케줄러, 네트워크 컴포넌트 등)를 이용하여 시뮬레이션이 실행된다. 시뮬레이션 결과는 트레이스(trace) 파일로 저장되거나 NAM(Network Animator)[26]라는 애니메이션 도구를 이용하여 화면으로 결과를 볼 수 있다.

NS2에서는 Tcl 스크립트 언어에 객체 지향의 개념을 추가한 OTcl과 C++ 언어를 사용한다. 대부분의 네트워크 연구 분야는 다양하게 변하는 파라미터 및 구성, 매우 복잡한 시나리오에 의해서 개발된다. 이러한 경우 모델을 변경하거나 재 구동 할 때에 소요되는 반복 시간이 중요하다. OTcl은 매우 느리게 구동하지만 빨리 변경할 수 있어, 시뮬레이션 구성에 있어서 매우 이상적이다. 반면 C++는 구동이 빠르지만 변경에는 다소 느리므로 자세한 프로토콜 구현에 적합하다. 따라서 OTcl은 사용자 인터페이스와 네트워크 토폴로지에 대한 매개 변수 등을 정의하며, C++은 주로 네트워크 구성 요소들을 정의한다. 그리고 OTcl과 C++ 언어를 서로 연동시키기 위해 인터프리터를 사용한다.

가. 이벤트 스케줄러

스케줄러는 가장 먼저 도착한 이벤트를 선택하여 동작하며, 단위는 초(second)를 이용한다. NS2는 단일 쓰레드로 구현되며 유일한 쓰레드가 바로 스케줄러이다. 그러므로 스케줄러는 한 번에 오직 한 가지 이벤트만을 처리하며 대기하고 있는 이벤트를 순차적으로 처리한다. 즉 만약 둘 이상의 이벤트를 처리하도록 스케줄링 하였다면, 스케줄러는 먼저 스케줄링 된 이벤트를 먼저 처리(First Scheduled First dispatched)하도록 한다.

그림 2.3은 스케줄러의 동작 원리를 보여주고 있다. 네트워크 오브젝트는(Network Object)는 노드나 링크 등과 같은 네트워크 구성 요소로서 이벤트를 등록하는 주체이다. 네트워크 오브젝트는 표 2.1과 같은 변수 값을 설정한 Event 객체를 생성하여 스케줄러에 삽입한다.

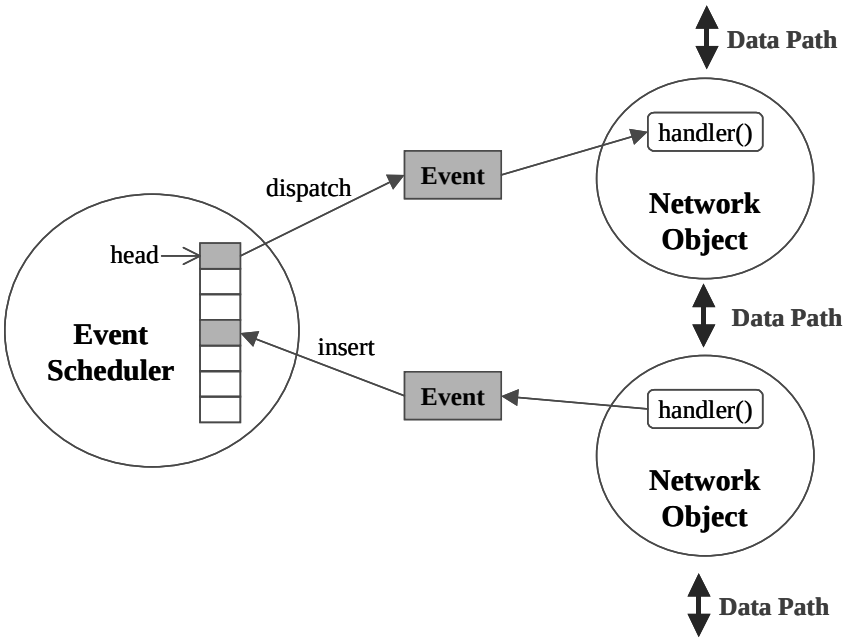

그림 2.3 스케줄러의 동작 원리

표 2.1 Event 객체의 멤버 변수

멤버 변수	설명
uid_	이벤트를 식별하는 유일한 ID
time_	이벤트를 실행할 시간
handler_	이벤트가 준비되었을 때 실행할 메소드

스케줄러는 순차적으로 이벤트를 실행하며, 해당 시간에 네트워크 오브젝트 안에 있는 hander_에 등록된 메소드를 호출하여 이벤트를 실행한다. 여기서 네트워크 오브젝트 간의 데이터 경로(Data Path)는 이벤트 경로와 다르다. 데이터 경로는 이벤트가 실행된 이후, 네트워크 오브젝트 안에 있는 다음과 같은 멤버 함수에 의해 패킷을 전달한다.

- 송신자 멤버 함수 : send(Packet *p) { target_->recv(p) }
- 수신자 멤버 함수 : recv(Packet*, Handler *h=0)

이벤트를 등록하는 방법에는 OTcl 명령어를 사용하는 방법과 타이머를 사는 방법이 있다.

다음은 OTcl 스크립트 코드에서의 이벤트 등록 과정이다.

- Simulator 클래스로부터 오브젝트 인스턴스 생성

  set ns [new Simulator]

- 이벤트 스케줄링

  $ns at <time> <event>

  <time> : 시뮬레이션 실행 시간

  <event> : 실행할 메소드

- 시뮬레이션 실행

  $ns run

타이머는 timer-handler.{h,cc}에 정의되어 있으며, 표 2.2는 TimerHandler 클래스의 멤버 함수이다.

표 2.2 TimerHandler 클래스의 멤버 함수

멤버 함수	기능
void sched(double delay)	delay 초 후에 타이머가 만료(expire) 되도록 스케줄링
void resched(double delay)	sched() 와 비슷함, 타이머가 PENDING 상태일 수 있음
void cancel()	pending 타이머를 취소한다.
virtual void expire(Event *e)=0	사용자에 의해 반드시 구현되어야 함
virtual void handler(Event *e)	이벤트를 실행

그림 2.4는 타이머를 사용하여 이벤트를 등록하는 예로 2.0초 후에 “Timeout″ 이라는 글자가 출력된다. resched() 함수에 의해 2.0초 후에 CppTimer가 만료되면, CppTimer 내의 expire() 함수가 호출되고, NewCpp 내의 timeout() 함수가 호출되어 ”Timeout″ 이 출력된다.

```
/* CppTimer */
class CppTimer : public TimerHandler {
public:
        CppTimer(NewCpp *a) : TimerHandler() {a_=a;}
        virtual void expire(Event *e);
protected;
        NewCpp *a_;
};
void CppTimer::expire(Event *e)
{
        a_->timeout();   /* NewCpp->timeout() 호출 */
}

/* NewCpp */
class NewCpp
{
        void timeout();
        CppTimer timer_;
};
NewCpp::NewCpp : timer_(this)
{
        timer_.resched(2.0); /* 2.0초 후에 타이머는 expire 됨 */
}
void NewCpp::timeout()
{
        printf("Timeout\n");
}
```

그림 2.4 타이머 사용 예제

나. 네트워크 컴포넌트

(1) 노드

노드는 라우터, 허브, 기지국 등과 같은 네트워크에서의 중계용 장비를 말한다. 노드는 노드 엔트리 오브젝트(Node Entry object)와 분류기 오브젝트(Classifier object)로 구성되고, 유니캐스트(unicast) 노드와 멀티캐스트(multicast) 노드가 있다.

유니캐스트는 오직 목적지 네트워크 디바이스 한곳으로만 데이터를 전달하는 형태를 말하며, 멀티캐스트는 패킷 하나를 복사한 똑같은 패킷들을 다수의 네트워크 디바이스로 전달하는 것이다.

유니캐스트 노드는 어드레스 분류기(Address Classifier)와 포트 분류기(Port Classifier) 두 개의 TclObject로 구성되며 그림 2.5와 같다.

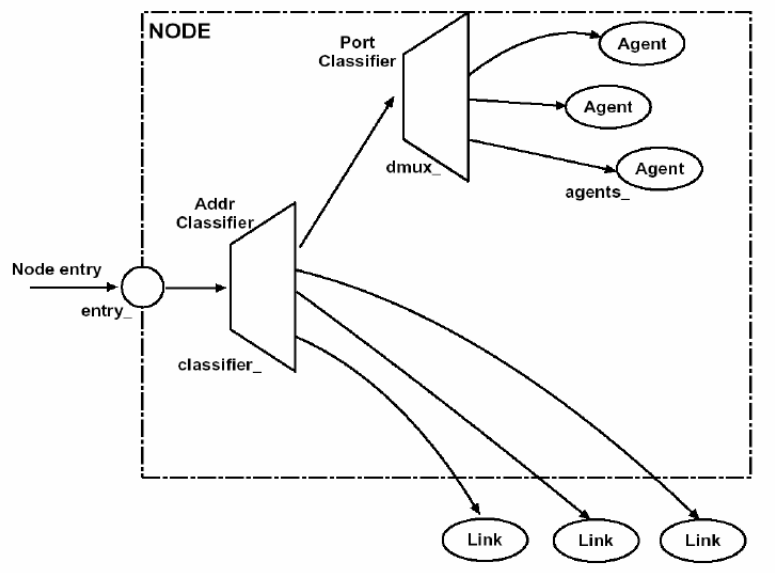

그림 2.5 유니캐스트 노드

어드레스 분류기는 받은 패킷을 올바른 에이전트 또는 링크로 전달하는 기능을 한다. NS2에서는 유니캐스트 노드를 디폴트 노드로 사용하고 있다. Tcl 스크립트 코드에서 노드를 생성하는 형식은 다음과 같다.

```
set ns [new Simulator]
$ns node
```

(2) 링크와 큐

링크는 노드를 연결하는 네트워크 컴포넌트 오브젝트로서, 한쪽 방향으로만 데이터를 전달할 수 있는 단방향 링크(Simplex-Link)와 양방향으로 데이터를 전달할 수 있는 양방향 링크(Duplex-Link)가 있다.

Tcl 스크립트 코드에서 링크를 생성하는 형식은 다음과 같다.

```
set ns [new Simulator]
$ns simplex-link <src_node> <dest_node> <bandwidth> <delay> <queue_type>
        <src_node>      : 송신 노드
        <dest_node>     : 목적지 노드
        <bandwidth>     : 링크의 대역폭 (단위: Mbps)
        <delay>         : 링크의 지연 시간 (단위: ms)
        <queue_type>    : 큐의 타입을 지정
                Drop-Tail(First In First Out)
                RED(Random Early Detected)
                CBQ(Class Based Queueing)
                WFQ(Weighted Fair Queueing)
                SFQ(Stochastic Fair Queueing)
```

(3) 에이전트

에이전트는 TCP, UDP와 같은 전송 계층 프로토콜로서 노드에 사용자가 원하는 기능을 수행할 수 있도록 하는 것이다. 노드를 생성하면 네트워크 계층까지 완성될 뿐, 전송 계층과 응용 계층 프로토콜이 노드에 없기 때문에 어떤 트래픽도 발생시키지 않는다. 따라서 에이전트를 생성하여 노드에 붙여야만 한다. 이 과정을 “Attachment"라고 한다.

다음은 에이전트를 생성하여 각 노드에 attach하는 예제를 보여준다. TCP 패킷을 전송할 송신 에이전트인 TCP, 수신 에이전트인 TCPSink를 생성한다. 생성한 각 에이전트를 노드 n0와 n1에 각각 attach 한 다음 TCP와 TCPSink를 서로 연결한다.

```
set tcp [new Agent/TCP]
set tcpsink [new Agent/TCPSink]
$ns attach-agent $n0 $tcp
$ns attach-agent $n1 $tcpsink
$ns connect $tcp $tcpsink
```

표 2.3은 NS2에서 지원하는 프로토콜 에이전트의 일부로서 OTcl에서 이용하는 에이전트의 이름을 각각 나타낸다.

표 2.3 프로토콜 에이전트 종류

프로토콜 에이전트	설명
TCP	“Tahoe" TCP sender
TCP/Reno	fast recovery 기능을 지원하는 "Reno" TCP sender
TCP/Sack1	SACK TCP sender
TCP/Fack	"forward" SACK sender TCP
TCP/FullTcp	2-way 트래픽을 지원하는 모든 기능을 갖춘 TCP
TCP/Vegas	"Vegas" TCP sender
TCP/Asym	비대칭 링크에 대한 실험적인 Tahoe TCP
TCPSink	Reno나 Tahoe TCP receiver
UDP	기본적인 UDP 에이전트
RTP	RTP(Real-time Transport Protocol)[27] sender와 receiver
RTCP	RTCP(Real Time Control Protocol) sender와 receiver
LossMonitor	Loss를 검사하기 위한 패킷 sink
Message	텍스트 메시지를 운반하기 위한 프로토콜
Null	폐기되는 패킷에 대한 소멸 Agent

(4) 어플리케이션

어플리케이션 계층은 TCP/IP 프로토콜 모델에서 전송 계층 위에서 구동하는 프로토콜로 구성된다. 이러한 어플리케이션 계층 프로토콜에는 FTP, Telnet, NFS, DNS, SNMP(Simple Network Management Protocol) 등이 있다.

다음은 어플리케이션을 생성하여 에이전트에 attach 하는 과정으로 FTP 어플리케이션을 생성 한 후, TCP 에이전트에 attach 한다. 생성된 ftp 어플리케이션은 0.0초에 트래픽 발생을 시작한다.

```
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 0.0 "$ftp start"
```

2.2.3 NS2 클래스 계층 구조

NS2는 OTcl 인터프리터를 전처리기로 이용하며, C++로 작성된 객체 지향 시뮬레이터이다. OTcl 인터프리트된 클래스 계층과 C++ 클래스 계층 사이에는 일대일 대응관계처럼 보인다. TclClass는 순수 가상 클래스로, C++ 클래스 계층 구조를 반영하기 위하여 인터프리트 된 계층을 구축한다.

OTcl 계층 구조나 C++ 계층 구조에 있는 기본 클래스(base class)는 TclObject로 그림 2.6과 같다.

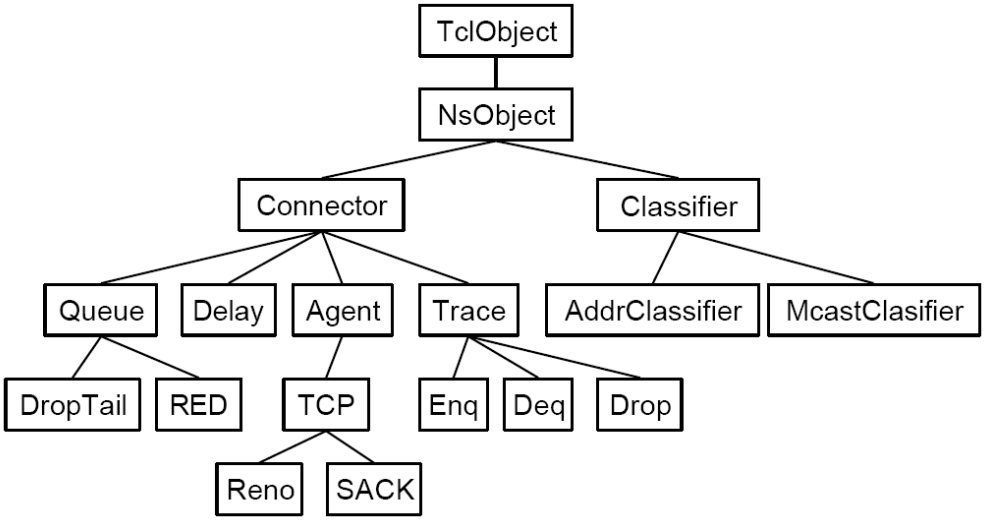

그림 2.6 NS2 클래스 계층 구조

가. OTcl linkage

그림 2.7은 TCP 에이전트를 생성할 경우 OTcl과 C++ 오브젝트와의 관계를 나타내며, C++코드는 그림 2.8과 같다. TclObject에 있는 모든 오브젝트는 인터프리터를 이용하여 사용자에 의해서 생성된다. 이와 동등한 쉐도우(shadow) 오브젝트는 C++ 오브젝트와 매우 밀접한 관계가 있다.

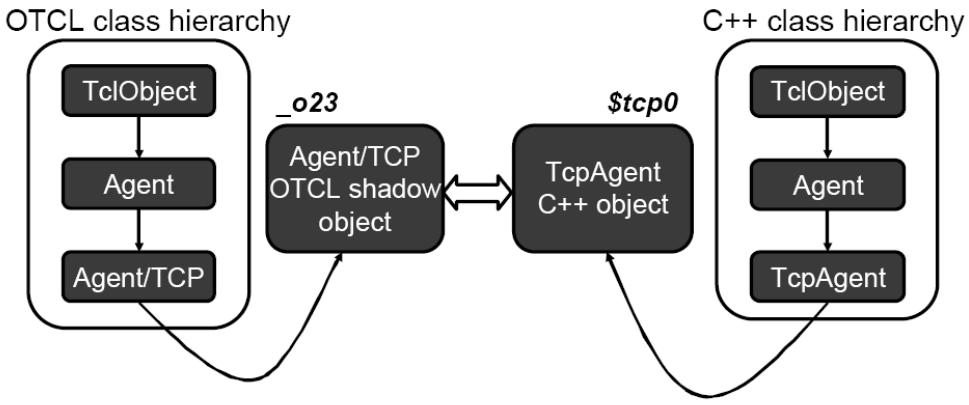

그림 2.7 TCP 에이전트 linkage

```
static class TcpClass : public TclClass {
public:
        TcpClass() : TclClass("Agent/TCP") {}
        TclObject * create(int, const char*const*){
                return(new TcpAgent());
        }
}class_tcp;
```

그림 2.8 TCP 에이전트 linkage 코드

TclClass 는 순수 가상 클래스로, C++ 클래스 계층 구조를 반영하기 위하여 인터프리트 된 계층을 구축하며 TclObject들을 초기화한다. 사용자는 “Agent/TCP” 이름으로 OTcl 코드에서 클래스를 생성하면 OTcl linkage인 TcpClass에 의해 TcpAgent(C++)가 생성된다. TcpAgent는 TclObject에서 파생된 Agent로부터 파생되었다. 그림 2.9는 TCP 오브젝트가 생성되는 과정을 보여준다.

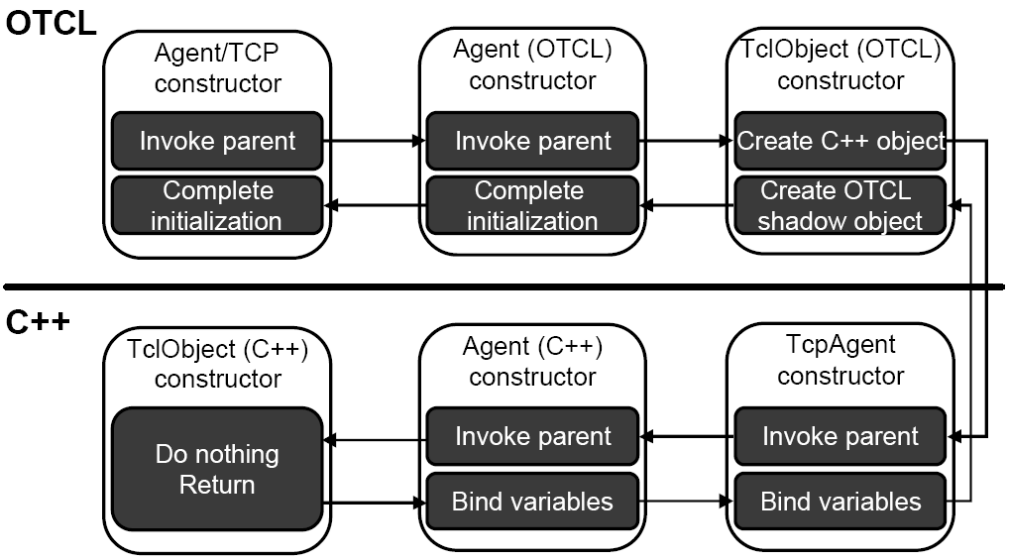

그림 2.9 TCP 오브젝트 생성 과정

2.2.4 패킷 헤더

NS2에서의 패킷은 각 프로토콜 계층의 헤더가 스택 형태로 구성되며 패킷 헤더 포맷은 그림 2.10과 같다.

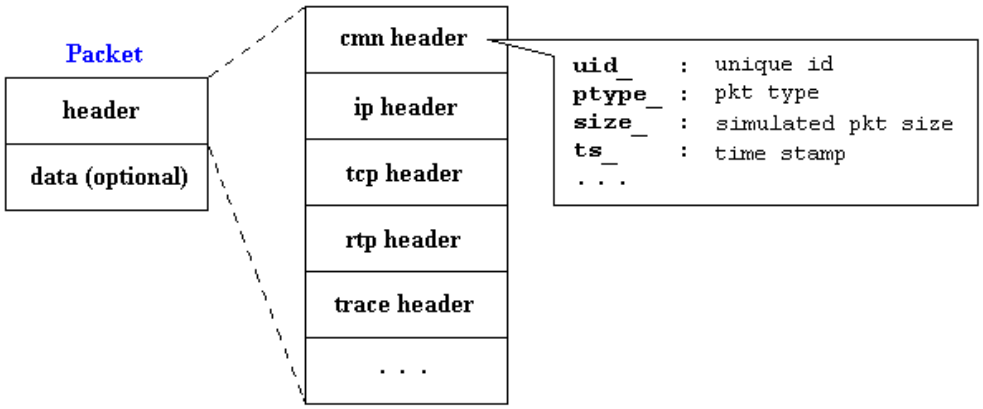

그림 2.10 NS2 패킷 헤더 포맷

NS2 패킷은 헤더들의 스택과 데이터 영역으로 구성된다. 패킷 헤더 포맷은 시뮬레이터 오브젝트(Simulator Object)가 생성될 때 초기화되며, IP 헤더, TCP 헤더, RTP 헤더(UDP는 RTP 헤더를 사용), trace 헤더와 같은 공통 헤더가 초기화되고, 이것이 모여서 헤더 스택을 구성한다. 에이전트에는 패킷을 하나 할당할 때 등록된 전체 헤더 스택을 모두 생성하며, 네트워크 오브젝트는 패킷 스택에 있는 임의의 헤더에 offset 값으로 액세스할 수 있다. 패킷의 데이터 공간은 실제 어플리케이션에서는 데이터를 할당하여 전송하지만, 시뮬레이션에는 의미가 없는 데이터를 전달하고 있기 때문에 null 값으로 구현되어있다. 사용자는 새로운 프로토콜을 정의하고 싶다면 패킷 헤더를 새로 만들거나 기존 패킷 헤더를 확장하여 사용하면 된다.

2.2.5 시뮬레이션 실행 예

가. 네트워크 구성

시뮬레이션을 하기 위해서는 먼저 네트워크 토폴로지를 구성해야 한다. 먼저 노드와 링크의 특성을 결정하고, 노드에서 이용하는 TCP, UDP 등과 같은 전송 계층 프로토콜을 결정한다. 그 다음으로는 FTP, HTTP, Telnet 등과 같은 어플리케이션 서비스 타입을 지정한다. 마지막으로는 시뮬레이션 시간을 결정하는 작업을 한다.

나. 시뮬레이션

그림 2.11은 CBR 형태의 트래픽을 전송하는 시뮬레이터의 구조이다. 먼저 도 노드는 양방향 링크로 연결되어 있으며, 송신 노드는 0번, 목적지 노드는 1번 노드이다. 링크의 네트워크 대역폭은 1Mbps이고 전송 지연 시간은 10ms 이다. 노드 0번에는 패킷을 전송하기 위한 UDP 에이전트가 존

재하며, 노드 1번에는 패킷을 수신하기 위한 Null 에이전트가 존재한다. CBR 어플리케이션은 CBR 형태의 패킷을 생성하여 UDP 에이전트를 통해 목적지 노드인 Null 에이전트로 전달한다.

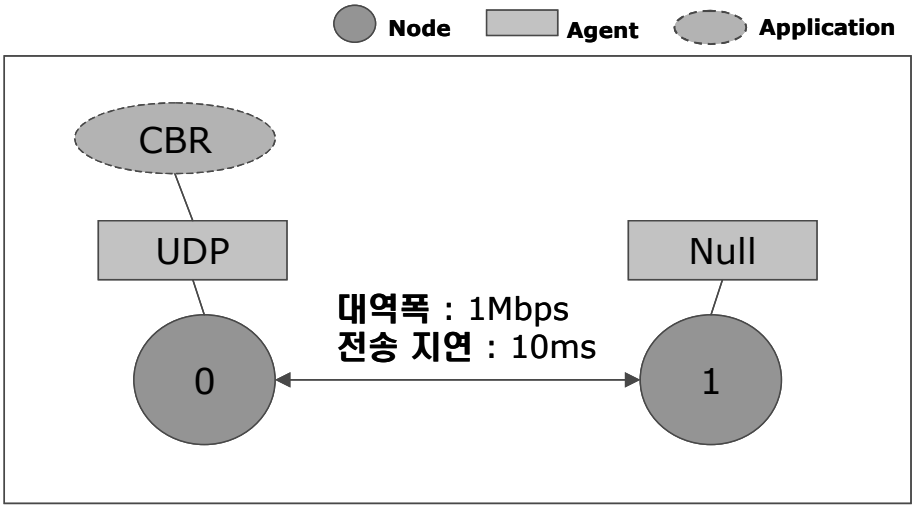

그림 2.11 CBR 시뮬레이터 구조

제 3 장 P2P 스트리밍 시스템 모델링

3.1 P2P 스트리밍 시스템 개요

3.1.1 인터넷 스트리밍 시스템 문제점

인터넷 스트리밍 시스템은 스트리밍 소스를 공급하는 미디어 서버와 이를 분배하는 스트리밍 서버, 그리고 스트리밍 단말기들로 구성되며, 기존에는 하나의 스트리밍 서버에 다수의 단말기들이 직접적으로 연결되는 방식으로 취하고 있다. 이러한 방식은 그림 3.1과 같이 한 스트리밍 서버의 용량에 스트리밍 단말기의 개수가 제한되는 단점을 근본적으로 가지고 있다.

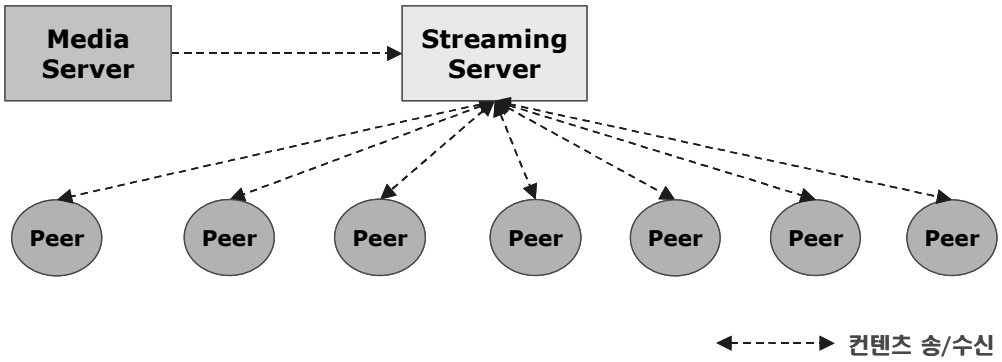

그림 3.1 인터넷 스트리밍 시스템

3.1.2 P2P 스트리밍 시스템의 필요성

본 논문에서는 인터넷 스트리밍 시스템의 한계점을 극복하기 위해 그림 3.2와 같이 독립된 피어들 사이의 자원을 공유하는 분산 시스템인 P2P 네트워크를 인터넷 스트리밍 시스템에 이용함으로써 해결하고자 한다.

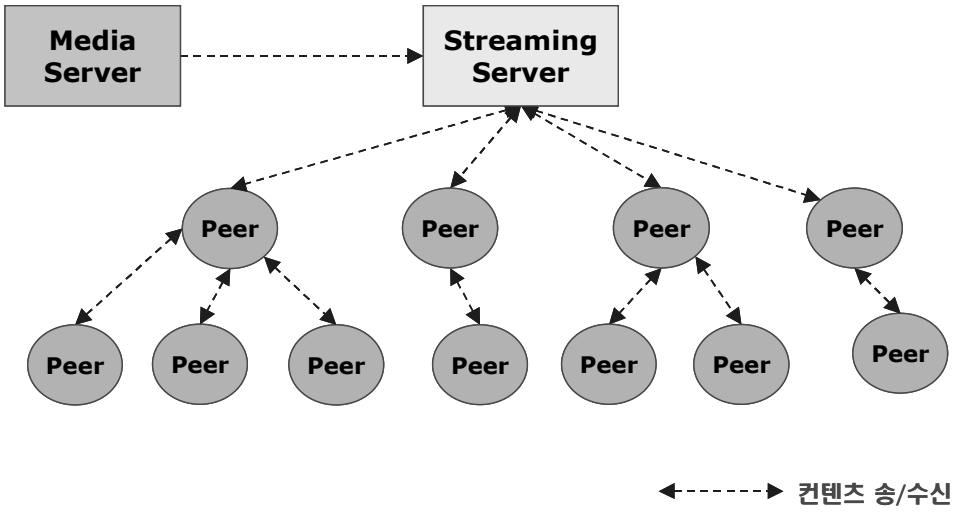

그림 3.2 P2P 스트리밍 시스템

3.1.3 P2P 기반 스트리밍 관련 연구

최근 들어 인터넷 스트리밍 시스템에 P2P 네트워크를 이용함으로써 이러한 단점을 해결하는 연구들이 진행되고 있다. PeerCast[6]는 P2P 방식의 인터넷 방송을 위한 오픈 소프트웨어로서 Gnutella 프로토콜을 사용하여 개발되었다. 실제로 TV 화면 등의 영상과 음악, 라디오 방송 등을 스트림 형식으로 전송하고 있다. T. Hama[8] 등은 P2P의 피어가 이탈함에 따라 하위 피어들의 스트림 끊김 현상을 해결하는 연구를 수행하였으며, CoopNet[9]은 스트리밍 서버의 트래픽 병목 현상을 완화시키는데 초점을 맞추고 있다. 이를 위해 다중 디스크립션 코딩(Multiple Description Coding)을 이용하여 스트리밍 서버에 문제가 발생하더라도 계속적인 서비스가 가능하도록 연구하였다. ZIGZAG[28]는 송신자와 수신자간의 단대단 지연 시간을 줄이고, 스트림 중단 되었을 때 빠른 시간 안에 서비스가 가능하도록 연구하였다.

3.2 P2P 스트리밍 시스템 모델

3.2.1 P2P 스트리밍 시스템 정의

본 논문에서는 P2P 스트리밍 시스템에 대해 다음과 같이 정의하고 그 모델의 구조 및 동작 방식 등을 기반으로 자세히 설명한다.

P2P 스트리밍 시스템 : Se, Ss, Sp, Proxy, PT, ST, Sl, F, Al 로 구성된 tuple이다.

Se : a set of encoding servers
Ss : a set of streaming servers
Sp : a set of peers
Proxy : a proxy server
PT : a set of protocols for connection between peers
ST : a set of streams, ie. video and/or audio
Sl : is a set of network link properties between peers
F : is a set of functions, i.e., arrival, departure, and reconfigure
Al : a set of algorithms, i.e., parent peer selection, etc.

그림 3.3은 P2P 스트리밍 시스템의 한 사례로서 P2P 스트리밍 시스템은 하나의 인코딩 서버, 스트리밍 서버, 프록시 서버, N개의 피어들로 구성된다. 인코딩 서버는 비디오 또는 오디오 데이터를 압축하여 스트리밍 서버로 컨텐츠를 전송한다. 스트리밍 서버는 부모 피어의 역할로서, 연결된 자식 피어들에게 수신한 컨텐츠를 전달해준다. 프록시 서버는 피어의 도착과 이탈 시 피어에게 적당한 부모 피어를 탐색하여 알려주며, 네트워크 토폴로지를 재구성한다.

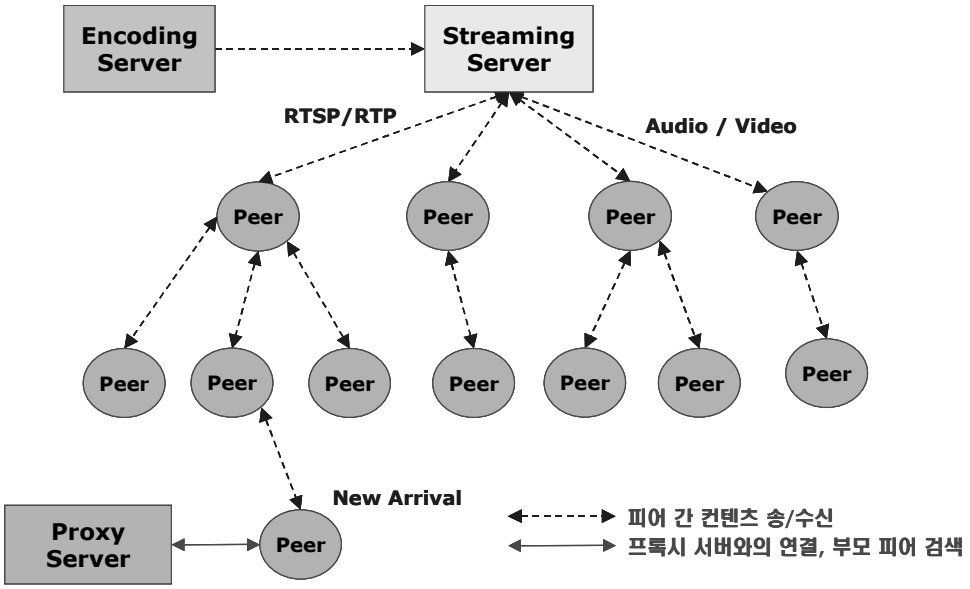

그림 3.3 본 논문에서 정의한 P2P 스트리밍 시스템 모델

3.2.2 P2P 스트리밍 시스템 분석

P2P 스트리밍 시스템을 위한 시뮬레이터를 개발하기 위해 시간 모델과 동작 모형을 정의하고, 시스템의 동작 과정동안 발생하는 시간의 흐름과 시간의 양에 대해서 분석한 내용을 보인다. 또한 P2P 스트리밍 시스템의 성능 지수들을 정의한다.

가. 동작 모델

(1) 스트리밍 및 릴레이

인코딩 서버로부터 말단 피어까지 스트리밍 서비스가 이루어지는 과정으로서 그림 3.4와 같이 묘사된다. 인코딩 서버는 일정한 시간간격으로 RTP 패킷을 생성하여 스트리밍 서버로 전달하며, 스트리밍 서버는 이를 연결된 피어 1(p1), 피어 2(p2), 피어 3(p3), 피어 4(p4)에게 분배한다. 피어는 스트리밍 서버로부터 수신한 RTP 데이터를 버퍼링한 후, 한 프레임씩

디코딩하여 재생한다. 또한 피어는 연결된 자식 피어들에게 각각 RTP 데이터를 릴레이 해주는 역할을 한다.

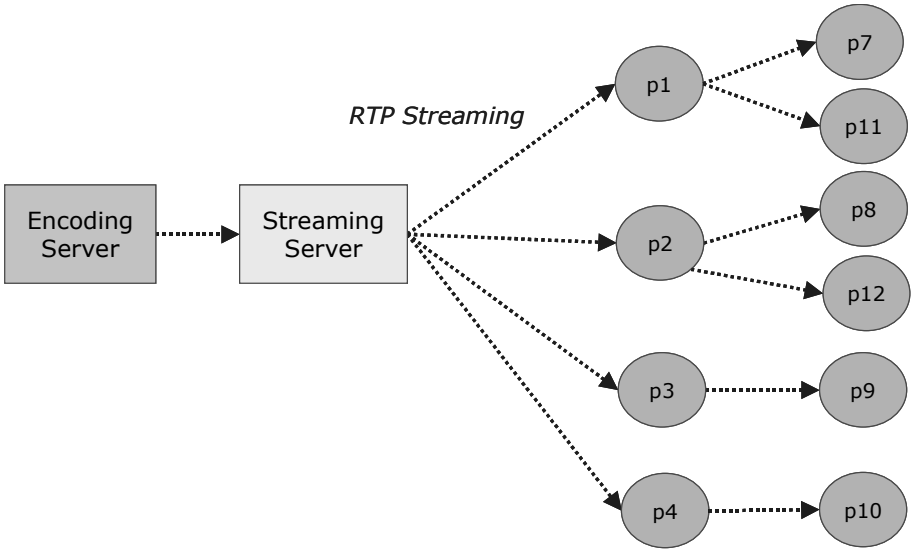

그림 3.4 스트리밍 및 릴레이

(2) 피어 참여

피어가 시스템 내에 도착하여 자신의 부모 피어로 접속되는 과정으로서 그림 3.5와 같이 묘사된다. 새로 도착한 피어 10(p10)은 프록시 서버에게 자신의 존재를 알린다. 프록시 서버는 적당한 부모 피어를 탐색하여 도착한 피어 10에게 알려준다. 피어 10은 프록시 서버로부터 할당 받은 부모 피어 4(p4)로 접속하여 RTSP(Real Time Streaming Protocol)[29] 세션을 설정한 후 RTP 데이터를 수신한다. 피어 10은 수신한 RTP 데이터를 버퍼에 저장하고 한 프레임씩 읽어 디코딩 후 재생한다.

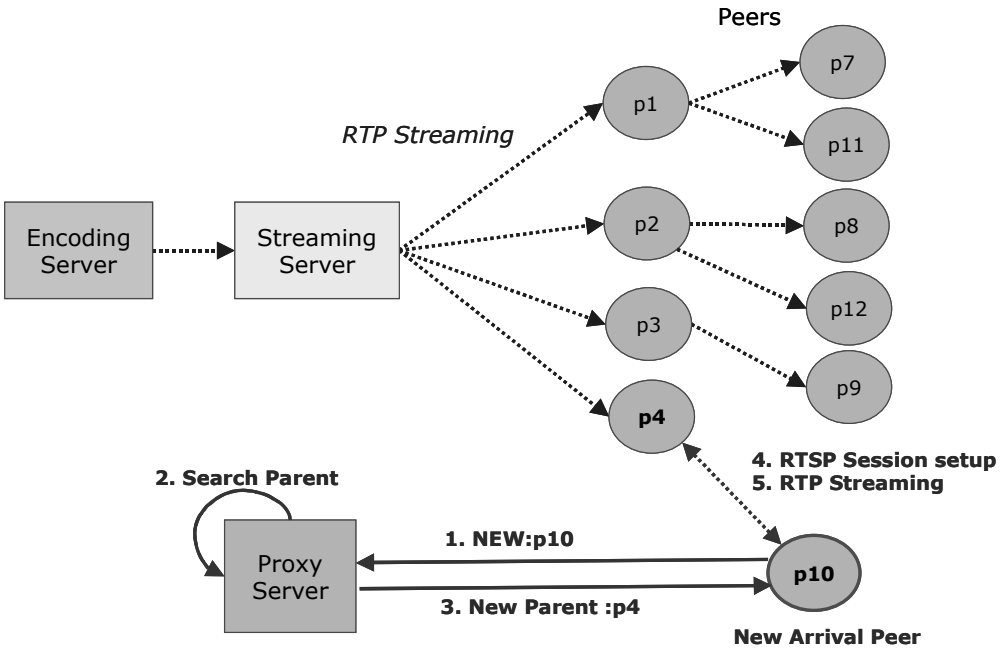

그림 3.5 피어 참여

(3) 피어 이탈 및 재배치

피어가 네트워크에서 이탈할 시, 연결된 자식 피어들이 새로운 부모 피어로 접속되는 과정으로서 그림 3.6과 같이 묘사된다. 이탈 하고자 하는 피어 3(p3)은 자신의 이탈 상황을 프록시 서버와 부모 피어(스트리밍 서버)에게 알린다. 프록시 서버는 현재 P2P 네트워크에 참여한 피어 중 적당한 부모 피어를 검색하여 이탈 피어의 자식 피어 9(p9)에게 새로운 부모 피어(스트리밍 서버)를 알려준다. 자식 피어 9는 새로운 부모 피어인 스트리밍 서버로 접속하여 RTSP 세션을 설정한다. 자식 피어 9는 스트리밍 서버로부터 RTP 데이터를 수신하여 디코딩 후 재생한다.

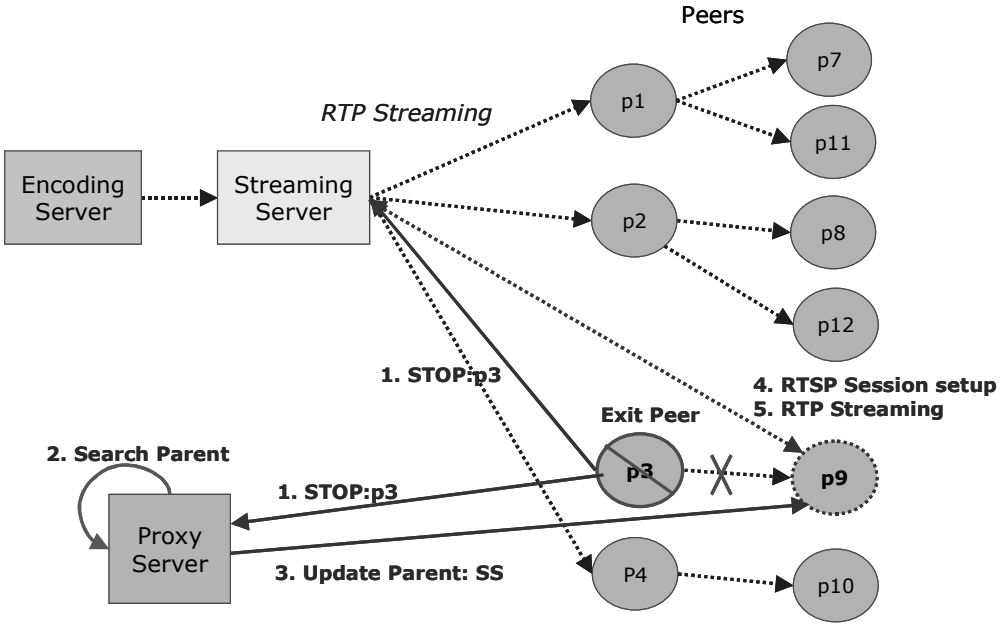

그림 3.6 피어 이탈 및 재배치

나. 시간 모델

(1) 시간 요소

P2P 스트리밍 시스템 시뮬레이터를 개발하기에 앞서 다음과 같은 시간적 요소를 분석한다. P2P 스트리밍 시스템에서의 시간 흐름과 시간적 양을 분석하며, 이는 시뮬레이션하기 위해 반드시 요구되어진다.

- $T_{connection}$: 임의의 피어가 접속을 시작하여 자신의 부모 피어가 결정되고 그 부모 피어에 연결이 완료될 때까지의 지연 시간. 이 시간은 피어가 프록시 서버에게 자신의 부모 피어를 결정하여 줄 것을 요청하고 부모 피어가 결정되면 이 부모 피어에 접속하는데 걸리는 전체 시간이다. 이 시간은 $T_{getserver} + T_{p2pconnection} + T_{initialframe}$ 시간의 합으로 표현된다.

- $T_{getserver}$: 피어가 프록시 서버에게 부모 피어를 알려줄 것을 요청한 시점으로부터 부모 피어를 통보받을 때까지의 시간 경과.
- $T_{p2pconnection}$: 임의의 피어가 자신의 부모 피어의 주소를 알고 있는 상황에서 자신의 부모 피어에 접속하여 RTSP 세션 설정을 완료할 때까지의 시간.
- $T_{initialframe}$: 피어가 자신의 부모 피어에 접속된 후에 처음으로 하나의 프레임을 성공적으로 받을 때까지의 경과 시간. 이 시간은 $T_{relocate}$ 시간의 일부분이 되며 부모 피어가 바뀐 경우에는 지터를 직접적으로 결정하는 시간이다.
- T_{proxyq} : 피어가 프록시 서버에게 부모 피어의 탐색을 요청하였을 경우, 서비스를 받기 위해 프록시 서버의 큐에서 대기하는데 걸리는 시간. 이 시간은 피어들의 도착과 부모 피어의 이탈로 인한 자식 피어들의 재접속에 의해 발생하는 시간으로서 도착율과 피어의 이탈율에 의해 좌우된다.
- T_{search} : 프록시 서버가 하나의 피어에 대해 부모 피어를 결정하는데 걸리는 탐색시간. 이 시간은 프록시 서버내의 피어의 연결 정보를 가진 트리를 탐색하면서 적당한 부모 피어를 선택하는 시간이다. 이 시간은 부모 피어가 바뀌는 피어에 대해서는 지터를 발생 하는 시간으로서 가능하면 짧게 이루어지도록 설계되어야 한다.
- T_{buffer} : 피어가 릴레이 버퍼에서 읽고 쓰는 시간.
- T_{decode} : 하나의 프레임을 디코딩하는데 소요되는 시간.
- T_{rtsp} : 피어가 RTSP 요청 또는 응답 하는데 소요하는 시간. 이 시간은 네트워크 전송 지연 시간을 포함한다.
- $T_{proxyconnection}$: 피어가 프록시 서버에게 요청 또는 응답하는 데 소요하는 시간. 이 시간은 네트워크 전달 지연 시간을 포함한다.

- T_{rtp} : 피어의 버퍼에 있는 패킷이 RTP 프로토콜 스택을 통과하여 자식 피어의 RTP 프로토콜 스택을 통과하는 데 걸리는 시간. 이 시간은 RTP 스택 통과 시간×2 + 네트워크 전달 지연시간으로 정의된다.
- T_{send} : 피어가 하나의 패킷을 하나의 자식 피어에게 전송하는데 걸리는 시간.
- $T_{totalsend}$: 피어가 하나의 패킷을 자신의 모든 자식 피어들에게 전송하는데 걸리는 총 시간. 이 시간은 자식 피어의 개수에 비례하는 시간으로서 이 시간은 피어에 도착하는 패킷의 시간 간격보다 항상 작도록 설계되어야 한다. 만일 크게 되면 피어의 서비스율이 도착율 보다 작아지게 되어 피어의 버퍼는 도착하는 패킷에 의해 이전의 패킷을 잃게 되어 모든 후손 피어들에게 지터를 발생시키는 원인이 된다.
- T_{exit} : 임의의 피어의 사용자가 작업을 종료하고자 한 시점에서 실제 완전한 종료가 이루어지는데 걸리는 시간.
- $T_{waitexit}$: 임의의 피어가 자신의 작업을 종료하고자 한 시점에서 시스템으로부터 완전히 분리되는 동안 기다릴 수 있다고 정의한 최대 시간.
- $T_{playgap(p)}$: 피어 p와 인코딩 서버와의 재생 시간에 대한 시간 차이. 즉 동일한 타임스탬프를 가진 프레임이 인코딩 서버에서 재생될 때와 피어 p에 의해 재생될 때의 시간 차이. 이 시간 차이는 인코딩 서버에서부터 피어 노드 p에 전송되는 과정동안 중간 피어들의 버퍼링 및 전달 지연 등에 의해 누적된 시간의 합에 의해 발생한다.
- $T_{relocate}$: 임의의 피어가 자신의 부모 피어의 이탈에 의해 프록시 서버로부터 새로운 부모 피어가 할당된 후, 새로운 부모 피어로 접속을 시작한 한때부터 첫 번째 프레임을 성공적으로 받을 때까지의 시간.

(2) 스트리밍 및 릴레이

그림 3.7은 피어들이 정상적으로 스트리밍 서비스가 이루어지는 시간 모델을 보여주고 있다.

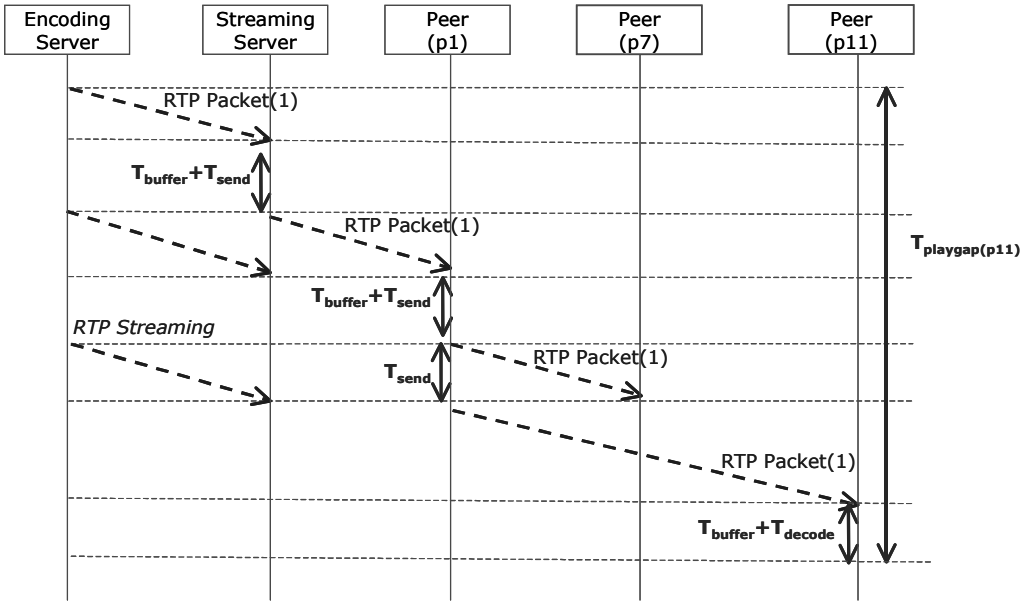

그림 3.7 스트리밍 및 릴레이

인코딩 서버는 오디오 또는 비디오 데이터를 압축하여 일정한 시간 간격으로 RTP 패킷을 생성하여 스트리밍 서버로 전달한다. 스트리밍 서버는 RTP 패킷을 수신하여 버퍼링 한 후, 연결된 자식 피어 1(p1)에게 이를 전달한다. 마찬가지로 피어 1(p1)은 RTP 패킷을 수신하여 버퍼링 한 후, 연결된 자식 피어 7(p7)과 피어 11(p11)에게 각각 릴레이 해주게 된다.

여기서 피어 11(p11)의 $T_{playgap(p11)}$은 인코딩 서버로부터 피어 11까지 RTP 데이터가 전송되는 동안 중간 피어들의 프로세싱 시간(버퍼링, 릴레이)과 네트워크 전달 지연 시간 등의 의해 누적된 시간의 합에 의해 발생함을 알 수 있다.

(3) 피어 참여

그림 3.8은 피어 참여에 대한 시간 모델을 보여주고 있다.

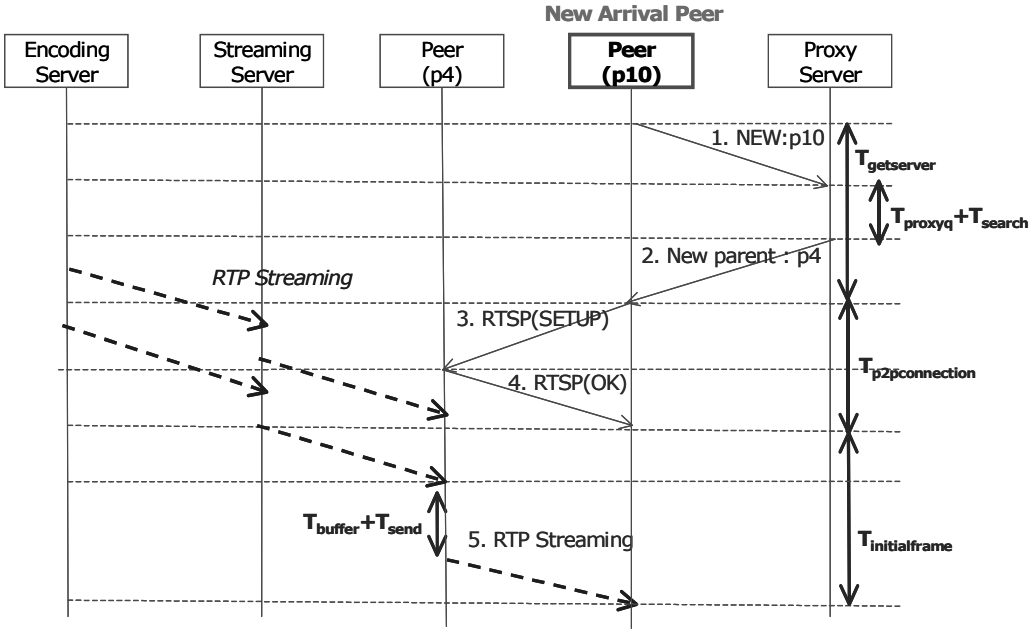

그림 3.8 피어 참여

새로운 피어 10(p10)은 프록시 서버로 접속하여 부모 피어를 할당받고 부모 피어로 접속하여 RTP 스트리밍 서비스를 시작한다. 피어 10에 대한 $T_{connection(p10)}$ 은 다음과 같다.

$$T_{connection(p10)} = T_{getserver(p10)} + T_{p2pconnection(p10)} + T_{initialframe(p10)}$$

$T_{connection}$은 연결 지연 시간으로 피어의 사용자에게 원활한 스트리밍 서비스를 제공할 수 있는지를 알 수 있는 요소로 고려되어진다.

(4) 피어 이탈 및 재배치

그림 3.9는 피어 이탈에 대한 시간 모델을 보여주고 있다.

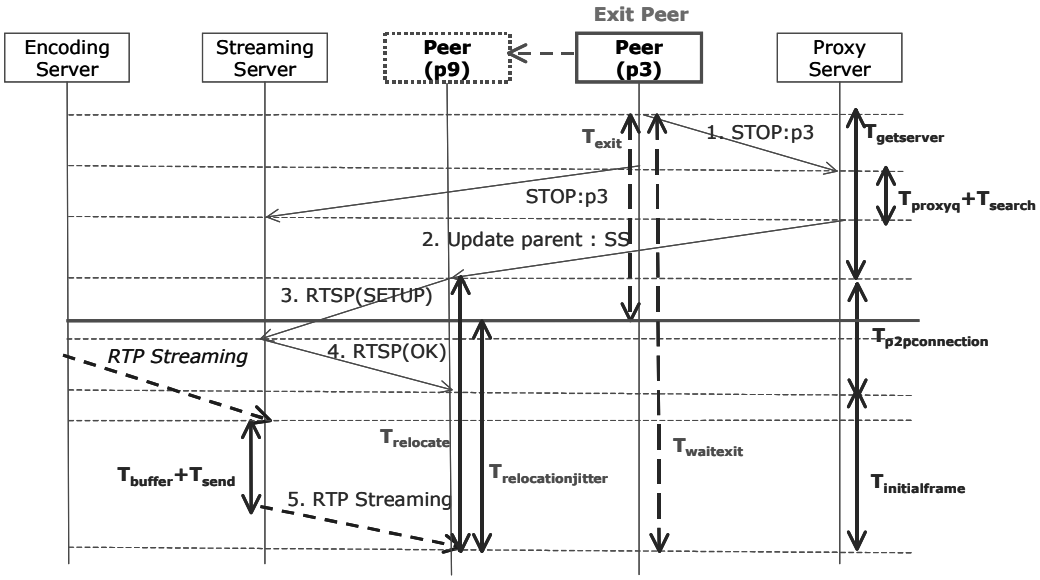

그림 3.9 피어 이탈 및 재배치

예를 들어 피어 3(p3)이 P2P 네트워크에서 이탈 할 경우 자식 피어 9(p9)는 피어 3의 이탈로 인해 스트림이 일시적으로 중단된다. 피어 9는 프록시 서버로부터 새로운 부모를 할당받고 다시 재접속하여 스트리밍 서비스를 시작한다. 피어 이탈과 재배치에 있어서 패킷 손실로 인한 지터가 발생할 소지가 다분하다. 여기서 새로운 시간 변수인 $T_{relocationjitter}$을 정의한다. $T_{relocationjitter}$은 부모 피어의 종료로 인해 더 이상 프레임을 전달받지 못해 생기는 시간 간격이며 그림 3.9에 표현하였다. 따라서 피어 9의 $T_{relocationjitter}$을 0으로 만들기 위해서는 피어 3의 T_{exit}가 $T_{waitexit}$ 이상이면 된다. 여기서 $T_{waitexit}$은 시스템에 설정하는 외부 상수인자로 설정할 수 있으며 사용자 혹은 피어 프로그램은 이 시간만큼 최소 기다리면 된다.

$T_{waitexit}$와 지터 발생의 상관관계를 분석하여 $T_{waitexit}$ 이용하면 지터를 제거 할 수 있다.

다. 성능 모델

본 논문에서는 P2P 스트리밍 시스템에 대한 성능 평가를 하기 위한 성능 지수를 다음과 같이 정의한다.

(1) 연결 지연 시간

연결 지연 시간은 피어가 시스템 내에 도착하는 시점부터 첫 번째 프레임을 받는 때까지의 걸리는 시간으로, 즉 사용자가 스트리밍 서비스를 받기 위해 기다리는 초기 연결 지연 시간을 의미한다. 이 시간은 결론적으로 시간 요소에서 정의한 $T_{connection}$과 동일한 값이다. 이는 참여할 P2P 네트워크에 어느 정도의 과부하가 발생하는 가를 추측하여, 사용자에게 서비스 할 여부를 결정할 수 있는 성능 지수로 작용할 수 있다. 예를 들어 네트워크에 참여하는 사용자가 증가하더라도 연결 지연시간이 일정하다는 것은 스트리밍 서비스를 안정적으로 받을 수 있다는 것을 의미한다.

(2) 재생 시간 갭

재생 시간 갭은 동일한 프레임이 인코딩 서버에서 재생 될 때와 임의의 피어에서 재생되는 시간 차이의 값으로 이 시간은 $T_{playgap}$과 동일하다. 이 시간은 컨텐츠가 인코딩 서버로부터 피어까지 전송되는 동안 중간 피어들의 처리 시간(버퍼링, 릴레이)과 네트워크 전송 지연 시간으로 인해 발생하게 된다. 예를 들어 실시간으로 스트리밍 서비스를 할 경우, 사용자가 증가할수록 재생 시간 갭이 커져서 서비스의 질이 떨어질 수 있다. 이는 P2P 기반 스트리밍 서비스를 할 경우, 시스템의 서비스 질을 결정하는 요소로 고려되어진다.

(3) 지터율

지터율은 총 스트리밍 시간동안 피어가 수신한 총 프레임 수 중 패킷 손실로 발생한 지터의 비율로서, $T_{relocation jitter}$동안 주로 발생한다. 이는 P2P 시스템에서 피어의 자유로운 이탈로 인하여, 스트림이 일시적으로 중단되어 패킷 손실이 발생한다. 이로 인해 사용자는 비디오 또는 오디오의 질이 떨어지게 된다. 따라서 지터는 컨텐츠의 품질을 결정하는 중요한 요인이 된다.

제 4 장 P2PStreamSim : P2P 스트리밍 시스템 시뮬레이터

4.1 시뮬레이터의 요구사항

P2P 스트리밍 시스템을 정확히 시뮬레이션하기 위해 시뮬레이터의 설계 시 요구되는 요구 사항은 다음과 같다.

- 구조적 요구 사항 : 구조적으로 구현되어야 하는 사항
  - 인코딩 서버, 스트리밍 서버, 피어, 프록시 서버
  - RTSP, RTP, 릴레이(Relay) 에이전트
  - 피어 도착, 이탈, 네트워크 토폴로지 재구성
- 성능 평가에 요구되는 사항 : 평가되어야 하는 성능 요소
  - 연결 지연 시간, 재생 시간 갭, 지터율

4.2 시뮬레이터 구조

본 논문에서 개발한 시뮬레이터 ***P2PStreamSim***의 구조는 그림 4.1과 같다. 시뮬레이션의 실행은 P2PSim.tcl 코드에서 시작되며, P2PSim.tcl은 시뮬레이션 파라미터를 설정하는 Config.tcl 데이터 파일을 입력으로 받는다. NS2에서는 크게 OTcl과 C++ 영역으로 나누어지며, P2P 스트리밍 시뮬레이터 객체들은 NS2 시뮬레이터 라이브러리를 이용하여 C++로 구성하였다. 시뮬레이션 결과는 out.nam과 P2PTrace.txt 파일로 저장된다. out.nam은 NS2에서 지원하는 모든 노드에 대한 트레이스를 저장한 결과이며, P2PTrace.txt는 본 논문에서 P2P 스트리밍 시스템의 성능 평가를 위해 따로 결과를 저장한 파일이다.

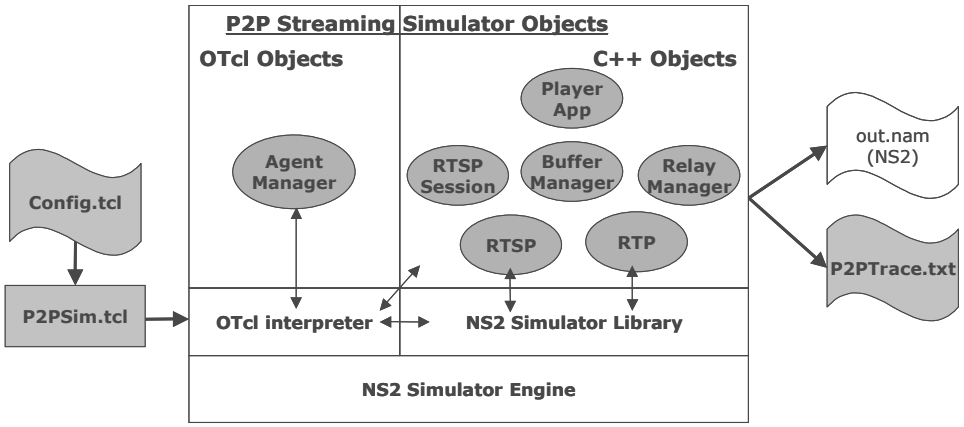

그림 4.1 P2PStreamSim 구조

현재 NS2에서는 P2P와 같은 동적인 네트워크 토폴로지를 구성하여 시뮬레이션 하기에는 다소 부적합한 구조로 되어있다. 따라서 동적인 피어의 도착과 이탈을 시뮬레이션 하기 위해 노드 간 링크는 메쉬 구조로 구성하고, 동적인 연결을 위해 RTSP와 RTP 에이전트를 런타임 중에 생성하여 연결하도록 구현하였다. 그림 4.1에서 AgentManager는 동적으로 RTSP와 RTP를 생성하여 노드에 attach 하거나 detach 하기 위한 OTcl 클래스로, 동적인 피어의 참여와 이탈을 시뮬레이션 하기 위해 필요하다.

Config.tcl과 P2PSim.tcl 코드의 내용은 그림 4.2과 4.3에 각각 나타내었다.

```
# Config.tcl
# Time unit        = Second
# Bandwidth unit = Mbps
# Rate unit        = Mbps
# Frame Rate unit= Fps

set cfg [new Agent/ConfigFile]
$cfg setConfigFile

# P2P Streaming System Model
$cfg set PeerBufferCount            2
$cfg set BufferWriteTime            0.000001
$cfg set BufferReadTime             0.000001
$cfg set DecodeTime                 0.001
$cfg set ProxyProtocolStackTime     0.001
$cfg set PeerSearchTime             0.001

# Experiment Workload
# Network
set Bandwidth                       100Mbs
set TransmissionDelay               10ms

# Encoding Server
set SimulationTime                  500
$cfg set SimulationTime             $SimulationTime
set ES_encodingType                 CBR
set ES_dataType                     Video
$cfg set ES_BitRate                 71Kbps
$cfg set ES_FrameRate               30
$cfg set VariationRate              100
$cfg set Duration                   5

# Peer
set ArrivalPeerCount                100
$cfg set ArrivalPeerCount           $arrivalPeerCount
set ArrivalIntervalDistribution     Exponential
$cfg set MeanArrivalIntervalTime    1.0
$cfg set ExitPeerPercentage         30
set ExitlIntervalDistribution       Exponential
$cfg set MeanExitIntervalTime       1.0
$cfg set ExitStartTime              200.0
$cfg set PeerCapability             5
$cfg set WaitExit                   0
```

그림 4.2 Config.tcl 스크립트 코드

```
source p2p/config.tcl
source p2p/AgentManager.tcl

# Create a simulator object
set ns [new Simulator]
.....
.....
# Node : n0(Encoding Server), n1(Streaming Server)
#        n2(Proxy Server)
# Create nodes
for {set i 0} {$i <= $ArrivalPeerCount+2} {incr i} {
        set n($i) [$ns node]

}
# Create network link
$ns duplex-link $n(0) $n(1) $Bandwidth $TransmissionDelay DropTail
for {set j 1} {$j < $ArrivalPeerCount+2} {incr j} {
        for {set k 2} {$k <= $ArrivalPeerCountr+2} {incr k} {
                if {$j != $k} {
                        $ns duplex-link $n($j) $n($k) $Bandwidth
$TransmissionDelay DropTail
                }
        }
}
.....
.....
# Simulation start
# SimManager : 피어 생성, 이탈 이벤트를 등록하는 C++ 클래스
$ns at 0.0 "$simManager start"
$ns run
```

그림 4.3 P2PSim.tcl 스크립트 코드

그림 4.4와 4.5는 시뮬레이션 결과를 저장한 out.nam과 본 논문에서 따로 저장한 P2PTrace.txt 파일의 내용이다.

```
.....
.....
- 279.31003 1 56 udp 295 ------- 0 1.11 56.9 8380 601744
r 279.320049 1 48 udp 295 ------- 0 1.3 48.11 8380 601740
r 279.32005 1 50 udp 295 ------- 0 1.5 50.11 8380 601741
r 279.320051 1 52 udp 295 ------- 0 1.7 52.11 8380 601742
+ 279.320051 48 73 udp 295 ------- 0 48.12 73.7 8380 601745
- 279.320051 48 73 udp 295 ------- 0 48.12 73.7 8380 601745
r 279.320052 1 54 udp 295 ------- 0 1.9 54.11 8380 601743
+ 279.320052 50 75 udp 295 ------- 0 50.12 75.7 8380 601746
- 279.320052 50 75 udp 295 ------- 0 50.12 75.7 8380 601746
.....
.....
```

그림 4.4 out.nam

Node	PNode	Start	Stop	Frame	Jitter	JitterRate	ConDelay	PlayingGap
.....								
.....								
44	21,54,	49.0971	0	13521	5	0.0369795	0.0763117	0.0401074
45	20,52,	51.2861	0	13455	5	0.0371609	0.0873347	0.0401064
46	19,50,	54.1265	0	13370	5	0.0373972	0.0802721	0.0401054
47	18,48,	58.6539	0	13235	4	0.0302229	0.0861674	0.0401044
48	8,1,	59.1731	0	13222	2	0.0151263	0.0670174	0.0401024
49	17,56,	60.2647	0	13186	5	0.037919	0.0753744	0.0401074
50	9,1,	62.3595	0	13125	3	0.0228571	0.0806404	0.0401034
51	16,54,	62.762	0	13110	6	0.0457666	0.0781094	0.0401064
52	10,1,	64.1207	0	13071	4	0.0306021	0.0860711	0.0401044
53	15,52,	64.4793	0	13059	6	0.0459453	0.0608334	0.0401054
54	11,1,	65.0866	0	13042	4	0.0306701	0.0868697	0.0401054
.....								
.....								

그림 4.5 P2PTrace.txt

본 논문에서 인코딩 서버, 스트리밍 서버, 피어, 프록시 서버는 NS2에서 정의한 노드로 표현한다. 또한 에이전트는 RTSP, RTP와 같은 전송 계층 프로토콜뿐만 아니라 P2P 스트리밍 서비스를 하기 위한 모듈을 에이전트로 구성한다. 그림 4.6은 본 연구에서 개발한 P2P 스트리밍 시스템 시뮬레이터 *P2PStreamSim*의 각 노드 구조를 보여주고 있다.

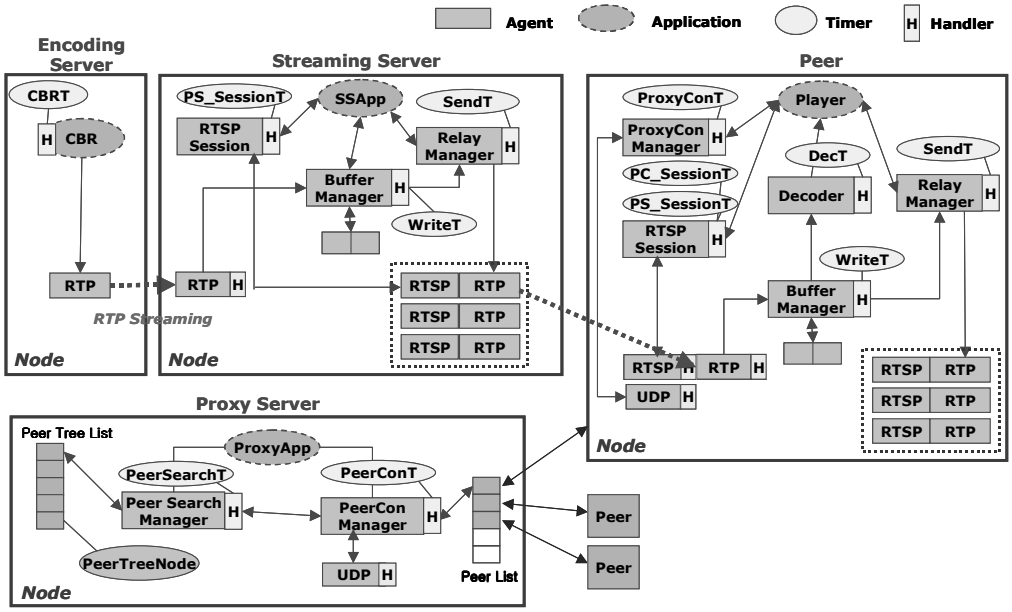

그림 4.6 P2PStreamSim 노드 구조

4.3 P2PStreamSim 구현

4.3.1 구현 환경

*P2PStreamSim*은 Fedora 5의 시스템에서 NS 버전 2.30을 기반으로 OTcl 스크립트와 C++ 언어를 사용하여 개발하였다. 시뮬레이션 파라미터와 네트워크 토폴로지 구성은 OTcl 스크립트로 프로그래밍 하였고, P2P 스트리밍을 위한 모듈은 C++ 언어로 구현하였다.

4.3.2 구현된 클래스

*P2PStreamSim*을 구현한 클래스 및 소스 파일들은 표 4.1과 같다.

표 4.1 클래스 및 소스 파일들

클래스 이름	정의 및 구현파일	설명
AgentManager	agentManager.tcl	RTSP, RTP를 동적으로 생성하여 연결
ConfigFile	configFile.{h,cc}	시뮬레이션 파라미터를 설정하는 config.tcl 과 연동
SimManager	simManager.{h,cc}	피어의 도착과 이탈 이벤트를 생성
CBRApp	cbrApp.{h,cc}	CBR 형태의 데이터 생성
VBRApp	vbrApp.{h,cc}	VBR 형태의 데이터 생성
SSApp	ssApp.{h,cc}	스트리밍 서버의 어플리케이션
PlayerApp	playerApp.{h,cc}	피어의 어플리케이션
ProxyApp	proxyApp.{h,cc}	프록시 서버의 어플리케이션
RTSPSession	rtspSession.{h,cc}	RTSP 요청/응답 메시지 생성
BufferManager	bufferManager.{h,cc}	RTP 패킷을 버퍼에 저장
RelayManager	relayManager.{h,cc}	다른 피어에게 RTP 패킷을 릴레이
Decoder	decoder.{h,cc}	한 프레임 단위로 디코딩하여 재생
ProxyConManager	proxyConManager.{h,cc}	피어가 프록시 서버와 통신하기 위한 요청 메시지 생성
PeerConManager	peerConManager.{h,cc}	프록시 서버가 피어와 통신하기 위한 응답 메시지 생성
PeerSearchManager	peerSearchManager.{h,cc}	부모 피어 탐색
PeerTreeNode	peerTreeNode.{h,cc}	피어에 대한 트리 노드
P2PTrace	p2pTrace.{h,cc}	트레이스 파일
RTSP	rtsp.{h,cc}	RTSP 프로토콜
RTP	rtp.{h,cc}	RTP 프로토콜

4.3.3 RTSP/RTP 구현

본 논문에서는 P2P 스트리밍 서비스를 하기 위해 RTSP와 RTP 프로토콜 에이전트를 구현하였다. RTSP는 실시간으로 미디어를 송수신하기 위한 프로토콜로서, 멀티미디어 컨텐츠 패킷 포맷을 지정하기 위해 RTP를 사용한다. 현재 NS2에서 지원하는 RTP 헤더 정보에는 패킷의 시퀀스 번호만 존재하며, 표준 RTP 헤더 정보를 모두 포함하지 않고 있다. 따라서 P2P 스트리밍 시스템을 시뮬레이션 하기 위해 RTP 헤더를 새롭게 만들어 NS2에 추가하였다.

그림 4.7은 본 논문에서 정의한 RTP 헤더 포맷이다.

```
struct hdr_p2prtp {
        int seqNumber;              /* 시퀀스 번호 */
        int nbytes;                 /* 패킷 바이트 수 */
        double timeStamp;           /* RTP 데이터가 샘플링 된 시점 */
        int marker;                 /* 음성 또는 화상에 대한 정보를 구별 */
        static int offset_;         /* 헤더 접근하기 위한 offset */
        inline static int& offset() { return offset_; }
        inline static hdr_p2prtp* access(const Packet *p)
        { return (hdr_p2prtp*) p->access(offset_); }
}
```

그림 4.7 RTP 헤더 포맷

RTSP와 RTP 프로토콜 에이전트는 NS2에서 지원하는 UDP 프로토콜 에이전트를 상속받아 구현하였으며, send_msg() 멤버 함수를 통해 패킷을 전송하고 recv() 멤버 함수로 패킷을 수신한다.

4.3.4 인코딩 서버 구현

인코딩 서버는 비디오 또는 오디오 스트림을 CBR(Constant Bit Rate) 또는 VBR(Variable Bit Rate) 형태로 압축하여 이를 스트리밍 서버에게 전송하는 서버이다. 그림 4.8은 본 논문에서 구현한 인코딩 서버이다.

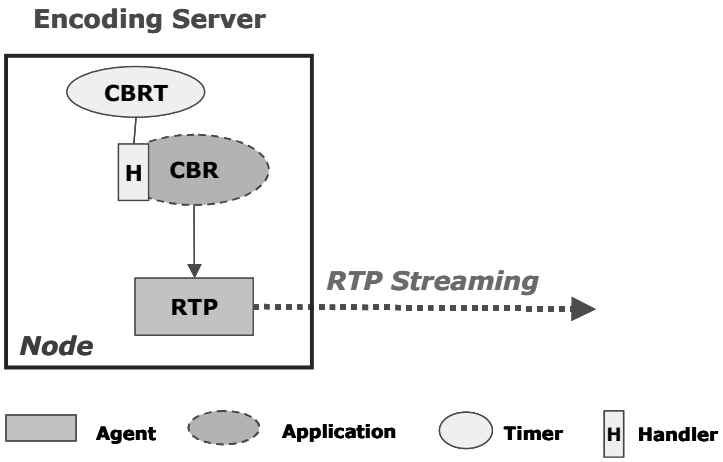

그림 4.8 인코딩 서버 구조

인코딩 서버는 CBR 또는 VBR 형태의 트래픽을 발생시키는 어플리케이션과 RTP 패킷을 전송하는 RTP 에이전트로 구성된다. 예를 들어 CBR 어플리케이션은 CBRTimer 가지고 있어 일정한 간격으로 RTP 데이터를 생성한다. RTP 에이전트는 어플리케이션이 생성한 RTP 데이터를 패킷으로 나누어 전송한다.

표 4.2에서는 외부 Tcl 스크립트 코드에서 호출 가능한 CBR 어플리케이션의 인터페이스를 보여주며, 알고리즘은 그림 4.9와 같다.

표 4.2 CBR 어플리케이션 인터페이스

인터페이스	기능
start()	CBR 형태의 RTP 데이터를 생성하여 RTP 에이전트로 전송을 시작한다.
stop()	RTP 데이터의 전송을 중지한다.

```
CBRTimer cbrTimer; /* RTP 데이터를 생성하여 전송하기 위한 타이머 */

/* CBRT : CBR Timer */
void CBRTimer::expire(Event *e)
{
        CBRApp의 sendData() 호출한다.;
}

/* CBR Application */
void CBRApp::sendData()
{
        RTP 데이터를 생성한다.;
        RTP 헤더 정보(시퀀스 번호, 타임스탬프 등)를 설정한다.;
        RTP 에이전트로 데이터를 전송한다.;
        /* interval 초후에 타이머는 만료(expire) 됨 */
        cbrTimer.resched(interval);
}
```

그림 4.9 CBR 어플리케이션 알고리즘

4.3.5 스트리밍 서버 구현

스트리밍 서버는 스트림을 분배하는 부모 피어로서, 인코딩 서버로부터 수신한 RTP 데이터를 연결된 자식 피어에게 전달해준다. 그림 4.10과 같이 스트리밍 서버는 RTSP Session, Buffer Manager, Relay Manager, RTSP, RTP 에이전트로 구성되며, 각 에이전트는 실행할 시간의 흐름을 나타내기 위한 타이머를 가지고 있다. 새로운 피어가 세션을 요청할 경우

RTSP 세션을 설정하기 위해 RTSP Session은 응답 메시지를 생성하여 RTSP로 보낸다. 세션 설정 후, Buffer Manager는 RTP로 부터 수신한 RTP 데이터를 버퍼에 저장한다. Relay Manager는 버퍼에서 한 패킷씩 읽어 RTP를 통해 연결된 자식 피어에게 데이터를 전송한다.

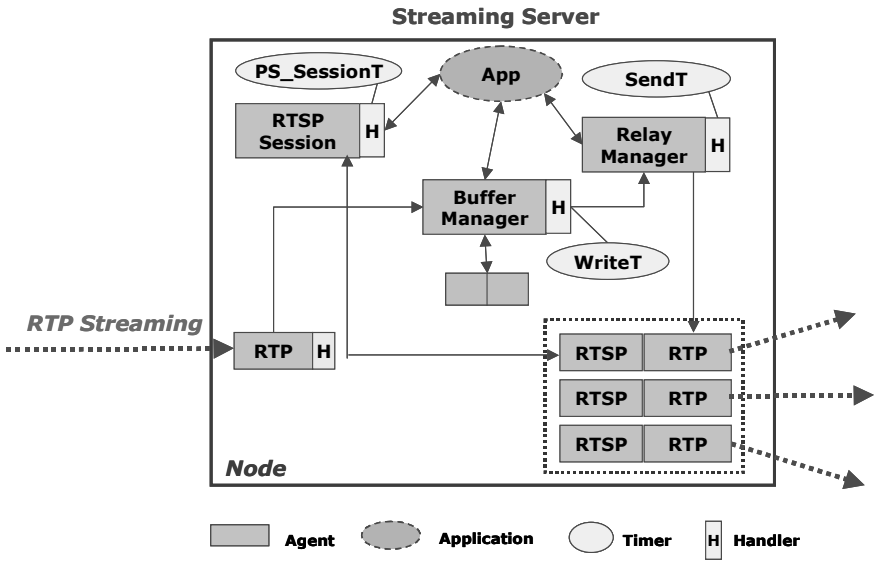

그림 4.10 스트리밍 서버 구조

• RTSP Session : RTSP 세션을 설정하기 위해 요청 또는 응답 메시지를 생성하는 에이전트이다. RTSP Session은 피어의 RTSP 요청에 대한 응답 소요시간을 나타나는 PS_SessionTimer가 존재한다. 본 논문에서는 RTSP 세션설정을 위한 메시지를 표 4.3과 같이 정의하였다.

표 4.3 RTSP 메시지 종류

메시지 종류	설명
SETUP	피어가 부모 피어로 RTSP 세션을 요청 하는 메시지
OK	RTSP 요청을 수락하는 응답 메시지
STOP	피어가 자신의 종료 상황을 알리는 메시지

• Buffer Manager : 수신한 패킷을 버퍼에 저장하기 위한 에이전트로서 알고리즘은 그림 4.11과 같다.

```
/* 총 버퍼의 수는 2개로 가정 한다 */
rear=0;              /* 데이터를 저장할 버퍼 번호 */
bufferCount=2;       /* 총 버퍼의 수 */
/* WriteT : 버퍼에 쓰는 시간을 소요하기 위한 타이머 */
WriteTimer writeTimer;

Algorithm Buffer_Manager
input: RTP 패킷
output: none
{
        수신한 RTP 패킷을 버퍼에 저장한다.;
        /* 데이터를 저장할 버퍼 번호를 다음 버퍼 번호로 설정 */
        rear= (rear+1) % bufferCount;
        /* bufferWriteTime 이후에 타이머는 만료됨 */
        writeTimer.resched(bufferWriteTime);
}
```

그림 4.11 Buffer Manager 알고리즘

• Relay Manager : 버퍼에 저장된 패킷을 하나씩 읽어 연결된 자식 피어에게 전송해주는 에이전트이다. Relay Manager는 버퍼에서 한 패킷을 읽어 하나의 피어에게 전송하는 시간을 소요하는 SendTimer를 가지고 있다. SendTimer는 연결된 자식 피어의 수만큼 돌며 패킷을 전송하게 된다.

4.3.6 피어 구현

피어는 P2P 네트워크에 참여하는 사용자 단말기로 수신한 RTP 데이터를 재생하거나 연결된 자식 피어에게 전달해주며 그림 4.12와 같다.

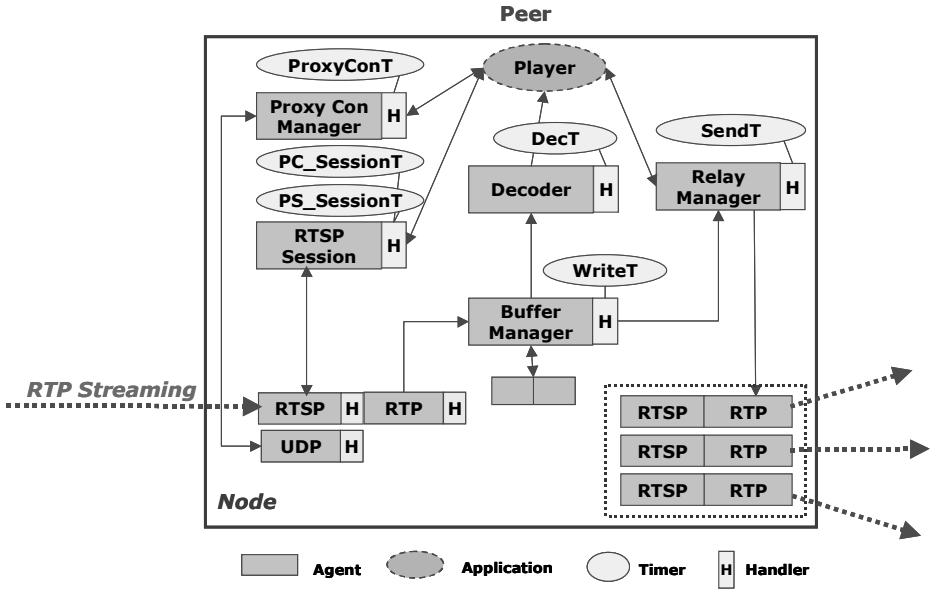

그림 4.12 피어 구조

피어는 Proxy Connection Manager, RTSP Session, Buffer Manager, Decoder, Relay Manager, RTSP, RTP, UDP 에이전트로 구성되며, 각 에이전트는 시간의 흐름을 나타내기 위한 타이머를 가지고 있다. 여기서 UDP는 NS2에서 지원하는 UDP 프로토콜을 이용하였다. 피어가 네트워크에 참여할 시 Proxy Connection Manager는 요청 메시지를 생성하여 UDP를 통해 프록시 서버에게 새로운 부모 피어를 요청한다. 피어는 프록시 서버로부터 할당받은 부모 피어로 RTSP Session을 이용하여 요청 메시지를 생성하고 RTSP를 통해 부모 피어로 전송한다. 부모 피어와 세션 설정 후, Buffer Manager는 RTP로부터 수신한 데이터를 버퍼에 저장한다. Decoder는 버퍼에서 한 프레임씩 읽어 디코딩 후 재생한다. 또한 연결된 자식 피어가 있을 경우 Relay Manager는 버퍼에서 한 패킷씩 읽어 RTP를 통해 연결된 피어에게 데이터를 전송한다.

피어의 구조는 스트리밍 서버와 유사하며 Proxy Connection Manager와 Decoder가 추가된 구조이다.

• Proxy Connection Manager : 피어가 네트워크에 참여 또는 이탈시, 프록시 서버에게 자신의 상황을 알리기 위한 에이전트로 요청 메시지를 생성하는 에이전트이다. 본 논문에서는 프록시 서버와 통신하기 위한 메시지를 표 4.4와 같이 정의하였다.

• Decoder : 수신한 패킷을 한 프레임 단위로 디코딩하여 재생하는 에이전트로 DecoderTimer를 사용하여 디코딩 시간을 소요한다.

표 4.4 프록시 서버와 통신하기 위한 메시지 종류

메시지 종류	설명
NEW	피어가 새로운 부모 피어를 요청하는 메시지로 피어는 자신의 ID(피어의 주소와 동일)를 프록시 서버에게 알린다.
STOP	피어가 자신의 종료상황을 알리는 메시지
OK	부모 피어 요청에 대한 프록시 서버의 응답 메시지로 결정된 부모 피어의 ID를 알려준다.

4.3.7 프록시 서버 구현

프록시 서버는 P2P 네트워크에 존재하는 피어에 대한 정보를 관리하는 서버로서, 피어가 네트워크에 참여하거나 이탈 시 부모 피어를 탐색하여 피어에게 알려주며 동적으로 네트워크 토폴로지를 재구성한다.

그림 4.13과 같이 프록시 서버는 Peer Connection Manager와 Peer Search Manager, UDP 에이전트로 구성된다. 여기서 UDP는 NS2에서 지원하는 UDP 프로토콜을 이용하였다. 피어가 부모 피어의 탐색을 요청하였을 경우 피어는 프록시 서버의 Peer List 큐에서 대기하게 된다. Peer Search Manager는 Peer Tree List에서 적당한 부모 피어를 탐색한 결과를 Peer Connection Manager에게 알려주며, Peer Connection Manager는 응답 메시지를 생성하여 큐에 대기하고 있는 피어에게 알려준다.

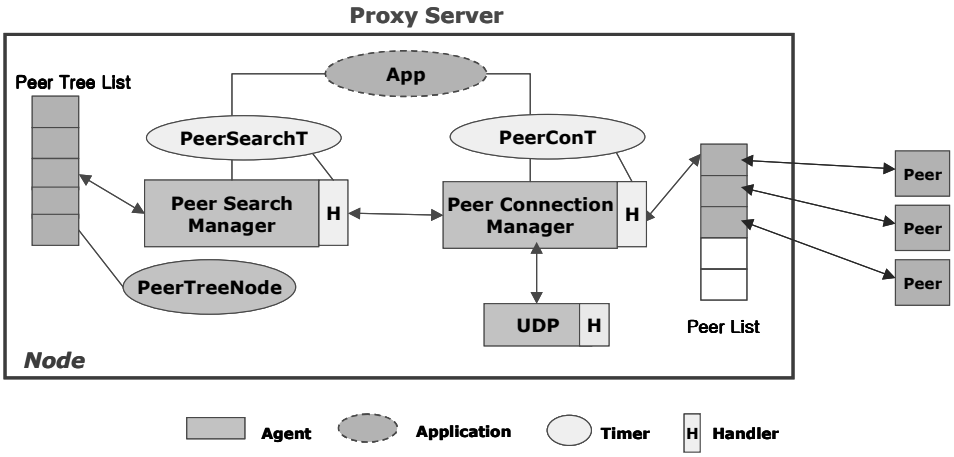

그림 4.13 프록시 서버 구조

• Peer Connection Manager : 피어의 참여 또는 이탈에 대한 요청 메시지를 처리하기 위한 에이전트로서, 응답 메시지를 생성하는 시간을 소요하기 위한 PeerConnectionTimer를 가지고 있다.

• Peer Search Manager : 피어의 부모 피어를 탐색하는 에이전트로서 탐색 시간을 소요하는 PeerSearchTimer를 가지고 있다. Peer Search Manager는 현재 네트워크에 참여한 피어에 대한 정보(PeerTreeNode)를 PeerTreeList에 트리 구조로 유지한다. PeerTreeNode는 표 4.5와 같은 정보를 유지하며 부모 피어 탐색 알고리즘은 그림 4.14와 같다.

표 4.5 PeerTreeNode의 멤버 변수

멤버 변수	설명
PeerID	피어를 식별하기 위한 ID로 피어의 주소와 동일
ParentPeerID	부모 피어 ID
PeerTreeNode *parent	부모 피어에 대한 포인터
vector<PeerTreeNode *> child	자식 피어들에 대한 벡터
childPeerCount	연결된 자식 피어의 수
hopCount	스트리밍 서버를 기준으로 한 피어의 홉 수

```
Algorithm Peer_Search
input: 피어 ID
output: 부모 피어 ID
{
        PeerTreeNode 중 홉 수가 작은 피어를 선택한다.;
        그 중 연결된 자식 피어수가 적은 피어를 부모 피어로 선택한다.;
        PeerTreeList에 요청한 피어를 추가한다.;
        PeerTreeList를 갱신한다.;
}
```

그림 4.14 부모 피어 탐색 알고리즘

제 5 장 시뮬레이션 및 성능 평가

5.1 성능 평가의 목적

본 논문에서 개발한 시뮬레이터 *P2PStreamSim*은 P2P 스트리밍 시스템 모델에 대한 설계의 정확성을 검증하고 다양한 환경에서 성능을 평가하기 위해 개발되었다. 따라서 테스트 P2P 스트리밍 시스템을 사례로 적용하여 시뮬레이터의 동작을 검증하고 성능 평가를 수행한다.

5.2 시뮬레이션 환경

*P2PStreamSim*은 Fedora 5의 시스템에서 NS 버전 2.30을 기반으로 OTcl 스크립트와 C++ 언어를 사용하여 개발되었다. 네트워크 토폴로지는 한 대의 인코딩 서버, 스트리밍 서버, 프록시 서버와 N개의 피어로 구성된다. N개의 피어는 모두 프록시 서버와 양방향 링크로 설정하였으며, N개의 피어는 메쉬 구조의 형태로 양방향 링크로 설정하였다.

5.3 시뮬레이션 파라미터

시뮬레이션 파라미터는 크게 다음 2개의 논리적인 영역으로 구분된다.

- **P2P 스트리밍 시스템 모델(P2P streaming system model)**
- **작업 부하(experiment workload)**

표 5.1은 P2P 스트리밍 시스템 모델에 대한 파라미터로서 시뮬레이션 할 모델에 대한 실제 측정값을 설정한다. 표 5.2는 작업 부하를 설정하기 위한 시뮬레이션 파라미터로서 P2P 스트리밍 시스템 모델에 대한 다양한

실험을 하기 위해 설정하는 값이다.

표 5.1. P2P 스트리밍 시스템 모델

요소	파라미터	설명
피어	bufferCount	버퍼 수
	bufferAccessDelay	버퍼 읽기/쓰기 시간
	RTSPStackTime	RTSP 요청/응답 시간
	decodingTime	한 프레임을 디코딩 하는 시간
프록시 서버	queueingDelay	피어가 서비스를 받기 위해 큐에서 대기하는 시간
	protocolStackTime	프록시 서버 요청/응답 시간
	algorithmTime	부모 피어의 탐색 시간

표 5.2 작업 부하

요소	파라미터	설명
네트워크 환경	bandwidth	네트워크 대역폭
	transmissionDelay	네트워크 전송 지연시간
인코딩 서버	streamingTime	총 스트리밍 시간
	encodingType	인코딩 데이터 타입(CBR/VBR)
	dataType	오디오 / 비디오
	bitRate	비트율
	frameRate	프레임율
	VBR_variationRate	비트율에 대한 VBR의 비율
	VBR_duration	초당 VBR의 연속 수
피어	arrivalPeerCount	참여한 피어의 총 수
	meanArrivalIntervalTime	피어의 평균 도착 시간 간격
	arrivalIntervalDistribution	피어의 도착 분포
	exitPeerPercentage	이탈하는 피어의 총 비율
	meanExitIntervalTime	피어의 평균 이탈 시간 간격
	exitIntervalDistribution	피어의 이탈 분포
	exitStartTime	피어의 이탈 시작 시점
	capability	피어가 가질 수 있는 자식 피어 수
	waitExit	피어가 종료하고자 한 시점에서 기다릴 수 있는 최대 대기 시간

5.4 성능 지수 정의

본 논문에서는 P2P 스트리밍 시스템에 대한 성능 평가를 하기 위한 성능 지수를 다음과 같이 정의하며, 3장의 성능 모델에서 언급한 바 있다.

- **연결 지연 시간($T_{connection}$)**

  연결 지연 시간은 피어가 시스템 내에 도착하는 시점부터 첫 번째 프레임을 받는 때까지의 걸리는 시간이다.

- **재생 시간 갭($T_{playgap}$)**

  재생 시간 갭은 동일한 프레임이 인코딩 서버에서 재생 될 때와 임의의 피어에서 재생되는 시간 차이의 값이다.

- **지터율(R_{jitter})**

  지터율은 총 스트리밍 시간동안 피어가 수신한 총 프레임 수 중 패킷 손실로 발생한 지터의 비율이다.

5.5 성능 평가

본 시뮬레이터의 정확성 및 효용성을 검증하기 위해 본 논문에서 정의한 P2P 스트리밍 시스템 모델을 적용하여 표 5.3과 같이 파라미터 값을 설정하였다.

표 5.3 시뮬레이션 파라미터(P2P 스트리밍 시스템 모델)

요소	파라미터	값
피어	bufferCount	2
	bufferAccessDelay	1us
	RTSPStackTime	1ms
	decodingTime	1ms
프록시 서버	protocolStackTime	1ms
	algorithmTime	1ms

5.5.1 시뮬레이션 결과

실험 결과는 총 20번의 시뮬레이션을 실행한 결과의 평균값이다.

가. 실험 결과 1 : 피어 참여에 따른 실험 결과

피어 ID는 피어를 구별하는 식별자로 피어의 주소와 동일하며, P2P 네트워크에 참여하는 순서대로 ID가 하나씩 증가한다. 총 500초의 스트리밍 시간동안 100개의 피어가 평균 1초 간격으로 도착하며 표 5.4와 같이 시뮬레이션 파라미터를 설정하였다. 여기서 인코딩 서버의 비트율은 실제 Ateme 사의 H.264 / AAC Kompressor 하드웨어 인코더[30]를 기준으로 해상도가 720×480일 때의 측정한 값을 시뮬레이션 파라미터로 설정하였다.

피어 ID에 따른 평균 연결 지연 시간을 측정한 결과는 그림 5.1과 같다.

표 5.4 실험 1 시뮬레이션 파라미터(작업부하)

요소	파라미터	값
네트워크 환경	bandwidth	100Mbps
	transmissionDelay	10ms
인코딩 서버	streamingTime	500s
	encodingType	CBR
	dataType	Video
	bitRate	71Kbps
	frameRate	30fps
피어	arrivalPeerCount	100
	meanArrivalIntervalTime	1.0s
	arrivalIntervalDistribution	Exponential
	capability	5

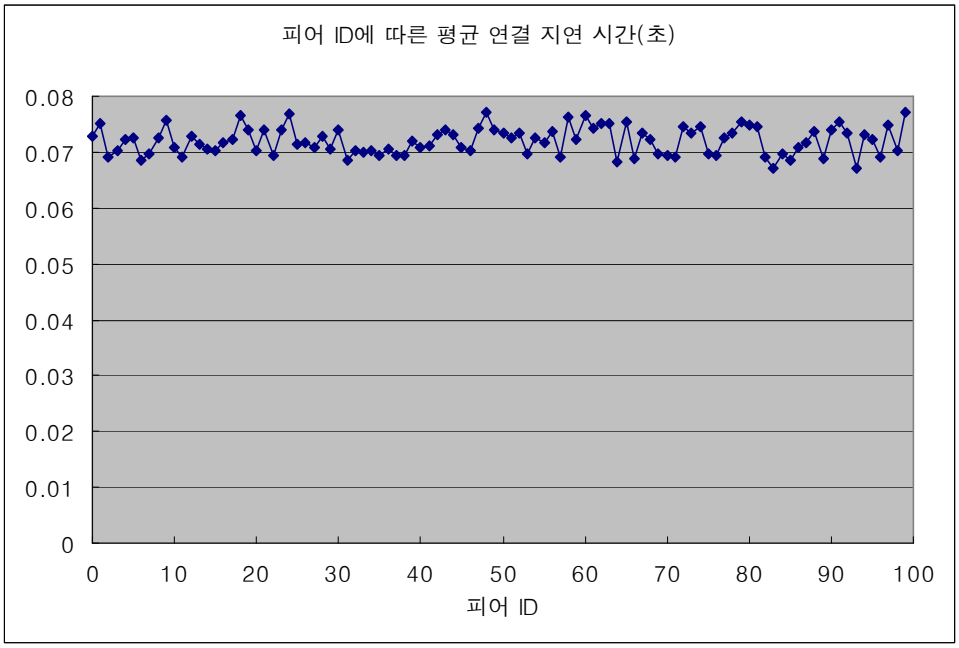

그림 5.1 피어 ID에 따른 평균 연결 지연 시간

실험 결과 총 100개의 피어에 대한 평균 연결 지연시간은 약 70ms 소요됨을 알 수 있다. 인코딩 서버의 프레임율이 30fps이므로 프레임 간 지연시간은 약 30ms 정도이다. 따라서 각 피어들은 첫 번째 프레임을 성공적으로 받을 때까지의 오차 범위가 약 30ms 이내로 나타남을 알 수 있다.

$T_{connection} = T_{getserver} + T_{p2pconnection} + T_{initialframe}$ 이므로 시뮬레이션 파라미터에 따라서 연결 지연 시간은 다양하게 측정될 수 있다.

그림 5.2는 위와 동일한 시뮬레이션 파라미터는 설정하여 홉 수에 따른 평균 재생 시간 갭을 측정한 결과이다. 실험 결과 홉 카운트 수가 증가함에 따라 피어의 재생 시간 갭이 증가함을 알 수 있다. 네트워크에 참여한 피어의 가용 능력, 즉 연결 가능한 자식 피어의 수가 모두 동일하며 피어의 프로세싱 시간(버퍼링, 릴레이)이 빠르므로 재생 시간 갭에 영향을 미치지 않는다. 따라서 홉 수에 따른 평균 재생 시간 갭은 네트워크 전송 지

연 시간에 비례하여 증가함을 알 수 있다.

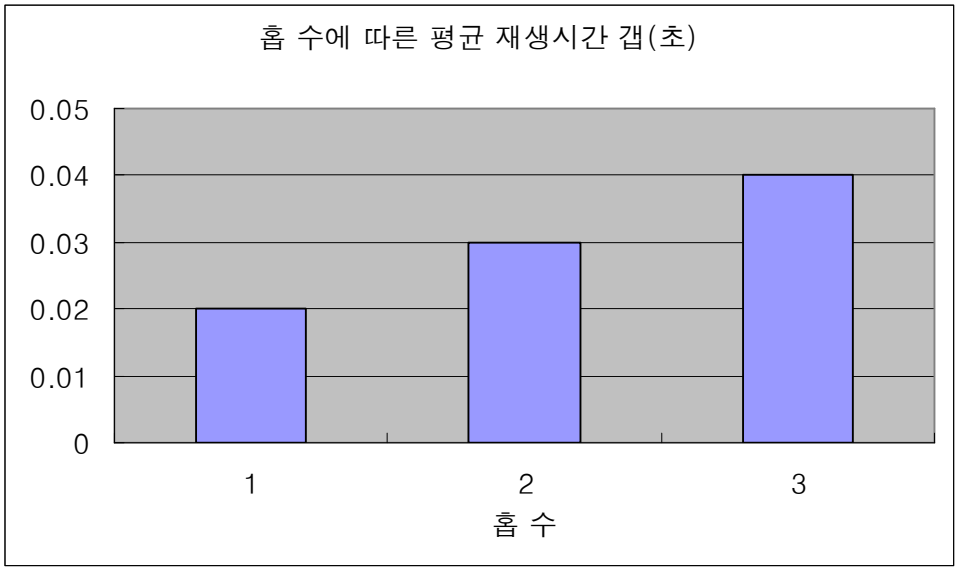

그림 5.2 홉 수에 따른 평균 재생 시간 갭

나. 실험 결과 2 : 피어 이탈에 따른 실험 결과

지터의 발생은 부모 피어의 이탈로 인해 자식 피어들의 스트리밍 서비스가 일시적으로 중단되기 때문이다. 즉, 자식 피어들은 새로운 부모 피어로 접속하여 첫 번째 프레임을 수신하기 전까지 패킷 손실로 인한 지터가 발생하게 된다.

임의의 피어가 스트리밍 서비스를 중단하고 이탈할 경우 이탈하고자 한 시점에서 실제 사용자가 완전한 종료가 이루어질 때까지 기다릴 수 있다고 정의한 최대 대기 시간을 $T_{waitexit}$ 정의하며, 이는 시스템에 설정하는 외부 상수이다. 따라서 이탈하는 피어가 자식 피어들이 다시 재배치되어 스트리밍이 정상적으로 이루어 질 때까지 $T_{waitexit}$을 유지한다면 지터를 줄일 수 있다.

실험은 표 5.5와 같이 총 500초의 스트리밍 시간동안 100개의 피어 중 30%의 피어가 평균 1초 간격으로 이탈할 경우를 시뮬레이션 하였다. 그림 5.3은 $T_{waitexit}$시간이 0ms부터 70ms까지 증가할 경우 피어 ID에 대한 평균 지터율을 측정한 결과이다.

표 5.5 실험 2 시뮬레이션 파라미터(작업부하)

요소	파라미터	값
네트워크 환경	bandwidth	100Mbps
	transmissionDelay	10ms
인코딩 서버	streamingTime	500s
	encodingType	CBR
	dataType	Video
	bitRate	71Kbps
	frameRate	30fps
피어	arrivalPeerCount	100
	meanArrivalIntervalTime	1.0s
	arrivalIntervalDistribution	Exponential
	exitPeerPercentage	30%
	meanExitIntervalTime	1.0s
	exitIntervalDistribution	Exponential
	exitStartTime	200s
	capability	5
	waitExit	0ms, 30ms, 70ms

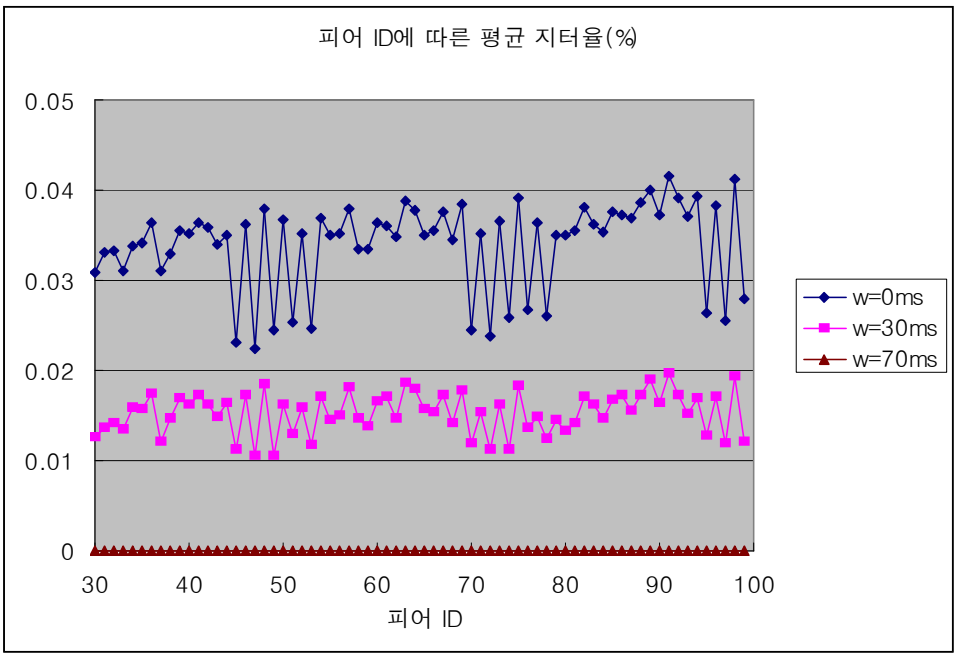

그림 5.3 피어 ID에 따른 평균 지터율

실험 결과 $T_{waitexit}$ 시간이 증가함에 따라 피어에 따른 평균 지터율은 감소하며, $T_{waitexit}$ 시간이 70ms일 때는 지터가 발생하지 않는다. 이것은 피어의 평균 연결 지연 시간이 약 70ms이므로 이 시간만큼 이탈 피어가 $T_{waitexit}$을 유지한다면 자식 피어들은 스트림 끊김 현상 없이 원활한 서비스를 받을 수 있다. 즉, 사용자나 피어에 설치된 프로그램이 종료해야한다고 결정하고 기다리는 시간이 약 70m로서, 사용자 측면에서 아주 빠른 시간이므로 테스트한 P2P 시스템은 매우 효율적이라고 평가되어진다.

제 6 장 결론 및 향후 연구

6.1 결론

P2P 기술은 이미 일상생활의 일부분이 되어버린 인스턴트 메신저, 인터넷 폰, 음악 또는 동영상 파일 공유, 멀티미디어 서비스, 온라인 게임 등 많은 분야에서 널리 사용되고 있다.

한편 기존 인터넷 스트리밍 시스템은 클라이언트-서버 형태의 중앙 구조로 서버에 대한 트래픽 집중으로 인한 병목 현상이 발생하며, 한 스트리밍 서버의 가용 능력에 따라 스트리밍 단말기의 개수가 제한되는 단점이 있다. 이러한 한계점을 극복하기 위해 인터넷 스트리밍 시스템에 P2P 네트워크를 이용하는 구조에 대한 연구들이 전 세계적으로 진행되고 있다.

그러나 P2P 기반 스트리밍 시스템은 설계 및 연구 개발에 있어 많은 컴퓨터들이 필요하며 성능 평가 시 많은 시간과 경비가 소모된다. 따라서 P2P 스트리밍 시스템을 설계 개발하기에 앞서 설계의 정확성을 검증하고 다양한 네트워크 토폴로지를 구성하여 실험하기 위한 시뮬레이터의 개발이 절실히 요구된다.

본 논문에서는 P2P 스트리밍 시스템에 대한 모델과 성능 지수를 정의하고 네트워크 기본 프로토콜과 체계를 제공하는 NS2라는 시뮬레이션 툴을 이용하여 P2P 스트리밍 시뮬레이터 ***P2PStreamSim***을 설계 및 구현하였다. 또한 테스트 P2P 스트리밍 시스템에 대한 성능 평가를 수행하였다.

결론적으로 본 논문 연구는 P2P 스트리밍 시스템을 개발하기 전 단계에서 설계의 정확성을 검증하고 그 성능을 평가할 수 있는 도구를 제공한다는 면에서 큰 의미를 지닌다.

6.2 향후 연구 과제

향후 P2P 스트리밍 시스템에 대한 실제 파라미터 값들을 적용한 실험을 수행하며 다양한 P2P 스트리밍 시스템 모델에 대한 성능 평가를 수행할 수 있도록 *P2PStreamSim*을 확장 보완하고자 한다.

향후 연구 과제의 세부적인 항목은 다음과 같다.

- **다중 부모 피어 실험**

부모 피어의 이탈로 인한 자식 피어들의 스트림 끊김 현상을 최소화하기 위한 방법으로 부모 피어를 두개 이상의 다중 피어로 유지하는 방법이다. 이는 피어가 네트워크 참여 시 프록시 서버로부터 N개의 부모 피어를 할당 받아, 부모 피어의 이탈 시 나머지 부모 피어로 바로 접속하여 스트림을 수신하도록 하는 방법이다. 즉, 피어는 프록시 서버로 다시 접속하여 부모 피어를 할당 받을 때까지의 지연 시간을 줄일 수 있으므로 스트림 끊김 현상을 줄일 수 있다.

- **네트워크 트래픽에 따른 실험**

본 논문에서는 네트워크의 각 링크의 현실적인 시뮬레이션은 무시하였으며, 향후 네트워크 대역폭과 전송 지연 시간을 다양하게 변화시켜 성능 평가를 수행한다.

- **피어의 가용 능력에 따른 실험**

피어의 가용 능력, 즉 피어가 가질 수 있는 자식 피어의 수를 다르게 하여 다양한 실험을 수행한다.

• **프록시 서버의 알고리즘에 따른 실험**

본 논문에서는 부모 피어 선택 탐색 알고리즘을 최소 홉 수와 최소 자식 피어수를 고려한 알고리즘을 선택하였으며, 향후 다양한 알고리즘에 따른 성능 평가를 수행한다.

• **버퍼 알고리즘에 따른 실험**

본 논문에서는 RTP 데이터를 버퍼링하기 위해 N개의 버퍼를 가진 피어를 설정하여 실험을 수행하였다. 추후 다음과 같은 알고리즘을 적용하여 성능을 평가 하고자 한다.

피어의 버퍼는 그림 6.1과 같이 prefetch buffer와 post buffer로 구성되며, prefetch buffer는 재생될(to be played) 프레임을 가지고 있는 버퍼이고 post buffer는 재생된(played) 프레임을 가지고 있는 버퍼이다.

그림 6.1에서 피어 2가 이탈할 경우, 피어 3은 프레임 5번까지 재생한다. 피어 3은 $T_{survival}$안에 피어 1로 접속하여 피어 1로부터 프레임 6번을 수신하여 재생한다.

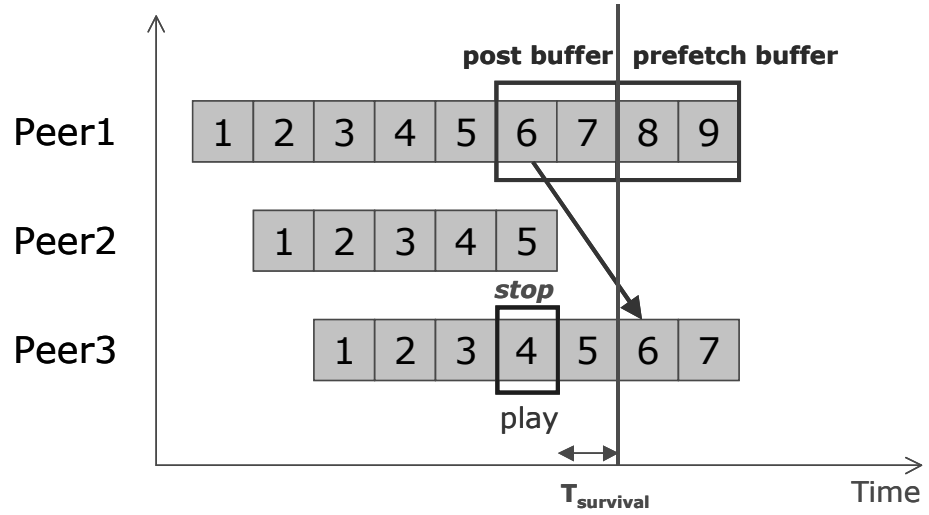

그림 6.1 Post Buffer 알고리즘

$T_{survival}$은 부모 피어가 이탈한 시점에서 자식 피어가 프레임 지터가 없을 때까지의 최대 가능한 생존 시간이며, $T_{relocate}$은 자식 피어가 새로운 부모 피어로 접속하여 첫 번째 프레임을 수신할 때까지의 시간이다. 따라서 $T_{survival} \geq T_{relocate}$을 만족하면 부모 피어의 이탈로 인한 스트림 끊김 현상을 줄일 수 있다.

- **Jitter-free를 위한 $T_{waitexit}$의 자동 조절**

현재 $T_{waitexit}$은 시스템에 설정하는 외부 상수 인자로 주어져 있지만, 프록시 서버가 각 피어의 지터값을 수집하여 $T_{waitexit}$와 지터의 관계를 관찰하고 피어 참여시에 $T_{waitexit}$ 값을 동적으로 지정하여 지터를 줄일 수 있도록 자동 조절 알고리즘을 개발한다.

참 고 문 헌

[1] K. Kikuma, Y. Morita, and H. Sunage, "A Study of a P2P Community on a P2P Communication Platform," *Proceeding of ICCT*, April 2003

[2] 박호진, 박광로, "P2P 기술 동향 및 홈 네트워크 응용," 2006

[3] Hye Sun Kim, Kitae Hwang, and Rafael A. Calvo, "An Analysis of SIP-based User Agent for Supporting Live Video Streaming," 4th Asia Pacific International Symposium on Information Technology, pp. 535-538, Jan. 2005

[4] 김혜선, 황기태, "라이브 비디오 스트리밍을 지원하는 RTC 기반 홈 게이트웨이의 설계 및 구현," 정보처리학회 논문지 C, 2005.8

[5] Hye Sun Kim, Nam Yun Kim, and Kitae Hwang, "Performance Parameters for Network Accessibility in MPEG-4 FGS Video Streaming," 5th Asia Pacific International Symposium on Information Technology, pp. 11-14, Jan. 2006

[6] PeerCast, http://www.peercast.org

[7] X. Liao, H. Jin and Y. Liu, "AnySee : Peer-to-Peer Live streaming," *In Proc. of IEEE INFOCOM*, pages 372-379, 2006

[8] T. Hama, K. Asatani and H. Nakazato, "P2P Live Streaming System with Low Signal Interruption," *In Proceedings of the 18th AINA IEEE*, pages 605-610, 2004

[9] V.N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai, "Distribution Streaming Media Content using Cooperative Networking," *In Proceedings of ACM NOSSDAV'02*, May 2002.

[10] NS2 Manual, http://www.isi.edu/nsnam/ns/ns-documentation.html

[11] Napster, http://opennap.sourceforge.net/napster.txt, 2000

[12] M.Ripeanu, "Peer-to-Peer Architecture Case Study: Gnutella Network," *In Proceedings of IEEE 1st Int'l Conf. on Peer-to-Peer Computing*, 2001

[13] JXTA, http://www.jxta.org/

[14] 고선기, 권혁찬, 나재훈, "P2P 관련 국제 표준화 동향," 2006

[15] Skype, http://www.skype.com

[16] I. Clarke, "Freenet's Next Generation Routing Protocol," http://freenet.sourceforge.net/index/php?page=ngrouting, 2003

[17] P2P-Radio, http://p2p-radio.sorceforce.net, 2004

[18] SCVI, http://www.scvi.net, 2004

[19] A. Rowstron and P.Druschel, "Storage Management and Caching in PAST, a Large-scale, Persistent Peer-to-Peer Storage Utility," *In 18th ACM SOSP'01*, 2001

[20] Pasta, http://www..cam.ac.uk/users/tlh20/papers/mph-pasta.pdf

[21] J. Kubiatowica, D. Bindel and Y. Chen, "OceanStore: an Architecture for Global-scale Persistent Storage," *In Proc. of ACM*, pages 190-201, 2000

[22] SETI@home, http://setiathome.berkeley.edu/

[23] Brent B. Welch, Ken Jones,"Practical programming in Tcl/Tk"

[24] NS2 tutorial, http://www.isi.edu/nsnam/ns/tutorial/index.html

[25] 배성수, 한종수, "네트워크 시뮬레이터," 세화도서, 2006

[26] NAM, http://www.isi.edu/nsnam/nam/

[27] RTP, H. Schulzrinne, S. Casner and R. Frederick, RFC 3550

[28] D.A. Tran, K. A. Hua, and T. T. Do, "ZIGZAG : An Efficient Peer-to-Peer scheme for Media Streaming," I*n Proceedings of IEEE INFOCOM 2003*, April 2003

[29] RTSP, H. Schulzrinne, A. Rao and R. Lanphier, RFC 2326

[30] H.264/AAC Kompressor, http://www.ateme.com/BB_kompressorAVC.php

ABSTRACT

NS2 based Simulator for Performance Evaluation of P2P Streaming Systems

Kim, Hye Sun
Major in Computer Engineering
Dept. of Computer Engineering
Graduate School
Hansung University

An Internet streaming system consists of a media server, a streaming server, and terminals. The media server delivers multimedia contents such as video and/or audio to the streaming server, which distributes the contents to terminals as well. Existing Internet streaming systems have a bottleneck problem in the streaming server. Also, the terminals can not be attached to a streaming server more than any limit because of lack of the processing capacity of the streaming server.

As a solution to this problem, P2P distributed architectures have been proposed lately. P2P technologies enable contents to be exchanged directly between terminals. Each terminal, called a peer, has roles as both a client and a server in P2P systems. In P2P streaming systems, each peer can be a parent peer of other peers and relays the multimedia contents to its children peers from its parent peer.

Actually, however, there exist many difficulties in design and implementation of P2P streaming systems, because it needs many computers and various network topologies.

In this thesis, an NS2 based P2P streaming system simulator, called *P2PStreamSim,* has been proposed and implemented. And also a P2P streaming system model such as architecture, timing model, behavior model, and performance metrics has been analyzed and defined. Finally, a test simulation has been run and the results have been analyzed.

We conclude that this thesis made great contributions in analysis and modeling of P2P streaming system and implementation of P2P streaming system simulator in detail.