CORBA 기반의 분산 오디오/비디오 스트림 서비스 프레임워크의 설계 및 구현

김 종 현[†] · 노 영 욱[††] · 정 기 동[†††]

요 약

본 논문에서는 CORBA 환경 하에서 오디오/비디오 스트림을 효율적으로 처리하고 제어하기 위한 분산 오디오/비디오 스트림 프레임워크의 설계와 구현에 관한 내용을 기술한다. 분산 오디오/비디오 스트림 서비스 프레임워크는 오디오/비디오 스트림의 효율적인 처리와 제어 그리고 전송을 위한 소프트웨어 구성 요소들을 분산 객체들로 설계한다. 그리고 오디오/비디오 데이터의 전송 효율을 최적화하기 위하여 제어 데이터와 미디어 데이터의 전송 경로를 분리한다. 분산 객체들은 IDL로 정의하며 JAVA로 구현한다. 그리고 미디어 데이터의 캡처, 재생 그리고 통신 채널 등 디바이스에 의존적인 기능들은 JMF에서 제공하는 컴포넌트들로 구현한다. 스트림 통신을 위한 분산 객체들 간의 연결설정과 제어 절차를 보여주며, 검증을 위하여 테스트 시스템을 구축하여 성능을 실험한다. 실험 결과 연결설정 지연은 TCP 연결에 비해 다소 지연을 가지나, 미디어 데이터의 전송은 RTP/UDP 프로토콜을 사용하여 CORBA의 IIOP 프로토콜에 비해 최적화된 성능을 보여준다. 또한 미디어 데이터를 전송할 때 서비스 품질을 측정한 결과 만족할 만한 성능을 보여준다.

Design and Implementation of a Distributed Audio/Video Stream Service Framework based on CORBA

Jong-Hyun Kim[†] · Young-Uhg Lho[††] · Ki-Dong Chung[†††]

ABSTRACT

This paper present a design and implementation of a distributed audio, video stream service framework based on CORBA for efficient processing and control of audio/ video stream. We design software components which support processing, control and transmission of audio/video streams as distributed objects. For optimization of stream transmission performance, we separate the transmission path of control data and media data. Distributed objects are defined by IDL and implemented using JAVA. And device dependent facilities like media capturing, playing and communication channels are implemented using JMF (Java Media Framework) components. We show a connection establishment and control procedure of streams communication. And for evaluation, we implement a test system and experiment a system performance. Our experiments show that test system has somewhat longer connection latency time compared to TCP connection establishment, but has optimized media transmission time compared to CORBA IIOP. Also test system show acceptable service quality of media transmission.

키워드 : 코바(CORBA), 스트림(Stream), JMF, RTP, 오디오/비디오(Audio/Video)

1. 서 론

최근 고속 네트워크와 멀티미디어 컴퓨팅 기술의 발전과 더불어 VOD, 화상회의, VoIP(Voice over IP) 등과 같은 새로운 형태의 분산 멀티미디어 응용에 대한 연구가 활발히 진행되고 있다. 그러나 분산 멀티미디어 응용은 멀티미디어 데이터의 특성을 수용하기 위하여 네트워크 환경에서 실시간 데이터 전송과 연속적이며 대용량 데이터 처리가 필요하다. 따라서 분산 멀티미디어 응용에서는 이질적인 플랫폼 환경이나 과중한 정보 전송으로 인한 네트워크의 지연 등 아직 해결해야 할 많은 문제점들이 있으며, 이러한 문제점을 극복하기 위한 방법들이 연구되고 있다[1].

분산시스템 환경에서 클라이언트/서버 구조의 응용 프로그램을 구현할 때 소프트웨어나 하드웨어 플랫폼 등의 이질성으로 야기되는 상호운용성(interoperability) 문제는 응용 시스템의 개발을 매우 복잡하고 어렵게 한다. OMG(Object Management Group)에서 제안한 분산 객체 미들웨어인 CORBA (Common Object Request Broker Architecture)는 상호운용성 문제를 해결하며, 객체 지향적인 개발 방법을 제공하여 표준 개방 분산 환경의 플랫폼으로 최근 주목을 받고있다[2, 3].

CORBA의 여러가지의 장점에도 불구하고 현재 구현된

† 정 회 원 : 동의공업대학 컴퓨터정보계열 교수
†† 종신회원 : 신라대학교 컴퓨터교육과 교수
††† 종신회원 : 부산대학교 전자계산학과 교수
논문접수 : 2002년 3월 5일, 심사완료 : 2002년 6월 11일

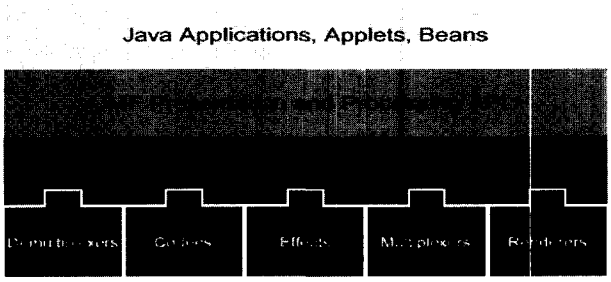
대부분의 CORBA는 성능의 최적화 문제, 실시간 및 QoS (Quality of Service) 기능과 연속 미디어의 처리에 대한 기능이 부족하여 분산 멀티미디어 응용에 효율적으로 적용되기 어렵다[4-7]. OMG에서도 이러한 점을 인식하고 최근 CORBA 상에서 오디오/비디오 스트림을 효율적으로 처리하기 위한 RFP(Request For Proposal)을 제시하고 있다[8].

본 연구에서는 CORBA 환경 하에서 효율적으로 오디오/비디오 스트림을 처리하기 위한 소프트웨어 기반 구조인 분산 오디오/비디오 스트림 서비스 프레임워크를 설계하고 구현한다. 분산 오디오/비디오 스트림 서비스 프레임워크는 오디오/비디오 스트림의 처리와 제어 및 전송을 지원하기 위해 관련된 분산 객체들을 IDL(Interface Definition Language)로 새로이 정의한다. 이러한 분산 객체들은 JAVA로 구현되며, 미디어 데이터의 캡쳐, 재생 그리고 전송 채널 등 미디어 디바이스에 의존적인 기능들은 JMF(Java Media Framework)에서 제공하는 컴포넌트들로 구현된다. 그리고 오디오/비디오 스트림 데이터의 전송 효율을 최적화하기 위하여 제어 데이터와 스트림 데이터의 전송 경로를 분리한다. 연결설정이나 전송 제어 그리고 디바이스의 제어 등과 같은 분산 객체들간의 제어 정보 교환을 위한 모든 제어 데이터들은 ORB(Object Request Broker)를 통하여 CORBA의 통신 스택인 IIOP(Internet Inter-ORB Protocol)/GIOP(General Inter-ORB Protocol)로 전달된다. 그러나 실시간 특성을 가지는 연속 미디어 데이터는 ORB를 거치지 않고 별도의 실시간 통신 프로토콜인 RTP(Real Time Transport Protocol)를 통하여 전달된다. 분산 오디오/비디오 스트림 서비스 프레임워크는 상용 CORBA 제품의 하나인 IONA 사의 Orbix 2000과 SUN 사의 JMF 2.0을 사용하여 테스트 시스템을 구축하고 성능을 평가한다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문과 관련된 연구와 사례를 살펴보고, 연구 배경에 대하여 소개한다. 3장에서는 분산 오디오/비디오 스트림 서비스 프레임워크의 기본 설계 목표와 설계된 시스템의 전체 구성 및 세부 사항에 대하여 기술한다. 4장에서는 3장에서 언급한 설계를 바탕으로 테스트 시스템을 구현하고 시험한 내용에 대하여 설명한다. 마지막으로 5장에서는 결론과 향후 과제에 대하여 기술한다.

2. 관련 연구

네트워크 환경 하에서 오디오/비디오 스트림 데이터를 효율적으로 처리하기 위한 연구들이 최근 활발하게 진행되고 있다. SUN 사의 JMF나 Microsoft 사의 NetShow 등은 네트워크 환경에서 오디오/비디오 스트림 데이터를 효율적으로 처리하기 위한 멀티미디어 응용 프로그램을 체계적으로 개발할 수 있도록 지원하고 있다[9, 10]. 특히 JMF는 플랫폼 독립성을 제공하고 객체 지향적인 JAVA 프로그래밍 언어에서 화상회의나 VOD 등과 같은 멀티미디어 응용을 손쉽고 체계적으로 개발하기 위한 잘 정의된 API를 제공한다. 그러나 JMF나 NetShow 등은 투명한 분산 처리환경을 제공하는 소프트웨어 구조가 아니다. 즉 한 컴퓨터 시스템 내에 존재하는 디바이스나 통신 채널 등의 소프트웨어 구성 요소들에 대해서는 통합적인 제어와 관리가 가능하지만, 이들 구성 요소들 중 일부가 네트워크 상의 임의의 컴퓨터 시스템 내에 분산되어 존재할 경우에는 통합적으로 제어하고 관리하는 것이 불가능하다. (그림 1)은 JMF 2.0의 구조를 보여 준다.



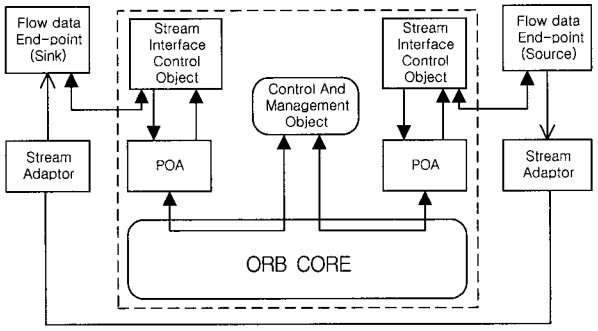
(그림 1) JMF의 상위 수준 구조

OMG에서 제안한 CORBA는 이질적인 자원으로 구성된 분산 환경에서 위치의 투명성, 프로그래밍 언어의 투명성, 하드웨어와 소프트웨어 및 네트워크 환경의 투명성 등의 상호운용성을 제공하는 분산 객체 미들웨어로 IONA 사의 Orbix, Microsoft 사의 DCOM등 주요 소프트웨어 업체들에 의해 개발되었다[11, 12]. 분산 객체 미들웨어를 기반으로 분산 스트림 서비스를 효율적으로 지원하기 위한 대표적인 연구로는 IMA의 MSS(Multimedia Stream Service)와 OMG에서 RFP(Request For Proposal)로 제안한 오디오/비디오 (OMG A/V) 스트림 서비스가 있다[8, 13].

MSS는 하드웨어, 소프트웨어, 연결 등의 유무형의 자원을 가상 장치(Virtual Device), 가상 연결(Virtual Connection), 가상 클럭(Virtual Clock) 등의 객체로 추상화하여 가상 자원 형태로 정의한 분산 멀티미디어 응용 프레임워크이다. MSS의 가상 자원들은 CORBA IDL로 정의되어 특정 환경이나 자원에 의존하지 않는 일관된 인터페이스를 사용자에게 제공하며, 이를 이용하여 분산 멀티미디어 응용을 개발한다. 또한 MSS는 미디어에 독립적인 스트림을 지원하기 위하여 MSP(Media Stream Protocol)을 사용한다. MSP는 SPX/IPX, TCP/UDP/IP, RTP/ST-II 등 다양한 전송 네트워크 상에서 동작한다. MSS는 방대하고 매우 잘 정의된 프레임워크로서 기타 유사 연구 연구들의 개념적 토대를 형성하는데 많은 영향을 미쳤으나 사양만 정의되었을 뿐 실제 구현된 사례가 없다.

OMG A/V 스트림 서비스는 오디오와 비디오 스트림의 제어와 관리에 관련된 소프트웨어 구성 요소들을 CORBA

객체로 모델링하고 IDL로 정의하여 분산 멀티미디어 응용 개발을 용이하도록 하며, 다양한 멀티미디어 디바이스 유형과 다양한 미디어 형식을 지원하도록 설계되었다. 그리고 제어 데이터와 스트림 데이터의 전송 프로토콜을 분리하여, 제어 데이터는 CORBA의 IIOP/GIOP를 통하여 전송되고, 스트림 데이터는 IIOP/GIOP 프로토콜과 독립적인 다른 적합한 프로토콜로 전송되게 하였다. (그림 2)는 OMG의 A/V 스트림 서비스의 구조이다.

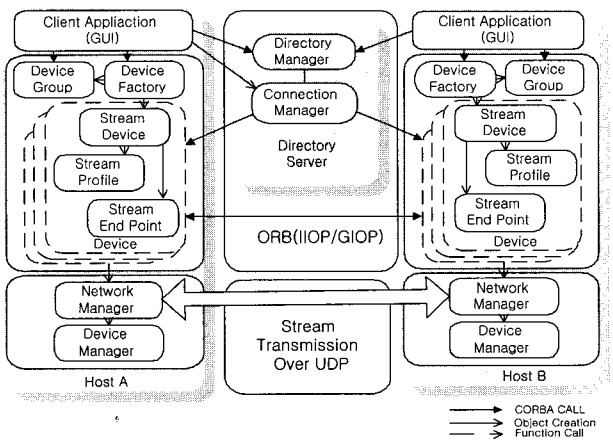


(그림 2) OMG A/V 스트림 서비스의 구조

OMG의 A/V 스트림 서비스는 다소 복잡한 API를 정의하고 있으며, 현재 일부 대학에서 실험적으로 프로토타입 수준에서 일부 구현하였으나 상용화나 표준화 단계에는 이르지 못하고 있다[14, 15].

3. 시스템 구조

본 논문에서 제안하는 분산 스트림 서비스 프레임워크는 (그림 3)과 같이 인터넷에서 하나의 디렉터리 서버(Directory Server)와 여러 개의 클라이언트들로 구성되어 있다. 디렉터리 서버와 각 클라이언트 호스트에 분산되어 존재하는 분산 객체들은 스트림의 연결설정과 전송 제어를 담당하는 CORBA 객체로 분산 스트림 서비스 프레임워크를 구성한다.



(그림 3) 분산 스트림 서비스 프레임워크의 구조

디렉터리 서버에는 연결하고자 하는 디바이스들의 위치 정보를 제공해 주는 분산 객체인 디렉터리 관리자(Directory Manager)와 디바이스들간의 연결설정과 전송 제어를 위한 연결 관리자(Connection Manager)가 존재한다. 연결 관리자는 하나의 스트림 전송 세션을 의미하며, 디렉터리 관리자 객체에 의하여 생성되어 관리된다.

각 클라이언트 호스트에는 멀티미디어 디바이스의 제어에 관련된 사항을 정의한 객체들의 그룹인 디바이스 서버(Device Server)가 존재한다. 디바이스 서버는 디바이스의 일반적인 제어에 관한 사항을 정의하고 있는 스트림 디바이스(Stream Device) 객체, 디바이스간 지원하고자 하는 미디어 형식을 정의하고 있는 스트림 프로파일(Stream Profile) 객체 그리고 미디어 데이터의 네트워크 상의 전송에 관련된 사항을 정의하고 있는 Stream End Point 객체를 포함하고 있다. 디바이스 그룹 객체는 여러개의 미디어 스트림 전송 세션을 그룹화하여 일괄적으로 제어하기 위한 사항을 정의하고 있다. 시스템 내의 모든 디바이스 서버는 각 클라이언트 호스트 내에 디바이스 팩토리(Device Factory) 객체에 의해 생성되고 관리된다.

클라이언트 응용은 사용자 인터페이스를 포함하는 제어 응용으로 CORBA 클라이언트 형태로 존재한다. 디바이스 서버를 비롯한 시스템 내의 분산 객체들은 CORBA 서버 형태로 존재하여 ORB가 요구하는 IDL로 표현된다. IDL로 정의된 디바이스 서버를 실제적으로 구현하는 부분은 클라이언트 호스트 내의 네트워크 전송 모듈과 미디어의 캡처와 재생 그리고 물리적인 디바이스의 관리를 담당하는 디바이스 관리자(Device Manager) 모듈을 이용한다. 따라서 클라이언트 응용은 분산시스템에서 위치에 상관없이 존재할 수 있으나, 디바이스 서버들은 기본적으로 실제적으로 미디어 데이터를 캡처하거나 재생하는 물리적인 디바이스가 있는 호스트에 존재해야 한다.

클라이언트 응용과 분산 객체들 간 혹은 분산 객체 자체들 간의 연결설정 및 전송 제어를 비롯한 모든 제어 데이터의 교환은 ORB를 통해 원격 메소드 호출(RPC)로 이루어진다. 그러나 실시간 특성을 가지는 스트림 데이터는 별도의 RTP/UDP 통신 프로토콜을 통하여 직접적으로 송수신되며 송수신된 스트림 데이터는 디바이스 관리자를 통하여 물리적인 디바이스로 전달된다.

4. 서비스를 위한 분산 객체

4.1 디렉터리 관리자

디렉터리 관리자 객체는 디렉터리 서버 내에 위치하여 디바이스 서버 간의 연결을 위한 위치 정보를 제공하며, 연결설정을 위하여 연결 관리자 객체를 생성하고 관리하는 역할을 한다. 연결을 원하는 모든 클라이언트 응용은 먼저

각 클라이언트 호스트 내에 디바이스 서버 객체를 생성하고, 디렉터리 관리자 객체에 바인드하여 사용자 정보(user ID)와 디바이스의 유형 그리고 디바이스 서버 객체의 레퍼런스(IOR : Interoperable Object Reference)를 등록한다.

디렉터리 관리자 객체에 등록된 디바이스 서버의 레퍼런스와 사용자 정보 그리고 디바이스의 유형은 데이터베이스나 연결 리스트로 구성되어, 연결을 원하는 클라이언트 응용 요구에 의해 디바이스 서버 객체 간의 연결 정보를 제공한다. (그림 4)는 디렉터리 관리자 객체의 IDL 정의이다.

```
struct FindInfo {
    String strUserID ;
    String strDevType ;
    StreamDevice sdObj ;
} ;
interface DirectoryManager {
    boolean Register(in String strUserID, in, String strDevType,
                     in StreamDevice sdObj) ;
    boolean Unregister(in String strUserID) ;
    StreamService Find(in String strUserID) ;
    boolean FindFirst() ;
    boolean FindNext() ;
    boolean GetAt(out FindInfo fiInfo) ;
    ConnectionManager CreateCM(in ConnectionManager cm) ;
    boolean DestoryCM(in ConnectionManager cm) ;
} ;
```

(그림 4) 디렉터리 관리자의 IDL 정의

4.2 디바이스 서버

디바이스 서버는 스피커나 마이크 등과 같은 물리적인 멀티미디어 디바이스의 제어와 처리에 관련된 사항을 정의한 객체들의 그룹으로 디바이스 팩토리, 스트림 디바이스, 스트림 프로파일, Stream End Point 그리고 디바이스 그룹 객체로 구성된다.

디바이스 팩토리 객체는 각 클라이언트 호스트에 하나씩 존재하여 시스템의 초기에 클라이언트 응용의 요청에 의해 새로운 디바이스 서버 객체를 생성하거나, 삭제하는 역할을 하는 팩토리(Factory) 객체이다. 따라서 모든 디바이스 서버 객체들은 디바이스 팩토리 객체를 통하여 생성된다. 그리고 디바이스 팩토리 객체는 여러 개의 디바이스 서버 객체들의 레퍼런스를 관리하기 위하여 클라이언트 호스트 내 하나의 디바이스 그룹 객체를 생성하고 관리한다. (그림 5)는 디바이스 팩토리 객체의 IDL 정의이다.

```
interface DeviceFactory {
        StreamDevice CreateSD(String DevType, int PortNo) ;
        boolean DestorySD(StreamDevice SD) ;
        DeviceGroup CreateDG(DeviceGroup DG) ;
        boolean DestoryDG(DeviceGroup DG) ;
}
```

(그림 5) 디바이스 팩토리의 IDL 정의

스트림 디바이스 객체는 시스템의 실행 초기에 디바이스 팩토리 객체에 의해 하나의 미디어 스트림 전송 당 하나씩 생성되는 객체로 일반적인 멀티미디어 디바이스에 공통적으로 적용될 수 있는 제어 인터페이스를 가지고 있으며, 디바이스의 상태와 유형 그리고 사용자 정보를 유지한다. 그리고 관련 객체인 스트림 프로파일 객체와 Stream EndPoint 객체를 생성하는 역할을 한다. (그림 6)은 스트림 디바이스 객체의 IDL이다.

```
enum DeviceState {DS_Ready, DS_Idle, DS_Active,
                  DS_Paused} ;
enum DeviceType {DT_Source, DT_Sink} ;
interface StreamDevice {
    readonly attribute String UserId ;
    readonly attribute DeviceState State ;
    readonly attribute DeviceType DevType ;
    boolean          start() ;
    boolean          stop() ;
    boolean          pause() ;
    boolean          resume() ;
    StreamEndPoint   CreateSP() ;
    boolean          deleteSEP() ;
    boolean          deleteSP() ;
} ;
```

(그림 6) StreamDevice의 IDL 정의

스트림 프로파일 객체는 디바이스 서버 사이에 지원하는 미디어 형식을 정의한다. 디바이스 서버들이 연결을 설정할 때 스트림 프로파일 객체는 SupportSinkCodec() 메소드와 SupportSourceCodec 메소드를 서로 호출하여 미디어 형식이 상호 부합되는지 확인하고, 만약 미디어 형식이 부합되지 않으면 연결은 실패한다. 그리고 스트림을 전송할 때 미디어 데이터를 가공해야 할 필요가 있으면 encoding 혹은 transcoding 등의 가공 작업을 수행한다. (그림 7)은 StreamProfile 객체의 IDL 정의이다.

```
enum AudioFormat    {ULAW_RTP, DTI_RTP, G723_RTP,
                     GSM_RTP} ;
enum VideoFormat    {JPFG_RTP, H261_RTP, H263_RTP} ;
enum AudioQoS       {sample_size, sample_rate} ;
enum VideoQoS       {frame_size, frame_rate, color_depth} ;

interface StreamProfile {
    boolean SetSinkFormat(in AudioFormat aCodec,
                          in VideoFormat vCodec ) ;
    boolean SetSourceFormat(in AudioFormat aCodec,
                            in VideoFormat vCodec) ;
    boolean SetSinkQoS(in AudioQoS aQoS) ;
    boolean SetSourceQoS(in VideoQoS vQoS) ;
} ;
```

(그림 7) StreamProfile의 IDL 정의

Stream End Point 객체는 미디어 데이터의 전송이 관련된 사항을 정의한 객체로 미디어 스트림의 실질적인 연

결 종단점을 의미한다. Stream End Point 객체는 먼저 ConnectSource()와 ConnectSink() 메소드를 통하여 상호간에 객체의 레퍼런스를 교환하고, RTP_IP() 메소드와 RTP_Port() 메소드를 호출하여 미디어 데이터 전송을 위한 네트워크 정보(IP 주소, 포트 번호)를 상호 교환한다. 그리고 RTPSend() 메소드와 RTPReceive() 메소드를 호출하여 미디어 데이터가 RTP/UDP 네트워크 프로토콜을 통하여 전송을 시작하거나 중단하도록 한다. (그림 8)은 Stream End Point 객체의 IDL 정의이다.

```
interface StreamEndPoint {
    readonly    attribute string  RTP_IP ;
    redonly     attribute long    RTP_Port ;
    boolean     ConnectSink(in StreamEndPoint Source_sep) ;
    boolean     ConnectSource(in StreamEndPoint Sink_sep) ;
    boolean     RTPSend() ;
    boolean     RTPReceive() ;
} ;
```

(그림 8) StreamEndPoint의 IDL 정의

디바이스 그룹 객체는 여러 개의 미디어 스트림 세션을 그룹화하여 전체 세션을 일괄적으로 제어하기 위한 객체이다. 이를 위하여 디바이스 그룹 객체는 특정 그룹에 속하는 각 스트림 디바이스 객체의 레퍼런스를 저장하기 위한 송수신 측의 리스트를 유지하며, join()과 leave() 메소드를 통하여 스트림 디바이스가 새로운 멤버로 그룹에 참여하거나, 기존의 그룹에서 탈퇴할 수 있도록 한다. 미디어 스트림의 전송에 참여하는 송수신 측의 StreamDevice 객체들은 DeviceGroup 객체에서 그룹화하여 연결설정 시에 인수로 전달된다.

```
Interface DeviceGroup {
    typedef     sequence <StreamDevice> SourceGroupList,
                  SinkGroupList ;
    readonly    attribute SourceGroupList, SinkGroupList ;
    boolean     Join(in StreamDevice sdObj) ;
    boolean     leave() ;
}
```

(그림 9) 디바이스 그룹의 IDL 정의

4.3 연결 관리자

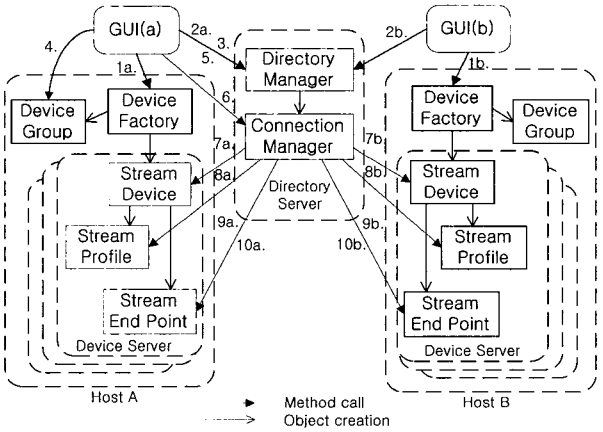
연결 관리자 객체는 연결을 원하는 디바이스 서버 간의 연결설정과 제어를 담당하는 객체로 하나의 스트림 전송 세션을 의미한다. 클라이언트 응용은 디바이스 서버 간의 연결을 위하여 먼저 송신측 그룹과 수신측 그룹에 참여한 스트림 디바이스들의 레퍼런스 리스트를 인자로 Connect() 메소드를 호출하여 연결을 시작한다. 연결이 완료되면 StartSource()와 StartSink() 메소드를 통하여 디바이스 서버의 시작을 요청하고, 디바이스 서버가 초기화되어 준비된 상태가 되면 스트림의 전송이 시작된다. 그리고 StopSource()와 StopSink() 메소드를 통하여 스트림 전송이 중지되어 초기 상태(연결된 상태)로 돌아가며, PauseSink(), PauseSource() 메소드와 ResumeSource(), ResumeSink() 메소드를 통하여 스트림의 전송이 일시 중지되거나 재개된다. 디바이스 서버 객체간 연결 해제는 Disconnect() 메소드의 호출로 이루어진다. (그림 10)은 연결 관리자 객체의 IDL 정의이다.

```
interface ConnectionManager {
    boolean     Connect(in SourceGroupList sourceObj,
                        in SinkGroupList sinkObj) ;
    boolean     Disconnect(in SourceGroupList sourceObj,
                           in SinkGroupList sinkObj) ;
    boolean     StartSource(in SourceGroupList sourceObj)
    boolean     StartSink(in SourceGroupList sinkObj) ;
    boolean     StopSource(in SourceGroupList sourceObj) ;
    boolean     StopSink(in SourceGroupList sinkObj ) ;
    boolean     PauseSource(in SourceGroupList sourceObj) ;
    boolean     PauseSink(in SourceGroupList sinkObj) ;
    boolean     ResumeSource(in SourceGroupList sourceObj ) ;
    boolean     ResumeSink(in SourceGroupList sinkObj) ;
} ;
```

(그림 10) 연결 관리자의 IDL 정의

5. 분산 객체를 통한 연결설정 및 제어

(그림 11)은 클라이언트 응용(GUI)과 분산 객체들 간의 연결과 제어 절차를 보여 준다. 송신측 호스트(Host A)에서 수신측 호스트(Host B)로 미디어 스트림을 전송하기 위한 스트림 제어 응용은 송신측 GUI(a)에 포함된다고 가정한다. 시스템의 초기 상태는 송수신측의 클라이언트 호스트 내에 CORBA 서버인 디바이스 팩토리 객체와 디렉터리 서버 내의 디렉터리 관리자 객체에서 클라이언트 응용으로부터 요청이 오기를 기다리고 있는 상태이다.



(그림 11) 연결설정 과정

단계 1 : GUI(a)와 GUI(b)는 자신의 지역 호스트(Host A, Host B) 내의 디바이스 팩토리에 바인드하여 연결하고자 하는 StreamDevice 객체들과 하나의 디바이스 그룹 객체를 생성하고, 스트림 디바이스의 레

퍼런스를 디렉터리 관리자 객체에 등록한다(1a, 1b, 2a, 2b).

단계 2 : 제어 응용인 GUI(a)는 User ID를 인자로 디렉터리 관리자 객체의 Find() 메소드를 호출하여 연결하고자 하는 송수신 측 스트림 디바이스 객체의 등록 여부를 확인하고 레퍼런스를 획득한다. 스트림 디바이스 객체들 간의 그룹 연결을 위하여 송신측 디바이스 그룹 객체의 join() 메소드를 호출하여 디렉터리 관리자로부터 획득한 송수신 측의 스트림 디바이스 객체의 레퍼런스들을 SourceGroupList와 SinkGroupList에 등록하여 디바이스 서버들 간의 그룹을 형성한다(3, 4.).

단계 3 : 디렉터리 관리자는 연결설정을 위하여 ConnectionManager 객체를 생성한다(5).

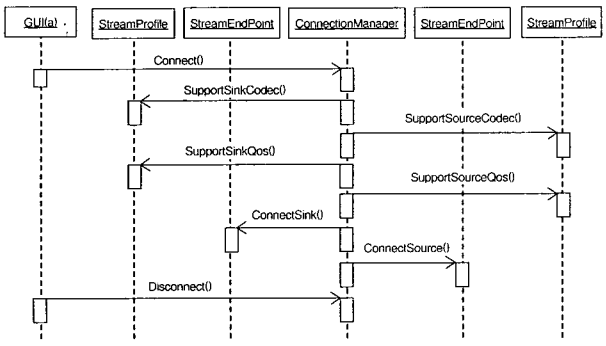
단계 4 : GUI(a)는 디바이스 그룹 객체에 등록된 송수신 측의 스트림 디바이스 객체의 그룹에 대한 레퍼런스 리스트를 인자로 ConnectionManager 객체의 connect() 메소드를 호출하여 연결에 참여한 디바이스 서버들에 대하여 연결설정을 시작한다(6).

단계 5 : ConnectionManager 객체는 디바이스 서버 그룹에 참여한 StreamDevice 객체의 CreateSEP() 메소드와 CreateSP() 메소드를 호출하여 StreamEndPoint 객체와 StreamProfile 객체를 각각 생성한다(7a, 7b).

단계 6 : ConnectionManager 객체는 StreamProfile 객체의 SupportSinkCodec() 메소드와 SupportSourceCodec() 메소드를 호출하여 상호간 지원 가능한 미디어 형식을 확인한다. 이때 상호간 지원되는 미디어 형식이 부합되지 않을 경우 연결은 종료된다(8a, 8b).

단계 7 : ConnectionManager 객체는 StreamEndPoint 객체의 ConnectSink() 메소드와 ConnectSource() 메소드를 호출하여 서로간의 객체 레퍼런스를 교환하고, 네트워크 프로토콜간의 연결을 위하여 네트워크 정보를 교환하여 연결설정을 완료한다(9a, 9b).

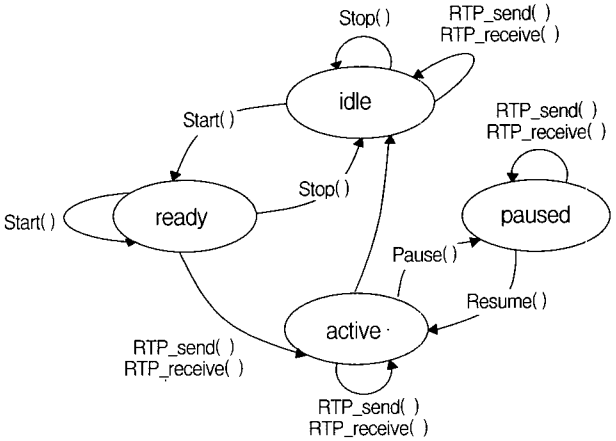
(그림 12)은 스트림 프로화일 객체를 통한 디바이스 간의 미디어 형식과 QoS 명세의의 협상 과정을 나타낸 것이다.



(그림 12) 네트워크 종단점간 연결과 매체 형의 협상

단계 8 : 연결설정이 완료되면 GUI(a)는 연결 관리자 객체의 StartSink() 메소드와 StratSource() 메소드 호출하여 디바이스 서버를 준비시킨다. 연결 관리자 객체는 다시 Stream End Point 객체의 RTPSend() 메소드와 RTPReceive() 메소드 호출하여 실질적인 미디어 스트림의 전송이 시작된다(10a,10b).

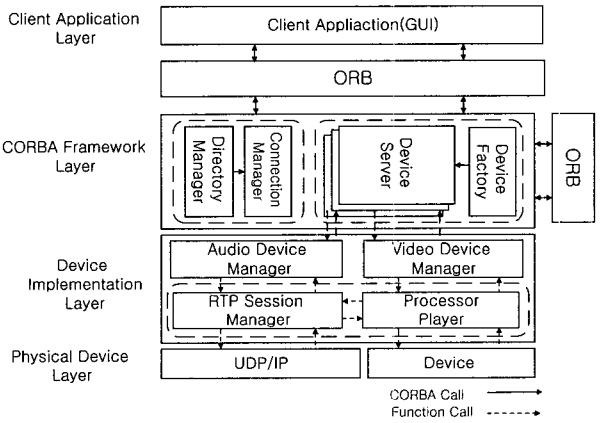
(그림 13)는 디바이스 서버의 연결설정과 스트림 전송에 따른 상태도를 보여준다.



(그림 13) 디바이스의 상태 전이도

6. 클라이언트 시스템의 내부 구조

클라이언트 시스템의 내부 구조는 클라이언트 응용 계층, CORBA 프레임워크 계층, 디바이스 구현 계층 그리고 물리적 디바이스 계층으로 구성된다. (그림 14)은 클라이언트 시스템의 내부 구조를 보여준다.



(그림 14) 클라이언트 시스템의 인터넷

최상위에는 전체 시스템의 클라이언트 역할을 하는 부분으로 클라이언트 응용 계층이 존재한다. 클라이언트 응용 계층은 사용자 인터페이스(GUI)를 포함하는 제어 응용으로 하위의 CORBA 프레임워크 계층에 존재하는 CORBA 서버

객체들에 정의된 메소드를 ORB를 통하여 호출하여 사용자의 제어 정보를 전달 해 주는 부분으로 CORBA 클라이언트 형태로 존재한다. 따라서 클라이언트 응용에서는 네트워크에 분산된 디바이스 서버 객체들을 같은 지역 공간에 존재하는 것처럼 투명하게 제어 할 수 있다.

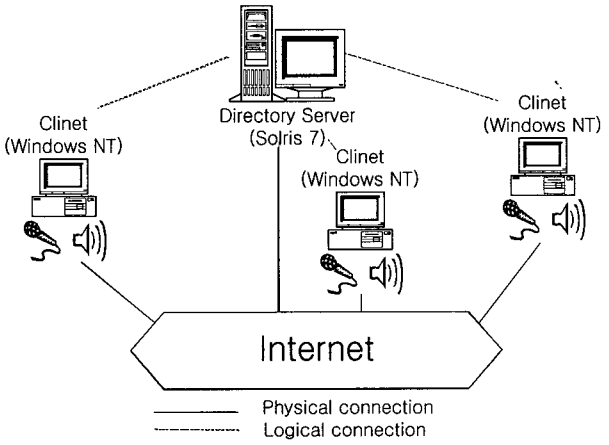
CORBA 프레임워크 계층에는 스트림 서비스를 지원하기 위하여 IDL로 정의한 분산 객체들을 존재한다. 클라이언트 응용과의 제어 메시지 교환과 클라이언트 시스템 간의 연결설정과 전송 제어 및 분산 객체들의 관리는 CORBA 프레임워크 계층에서 담당하며, 모든 제어 데이터의 교환은 CORBA ORB를 통하여 이루어 진다. 디렉터리 서버 내에 존재하는 디렉터리 관리자와 클라이언트 호스트 내에 존재하는 디바이스 팩토리는 시스템의 초기에 CORBA 구현 저장소에 등록되어 클라이언트 응용으로부터 항상 요청이 오기를 기다리는 시스템 데몬 역할을 하는 객체이다. 연결 관리자 객체는 디렉터리 관리자 객체를 통하여 생성 관리되며, 모든 디바이스 서버 객체들은 디바이스 팩토리 객체를 통하여 생성 관리된다.

디바이스 구현 계층에서는 오디오와 비디오 데이터의 캡쳐 및 재생 그리고 전송 채널 등 실제 오디오/비디오 데이터의 처리와 전송에 관련된 디바이스에 의존적인 기능을 담당한다. 이를 위하여 디바이스 및 전송 프로토콜에 관련된 모든 기능을 제공하는 Audio 디바이스 관리자와 Video 디바이스 관리자 모듈이 JMF을 이용한 패키지 형태로 제공한다. Audio 디바이스 관리자와 Video 디바이스 관리자는 CORBA 프레임워크 계층에 존재하는 분산 객체들의 제어에 의해 먼저 JMF에서 제공하는 캡쳐 및 재생을 위한 DataSource와 RTP 송수신을 위한 DataSource를 생성하고, 이를 코덱을 담당하는 Processor와 연결한다. 그리고 Session Manager로 RTP 송수신을 위한 DataSource와 캡쳐 및 재생용 DataSource를 연결하여 스트림 데이터의 전송을 시작한다. 클라이언트 응용과 분산 객체간 혹은 클라이언트 시스템 간의 모든 제어 데이터는 ORB의 IIOP/GIOP를 통하여 전송되지만, 미디어 데이터는 JMF에서 제공하는 RTP를 통한 UDP 프로토콜로 전송된다.

물리적 디바이스 계층은 네트워크 프로토콜인 UDP/IP와 스피커나 마이크와 같은 물리적인 디바이스가 존재한다.

7. 구현 및 평가

본 연구에서 설계한 내용을 구현하기 위한 테스트 시스템은 (그림 15)과 같다. 디렉터리 서버는 Solaris 7을 운영체제로 사용하는 SUN Ultra 10을 사용하였으며, 클라이언트 호스트는 화상 카메라 장치와 마이크 장치를 부착한 Windows NT 4.0을 운영체제로 사용하는 Pentium PC를 대상으로 하였다. 디렉터리 서버와 각 클라이언트 호스트에는 모두 IONA사의 CORBA 소프트웨어인 Orbix 2000을 설치하였으며 Gigabit 이더넷 백본에 연결되어 있다. 디렉터리 관리자와 연결 관리자 객체는 Solaris용 JAVA로 구현되며, 클라이언트 호스트 내의 분산 객체들은 Windows용 JAVA로 구현된다. 오디오/비디오 데이터의 캡쳐와 재생 그리고 전송 채널 등 디바이스에 의존적인 기능은 SUN 사의 JMF에서 제공하는 컴포넌트들을 이용해 구현되어 디바이스 서버 객체의 제어를 받는다. 테스트 시스템은 분산 오디오/비디오 스트림 서비스 프레임워크에서 제공하는 분산 객체들에 정의한 메소드들을 이용하여 하나의 오디오 전송 세션과 비디오 전송 세션으로 구성하였다.



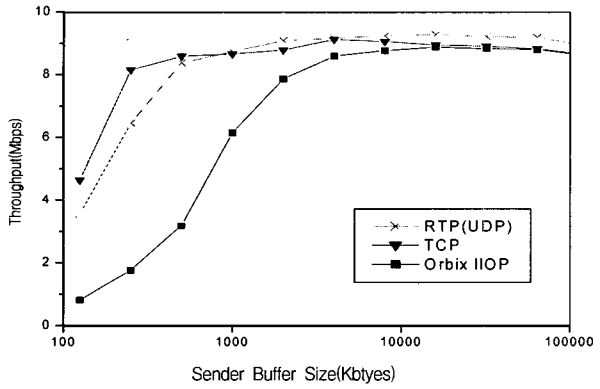
(그림 15) 테스트 시스템의 구현 환경

테스트 시스템의 미디어 데이터 전송 성능을 평가하기 위하여 RTP(UDP)와 TCP 프로토콜 그리고 Orbix의 IIOP 프로토콜로 전송되는 데이터의 전송처리율을 측정하였다. 실험은 (그림 15)의 환경에서 데이터를 전송할때 송신측의 버퍼 크기 변화(128byte~128Kbyte)에 따른 전송 처리율을 측정하였다. 이때 전송되는 데이터는 8bit의 octet 형식의 데이터 스트림을 사용한다. Orbix IIOP 프로토콜로 전송되는 octet 데이터 스트림은 oneway 호출을 통하여 수신측에게 전달된다. 송신측에서 10분 동안 연속적으로 데이터를 전송할 때 수신측에서 전달받은 octet 데이터의 합을 χ라고 하면 전송 처리율 (T)는 식 (1)과 같이 정의된다.

$$T = \frac{(\chi \times 8)}{(10 \times 60)} \times 10^{-6} (Mbps) \quad (1)$$

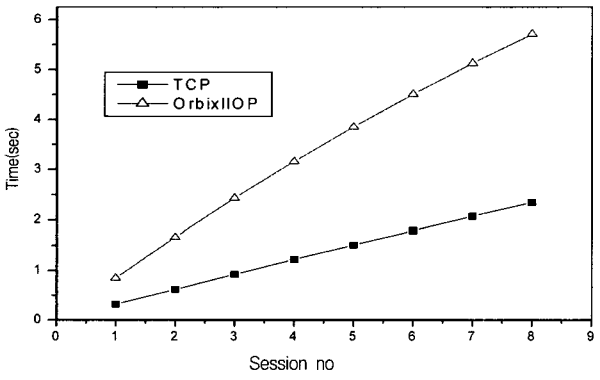
(그림 16)는 RTP(UDP)와 TCP 그리고 Orbix IIOP 프로토콜을 사용한 경우의 전송 처리율을 나타낸다. RTP(UDP) 프로토콜을 사용한 데이터의 전송 성능이 TCP보다 약간 우수함을 볼 수 있다. 그러나 Orbix IIOP 프로토콜의 경우 평균적으로 가장 낮은 전송 성능을 나타낸다. 이러한 이유는 Orbix의 전송계층인 ORB는 과도한 동적 메모리 할당과 자료의 복사 그리고 marshaling/unmarshaling 등과 같은 오버헤드로 인하여 전체 전송 성능을 현저히 저하시키기 때문이

다[14]. 따라서 실시간 데이터인 미디어 데이터는 CORBA의 ORB 전송 계층의 오버헤드를 피하기 위하여 독립적인 RTP/UDP 프로토콜로 전송하는 것이 바람직함을 알 수 있다.



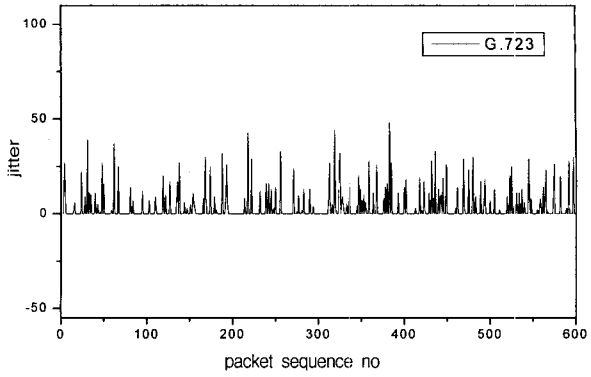
(그림 16) 처리율 결과

(그림 17)은 Orbix IIOP와 TCP 프로토콜을 사용한 연결설정 지연시간을 보여준다. 연결설정 지연시간은 클라이언트 응용에서 디렉터리 서버로부터 연결하고자 하는 두개의 디바이스에 대한 객체의 레퍼런스를 획득한 순간부터 디바이스 간의 네트워크 전송 계층의 TCP 연결이 완료되는 시점까지의 시간을 측정하였다. Orbix IIOP를 통한 연결설정이 TCP를 통한 연결설정 보다 평균적으로 0.4초 많이 지연이 발생되는 것을 알 수 있다. 그러나 제어 데이터인 연결설정 데이터의 전송량은 미디어 데이터 전송량에 비해 매우 작으며, 연결 후에는 미디어 데이터는 ORB와는 독립적인 경로로 처리되어 RTP(UDP) 프로토콜을 통하여 미디어 데이터를 전송되므로 전체 시스템의 성능에는 크게 영향을 미치지 않을 것으로 예상된다.

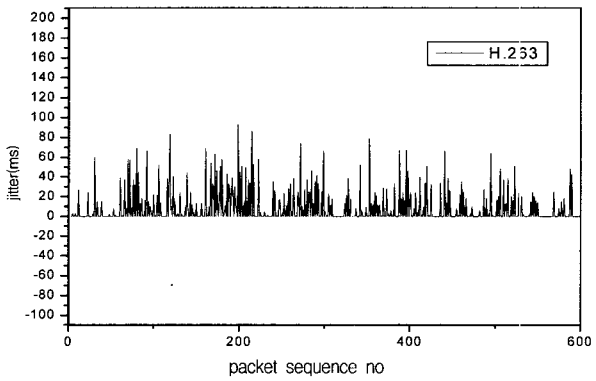


(그림 17) 연결설정의 지연

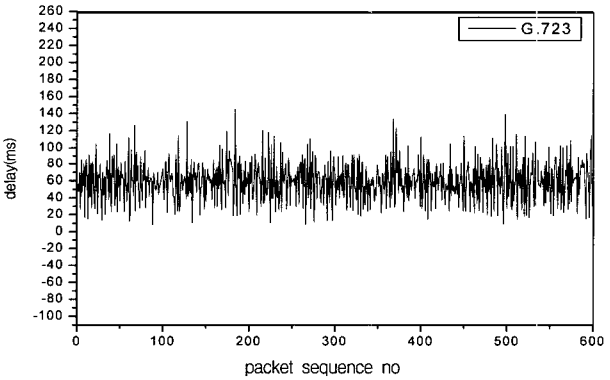
테스트 시스템의 미디어 데이터 전송 시의 서비스 품질을 알아보기 위하여 대표적인 오디오 형식인 G.723/RTP 데이터와 대표적인 비디오 형식인 H.263/RTP를 인터넷에서 전송할 때 지터(그림 18, 그림 19), 전송지연(그림 20, 그림 21)과 손실(그림 22, 그림 23)을 측정하였다.



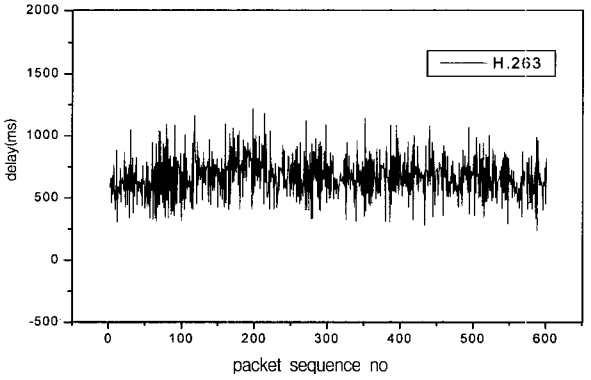
(그림 18) G.723의 지터



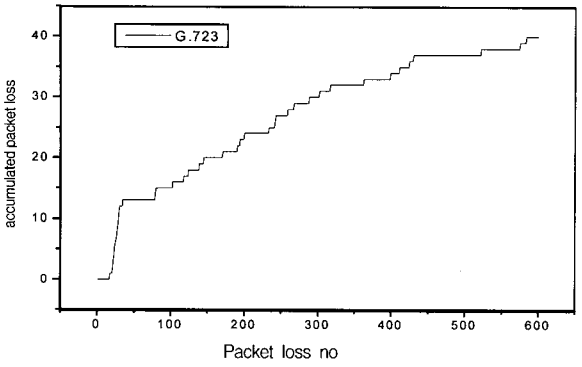
(그림 19) H.263의 지터



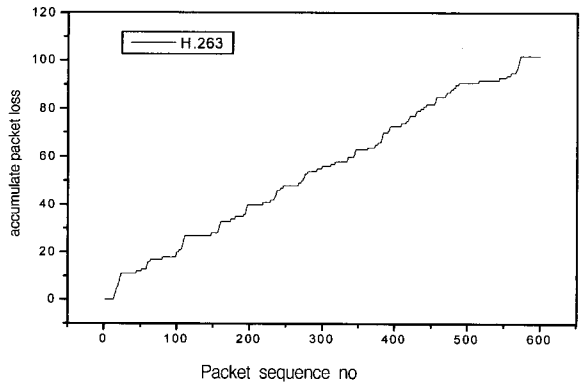
(그림 20) G.723의 지연시간



(그림 21) H.263의 지연시간



(그림 22) G.723의 손실



(그림 23) H263의 손실

측정 결과에서 보듯이 오디오 데이터와 비디오 데이터의 지터, 전송지연과 손실은 데이터의 양에 비례하여 증가됨을 알 수 있다. 일반적으로 인터넷 상에서의 지연은 필연적으로 발생하지만 안정적인 미디어 서비스를 위한 허용 범위는 70~160ms로 알려져 있다. 인터넷 트래픽이 많아지면 일반적으로 지연은 300ms.를 초과한다. ITU-T의 권고안에 따르면 인터넷 상의 허용 가능한 지연은 300 ms로 제안하고 있다. 오디오 데이터의 경우, G.723의 지연은 평균 60ms를 나타내고, H.263의 지연은 평균 700ms이다. 따라서 오디오 데이터와는 달리 비디오 데이터의 인터넷 전송은 많은 부담을 가진다. 테스트 시스템은 전송 프로토콜로 RTP/UDP를 사용하므로 VAT, VIC 등과 같은 유사한 멀티미디어 응용과 전송 서비스의 품질이 유사할 것으로 예상된다.

8. 결론 및 향후과제

본 연구에서는 분산 객체 미들웨어인 CORBA를 기반으로 오디오/비디오 스트림을 효율적으로 처리하고 제어하기 위한 스트림 분산 오디오/비디오 스트림 서비스 프레임워크를 설계하고 구현하였다. 분산 오디오/비디오 스트림 프레임워크는 분산 객체들을 새로이 정의하여 직관적이고 간단한 메소드들을 제공하여 플랫폼과 프로그래밍 언어에 독립적인 분산 멀티미디어 응용 개발을 용이하게 한다. 그리고 제어 데이터와 미디어 데이터의 전송경로를 분리하여 실시간 데이터인 미디어 데이터의 전송은 RTP 프로토콜을 통하여 전송하고, 비실시간 데이터인 모든 제어 데이터는 ORB를 통하여 전송하여 시스템의 전송 효율을 최적화한다. 설계한 분산 오디오/비디오 스트림 서비스 프레임워크는 하나의 오디오 전송 세션과 비디오 전송 세션을 가지는 테스트 시스템을 구축하여 미디어 데이터의 전송 효율과 연결설정 지연시간 등의 실험 하였다. 실험 결과 연결설정 지연은 일반적인 TCP 연결에 비해 다소 지연을 가지나, 대용량이며, 실시간 데이터인 미디어 데이터의 전송은 RTP/UDP 프로토콜을 사용하므로 CORBA의 ORB IIOP 프로토콜보다 월등한 전송 성능을 보여주었다.

현재까지의 JMF에서 제공하는 미디어 데이터의 캡쳐, 코덱 등 미디어 디바이스에 의존적인 일부 기능들은 C/C++와 같은 Native 코드로 구현한 것 보다 처리 성능이 저하된다는 단점이 있다. 따라서 향후 이러한 기능들은 JAVA에서 제공하는 JNI(Java Native Interface)를 이용하여 C/C++로 구현하여 처리 성능을 향상할 필요가 있다. 그리고 제시한 시스템에서는 네트워크의 부하에 따른 미디어 데이터의 손실이나 재전송 그리고 미디어 코덱의 변경 등 QoS 관리 기능을 제공하지 못하고 있다. 따라서 RTP에서 제공하는 제어 메시지인 RTCP를 이용하여 네트워크과 시스템의 상태에 따른 동적인 QoS 서비스를 담당하는 분산 객체들에 대한 연구가 필요하다.

본 연구 결과를 토대로 향후 인터넷 상에서 화상회의, 원격 교육 등 다양한 분산 멀티미디어 응용에 적용하고자 한다.

참 고 문 헌

[1] Guojun Lu, "Communication and Computing for Distributed Multimedia Systems," Artech House, Boston. London, 1996.

[2] Object Management Group Inc., "The Common Object Request Broker : Architecture and Specification, Revision 2.0," OMG TC Document, 1994.

[3] Jon Siegel, "CORBA Fundamental and Programming," John Wieley & Sons, Inc., 1996.

[4] T.H. Yun, J. Y. Kong and J. Won-Ki Hong, "A CORBA-based Distributed Multimedia System," Proc. of 1997 Pacific Workshop on Distributed Multimedia Systems, pp.1-8, 1997.

[5] Tomasz Mojsa, Krzysztof Zielinski, "Web enabled, CORBA Driven, Distributed Video Talk Environment on the Java Platform," Computer Networks and ISDN Systems, pp.865-873, 1997.

[6] 이성환, "CORBA에 기반한 DSM-CC 구현", KRNET '97, pp.505-517, 1997.

[7] Frank Siqueria, "The Design of a Generic OoS Architecture for Open Systems," Internal Report, Distributed Systems Group, Trinity College Dublin, 1988.

[8] Object Management Group, "Control and Management of A/V Stream specification," OMG Document telecom/97-05-07, 1997.

[9] Java Media Framework API, http : //java.sun.com/products/java-media/jmf/, Sun Microsystems Inc., 1998.

[10] Microsoft Crop., ActiveMovie SDK Documentation, http : // www.microsoft.com/devonly/tech/amovie2doc/, Microsoft Crop.
[11] IONA Technologies PLC, "Orbix Programmer's Guide," IONA Technologies Ldt., 1997.
[12] Browon, N., Kindel, C., "Distributed Component Object Model Protocol DCOM/1.0, Microsoft Crop.," Nov., 1996.
[13] IMA, "IMA : Interactive Multimedia Association Multimedia System Services," Version 1.0, June, 1993.
[14] Mungee S., Surendran N. and Schmidt D.C., "The Design and performance of a CORBA Audio/Video Stream Service," HICSS-32, Hawaii, Jan., 1999.
[15] IONA Technologies, "Orbix MX : A Distributed Object Framework for Telecommunication Service Development & Deployment," Apr., 1998.
[16] Uyless Black, "Voice Over IP," Prentice Hall, Upper Saddle River, Newsery 07458, 2000.

김 종 현

e-mail : jhkim@dit.ac.kr
1988년 부산대학교 계산통계학과(학사)
1990년 부산대학교 계산통계학과(석사)
1996년 부산대학교 전자계산학과(박사수료)
1991년~현재 동의공업대학 컴퓨터정보계열 부교수
관심분야 : 멀티미디어, 병렬/분산처리, 객체지향 컴퓨팅

노 영 욱

e-mail : yulho@silla.ac.kr
1985년 부산대학교 계산통계학과 졸업(학사)
1989년 부산대학교 대학원 전자계산학과 (석사)
1998년 부산대학교 대학원 전자계산학과 (박사)
1989년~1996년 한국전자통신연구원 연구원
1996년~현재 신라대학교 컴퓨터교육과 부교수
관심분야 : 운영체제, 멀티미디어, 병렬/분산시스템, 컴퓨터교육

정 기 동

e-mail : gdchung@pusan.ac.kr
1973년 서울대학교 공과대학(학사)
1975년 서울대학교 공과대학(석사)
1986년 서울대학교 계산통계학과(박사)
1990년~1991년 MIT, South Carolina 대학 교환교수
1998년~2001 한국멀티미디어학회 부회장
2001년~현재 한국정보처리학회 부회장
1978년~현재 부산대학교 전자계산학과 교수
관심분야 : 멀티미디어, 병렬/분산처리, 운영체제, 프로그래밍언어