Native ATM API의 설계 및 구현

성 종 진†

요 약

응용이 ATM 통신망 서비스를 이용하는 방법에는 IP over ATM이나 LAN Emulation over ATM 등과 같이 기존의 프로토콜 계층들을 활용하는 방법이 있다. 그러나 이러한 방법들로는 ATM 계층이 제공하는 Native ATM 서비스를 충분히 이용하지는 못한다. 본 논문에서는 Native ATM 서비스를 직접 이용할 수 있도록 하는 Native ATM API를 설계하고 구현한 연구 내용을 기술한다. ATM 포럼에서 권고하는 표준 지침을 따르는 Native ATM API의 사양을 자체적으로 정의하였고 이를 바탕으로 155Mbps ATM LAN 환경에서 라이브러리의 형태로 구현하였다. 본 논문에서는 먼저 Native ATM API의 개발에 있어서 고려할 사항들을 언급하였으며, 우리가 정의한 사양에서의 Native ATM 라이브러리 함수들을 설명하였다. 그리고 Native ATM API의 개발환경과 구현된 소프트웨어의 구조 그리고 개발된 Native ATM API를 이용한 응용 프로그래밍에 대한 내용들을 기술하였다.

Design and Implementation of A Native ATM API

Jong-jin Sung†

ABSTRACT

IP over ATM and LAN Emulation over ATM are common methods for applications to use ATM network. But these can hardly provide full ATM services because of legacy transport and network protocols they use. This paper presents work of design and implementation of a Native ATM API that can enable direct use of native ATM services. In our work, Native ATM API specification which accommodates ATM Forum's "Native ATM Services: Semantic Description" has been defined, and based on this, Native ATM API has been implemented in a form of library in an 155Mbps ATM LAN environment. In this paper, considerations and limitations of our development are addressed, and implementation environment, software architecture, Native ATM API library functions, and application programming using our Native ATM API are described.

1. 서 론

컴퓨터 통신 응용들의 고속 멀티미디어화 추세에 따라 통신망을 통해 전달되는 데이타가 양적으로는 대용량화 되고 있으며 질적으로는 다양화되고 있다. 이러한 고속 멀티미디어 통신 응용들은 빠른 전송능력 뿐만 아니라 전송 서비스의 품질 제어 능력을 요구한다. ATM(Asynchronous Transfer Mode)은 전송 서비스의 품질 제어 능력면에서 다른 고속 통신망들에 비해 보다 나은 서비스를 제공하고 있는 것으로 보인다.

ATM 통신망을 이용하는 응용의 개발은 점진적으로 이루어지고 있다. 우선은 기존의 통신망에서 사용

† 정 회 원: 한국전자통신연구원
논문접수: 1996년 7월 24일, 심사완료: 1997년 2월 24일

되고 있는 응용들을 그대로 ATM 상에서 이용할 수 있도록 하는 방안에 대한 연구가 활발하다. 이러한 방법으로는 IP over ATM[1]이나 LAN Emulation over ATM[2] 등이 있다. 이 방법들은 기존의 네트웍 및 데이타링크 계층 프로토콜들을 그대로 이용하고 있어서 ATM 통신망이 제공하는 우수한 성능과 장점들을 충분히 살릴 수 없다. 기존의 프로토콜들이 ATM이 보장할 수 있는 서비스의 품질을 그대로 응용에게 전달하지 못하기 때문이다. 따라서 ATM의 서비스 품질을 제대로 활용하기 위해서는 ATM 계층의 서비스를 바로 응용에게 전달할 수 있도록 해야 한다[3].

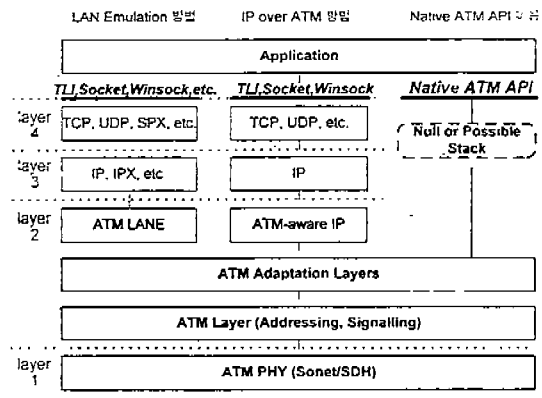
ATM 포럼에서는 이와 같이 응용들이 ATM의 서비스를 직접 이용할 수 있도록 하기 위하여 Native ATM 서비스를 제공하기 위한 응용 프로그래밍 인터페이스 (Application Programming Interface:API)를 연구하여 표준규격을 작성하려는 노력을 해왔다. 현재 ATM 포럼에서는 Native ATM API 자체의 표준을 정의하는 대신에 Native ATM 서비스에 대한 의미 명세인 "Native ATM Services:Semantic Description[4]"을 제정하여 이를 따르는 Native ATM API의 개발을 권고하고 있다. 이 표준 의미 명세를 따라서 ATM 서비스를 이용할 수 있는 실제적인 Native ATM API를 개발하도록 유도하는 것이다. ATM 포럼의 의미 명세를 따르는 ATM API의 대표적인 두가지 예를 들면 마이크로소프트사의 윈도우즈 계열 플랫폼에서 사용될 수 있는 WinSock 2 API[5, 6]와 UNIX 계열 플랫폼에서 사용될 수 있는 XTI(X/open Transport Interface)[7]와 X/Socket[8]을 꼽을 수 있다. 이 외에도 여러 API의 개발이 추진되고 있는 상황이다[9-14].

본 논문에서는 우리가 그 사양(specification)을 정의하고 Solaris 2.4 운영체제 플랫폼 상에서 구현한 Native ATM API에 대해 기술한다. ATM 포럼의 의미 명세를 따르는 Native ATM API 사양을 정의하여 나타내며, 이 사양에 맞게 라이브러리 형태로 구현한 Native ATM API 라이브러리와 이를 제공하기 위한 시스템의 구성을 설명한다.

다음 2장에서는 먼저 Native ATM API에 대한 개요 설명을 통해 Native ATM API의 필요성과 ATM 서비스를 제공하는 다른 API들과 비교되는 장단점을 분석한다. 3장에서는 Native ATM API의 개발을 위한 지침으로 ATM 포럼에서 권고하고 있는 "Native ATM Services:Semantic Description"에 대해 설명하고, 4장에서 Native ATM API 개발 사례를 알아본다. 그리고 5장에서 우리가 개발한 Native ATM API에 대해 고려사항 및 제한사항, 구현환경, 소프트웨어 구조, 라이브러리 함수 그리고 응용 프로그래밍 등의 내용을 설명하며 6장에서 결론을 맺는다.

2. Native ATM API의 개요

ATM 서비스를 이용하기 위한 프로토콜 계층 구조는 그림 1과 같이 구성될 수 있다. 그림 1에서 보는 바와 같이 ATM 서비스를 이용하는 방법으로는 기존의 트랜스포트 및 네트웍 프로토콜들을 그대로 이용하는 LAN Emulation over ATM이나 IP over ATM을 통하는 방법과 이를 통하지 않고 Native ATM API를 통하여 직접 ATM 계층의 Native ATM 서비스를 이용하는 방법이 있다.



(그림 1) ATM 서비스를 위한 프로토콜 계층 구조
(Fig. 1) Protocol Layer Stacks for ATM Services

ATM을 이용하기 위한 API라고 하면 ATM 서비스를 이용할 수 있도록 응용에게 제공되는 모든 API들을 말한다. 즉 LAN Emulation over ATM 이나 IP over ATM 상에 존재하는 TLI, Socket, WinSock 등 기존의 모든 API들도 포함할 수 있다. 그러나 이 API들은 ATM 통신망의 고유한 기능들을 충분히 이용하고자 하는 응용들에게 있어서는 진정한 의미의 ATM API라고는 할 수 없다. 따라서 Native ATM API라고

하는 진정한 의미의 ATM 서비스를 위한 API를 정의하여 개발할 필요가 있다.

이러한 Native ATM API를 이용한 방법은 LAN Emulation over ATM 이나 IP over ATM을 이용한 방법에 비해 기존의 중간 프로토콜 계층들을 생략함으로서 계층간 데이타 처리의 오버헤드를 줄일 수 있는 장점이 있다. 더우기 Native ATM API 이하 부분은 쉽게 하드웨어화 할 수 있어 한층 높은 고속의 데이타 처리를 실현할 수도 있다. 또한 ATM 계층과 AAL (ATM Adaptation Layer) 에서 제공하는 서비스를 응용이 직접적으로 접근하여 보다 폭 넓게 이용할 수 있으므로 다양한 멀티미디어 응용의 욕구를 만족시켜줄 수 있을 것이다.

반면에 Native ATM API를 이용하는 응용들은 순수 ATM 망으로만 데이타를 전송할 수 있다. LAN Emulation over ATM이나 IP over ATM 기술은 기존의 통신망들과 ATM 망간의 연동을 제공할 수 있는데 반해 Native ATM API는 기존망들과의 연동을 제공하지는 못한다. 그러나 궁극적으로는 기존의 망들에 비해 우수한 전송능력을 지닌 ATM 망에 의해 대부분의 통신망들이 교체될 것으로 기대되어 Native ATM API에 대한 요구가 늘어갈 것으로 전망된다.

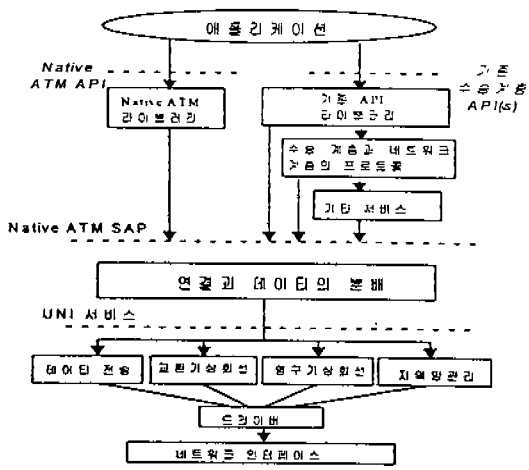
3. Native ATM Services: Semantic Description

우리가 개발한 Native ATM API는 ATM 포럼에서 권고한 "Native ATM Services: Semantic Description" 버전 1.0의 내용을 따르고 있다. ATM 포럼에서 작성하고 있는 "Native ATM Services: Semantic Description"은 ATM 망의 본질적인 서비스를 정의하며 그러한 ATM 서비스를 응용이 이용할 수 있도록 하기 위해 요구되는 프리미티브들을 기술하고 있다. ATM 포럼의 SAA (Service Aspects and Applications)/API 그룹에 의해 작성되고 있으며, 1996년 2월에 버전 1.0이 발표되었고 현재 버전 2.0으로 발전시키기 위한 작업을 진행 중에 있다.

이 의미 명세에서 정의하고 있는 본질적인 ATM 서비스 즉 Native ATM 서비스의 내용은 다음과 같다.

- ATM 계층과 AAL을 이용한 데이타 전송
- 교환 가상 회선(Switched Virtual Circuit: SVC)의 설정
- 영구 가상 회선(Permanent Virtual Circuit: PVC)의 설정
- 트래픽 형태, QoS(Quality of Service)를 고려한 트래픽 관리
- 응용이나 엔티티에 대한 연결과 그와 관련된 데이타의 바른 분배
- 지역적인 망관리 기능

이러한 서비스는 ATM API 뿐만 아니라 Native ATM 서비스를 사용하고자 하는 어떠한 소프트웨어 프로그램이나 하드웨어에도 적용 가능하다.



(그림 2) Native ATM Services의 참조 모델
(Fig. 2) Reference Model for Native ATM Services

그림 2는 ATM 포럼에서 정의한 Native ATM 서비스의 참조 모델을 보여 주며 각 구성 요소들의 기능은 다음과 같다.

- Native ATM API: Native ATM 서비스를 지원하기 위한 API.
- 기존 수송 계층 APIs: 응용이 기존의 수송 계층을 접근할 수 있도록 제공된 API. sockets, XTI, WinSock, Netbios 등
- Native ATM 라이브러리: Native ATM API를 응용에게 제공하는 소프트웨어 요소.
- 기존 API 라이브러리: 수송 계층 API를 응용에게 제공하는 소프트웨어 요소.
- 수송 계층 및 네트워크 계층 프로토콜: X.25, TCP

/IP, SPX/IPX, SNA, Netbeui, Appletalk 등의 데이타 통신 프로토콜.

- 기타 서비스:ATM LAN 에뮬레이션, Circuit 에뮬레이션 등이 해당.
- 연결과 데이타의 분배:다수의 응용이 동시에 ATM망의 기능들을 사용할 수 있도록 해주는 기능을 제공.

ATM 포럼의 의미 명세가 정의하고 있는 Native ATM 서비스를 위한 프리미티브는 표 1에서 제시한 바와 같이 제어 단계 (Control Plane), 데이타 단계 (Data Plane), 관리 단계 (Management Plane) 의 세 부류로 구분할 수 있다. 제어 단계에 포함되는 서비스 프리미티브들은 연결의 상태를 제어하기 위해 사용되는 것들로서, 연결의 설정과 해제 등에 관련된 기능을 나타내며, 이들의 이용에 의해 연결의 상태가 동적으로 변하게 된다. 데이타 단계의 프리미티브들은 설정된 연결을 통해서 데이타를 전송 및 수신하는 기능을 나타낸다. 관리 단계의 프리미티브들은 지역 망 관리 기능을 나타내는데, 응용에게 망 상태 정보를 제공하고 망 관리 응용이 루프백 (loop back) 테스트나 망에 관련된 오류의 처리 등 운영 관리를 수행할 수 있도록 한다.

4. 사례 연구

Native ATM API를 개발한 사례들은 아래와 같다. 아래의 개발 사례들 중에서 XTI를 확장한 ATM API와 X/Socket을 확장한 ATM API 그리고 WinSock 2 API는 ATM 포럼에 의해 그들의 의미 명세를 따르는 사양을 정의한 것으로 인정받았으며, 앞으로 각각 UNIX와 마이크로소프트 윈도우즈 환경의 통신 응용 제작에 영향력을 미칠 것으로 전망된다.

■ XTI의 확장 및 X/Socket의 확장 Native ATM API

XTI와 X/Socket은 X/Open 그룹에 의해 개발된 통신 API들로써 UNIX 환경의 업계 표준으로 인식되고 있다. XTI를 확장한 ATM API[7]와 X/Socket을 확장한 ATM API[8]는 WinSock 2 API와 함께 ATM 포럼에 의해 표준 의미명세를 따르는 Native ATM API로 인정받았다. 현재 XTI와 X/Socket ATM API를 위한 TPI (Transport Provider Interface)의 Native ATM API화가 추진되고 있다.

■ WinSock 2 API

WinSock 2 API[5]는 마이크로소프트 윈도우즈 운영체제 상에서 ATM 프로토콜을 포함한 다수의 수송계층 프로토콜(TCP/IP, IPX/SPX, DECnet, OSI 등)을 지원할 수 있는 통신 API로 개발되었다. 관리 단계의 서비스들을 제외하고는 ATM

〈표 1〉 Native ATM 서비스를 위한 프리미티브
〈Table 1〉 Native ATM Service Primitives

	Control Plane	Data Plane	Management Plane
Request	ATM_abort_connection ATM_call_release ATM_add_party, ATM_drop_party ATM_associate_endpoint ATM_connect_outgoing_call ATM_get_local_address_list ATM_prepare_incoming_call ATM_prepare_outgoing_call ATM_query_connection_attributes ATM_set_connection_attributes ATM_wait_on_incoming_call	ATM_receive_data ATM_send_data	ATM_initiate_loopback ATM_query_mgmt_variable ATM_set_mgmt_variable
Indication	ATM_arrival_of_incoming_call ATM_call_release	ATM_receive_data	ATM_indicate_error ATM_indicate_fault_alert
Response	ATM_accept_incoming_call ATM_reject_incoming_call		
Confirm	ATM_add_party_reject ATM_add_party_success		ATM_confirm_loopback

포럼의 의미 명세를 잘 지원하고 있어서 Native ATM API로서의 기능을 제공하는 것으로 인정받았다. QoS 메카니즘과 멀티포인트 통신을 위한 풍부한 기능들을 포함하고 있으며 ATM을 위한 배려가 많은 부분 이루어졌다. UNI (User-Network Interface) 3.0과 3.1에서 정의하고 있는 ATM 서비스들을 지원하기 위해 WinSock 2에서 확장된 부분에 대한 자세한 설명이 WinSock 2 ATM Annex[6]로 포함되어 있다.

■ Linux ATM API

Linux ATM API[9]는 Socket의 도메인을 확장하여 ATM의 SVC 및 PVC 전송을 가능하게 하며 기존의 도메인을 이용한 응용도 수용하면서 디바이스 드라이버 인터페이스를 통해 직접 AAL을 제어할 수 있는 Native ATM API 이다. BSD Socket인터페이스에서의 라이브러리 형식 그대로 사용하고 Socket을 생성할때 도메인을 PF_ATMPVC 또는 PF_ATMSVC로 선택하여 ATM 망에 직접 접근할 수 있다. ATM Socket을 이용한 데이타 송수신은 주소의 지정, 최적의 전송률을 고려한 버퍼의 크기 제한사항 등을 제외하면 기존의 BSD Socket의 방법과 거의 같다. 지원되는 AAL형태는 AAL5와 널 형식이며 전송률을 최적화하기 위해 getsockopt함수를 이용하여 트래픽 매개변수 값을 살펴볼 수 있다.

■ Fore Systems사의 ATM API

Fore System사 자체의 ATM API[10]는 사용자 레벨의 ATM 라이브러리 루틴들로 ATM 데이타 링크 계층에 이식이 쉬운 형태의 API이다. 연결설정을 위한 시그널링은 SPANS (Simple Protocol for ATM Network Signaling) 프로토콜을 사용하며, 각 종단 시스템은 NSAP (Network Service Access Point) 주소에 의해 식별될 수 있다. 연결은 설정시 전이중방식, 반이중방식, 멀티캐스트 방식 중 선택하여 지정할 수 있으며 QoS는 요구하는 대역폭만을 다루어 최대 대역폭, 평균 대역폭, 평균 버스트를 지정할 수 있다. AAL은 Null 형식, AAL 3/4, AAL 5를 지원하므로 사용자가 선택하여 지정할 수 있다. Fore System사의 ATM API 라이브러리는 QoS의 협상이나 ATM SAP 주소등에 있어서ATM 포럼 SAA/API의 의미 명세에 충실히 따르지 않으며 기존의 수송계층 API를 확장하지는 않고 그와 유사한 형식으로 독자적으로 개발된 Native ATM API이다.

■ Interphase사의 ATM API

Socket 지향의 인터페이스로 데이타의 전송이 특정 QoS를 지원하기 위해 확장되고 ATM 망의 연결 설정을 지원한다. 연결 설정을 위한 시그널링은 UNI 3.0과 3.1을 지원하며 시그널링 API를 통해 액세스 할 수 있다. 주소는 UNI 3.0에서 정의된 주소 지정 체계를 따르며 AAL은 Null 형식과 AAL 5만을 지원한다. 트래픽의 특성은 순방향과 역방향 모두 최대셀전송률, 지속셀전송률, 버스트 크기를 각각 지정할 수 있으며 광대역하위계층 정보도 지정할 수 있다[12].

■ 기타 ATM API

이상에서 열거한 ATM API 이외에도 독일의 Cellware사에서 B-ISDN 서비스를 지원하기 위한 TAPI를 개발하여 상품화하였으며 ATM 포럼의 의미 명세를 따르려는 연구가 진행 중이다. 또한, VINCE API[13]는 ATM 포럼의 의미 명세를 따르지는 않지만 ATMCore 인터페이스, 프로토콜 인터페이스, UNI로 구성되어 ATM 기능을 지원한다. ATML (Advanced Telecommunications Modules Limited)사의 ATMSock ATM API[14]는 WinSock API와 유사하며 Socket의 기본 기능을 지원하는 API이다.

이상의 개발 사례들은 그 개발이 마무리된 것이 아니라 버전 업그레이드의 형식으로 계속 진행 중인 상태이며, ATM 포럼의 권고사항에 맞추어 나가려고 노력하고 있다. 현재 ATM 포럼의 권고 사항인 "Native ATM Services: Semantic Description, 버전 1.0"의 작성이 일단 정리된 시점이므로 이것에 맞추어 Native ATM API의 개발이 진행되고 있고, 다음 버전 2.0이 작성되면 또한 그에 맞추어 개발 작업이 진행될 것이다.

우리가 개발한 Native ATM API는 ATM 포럼의 의미 명세 버전 1.0을 따르는 내용으로 개발된 것이다. 포럼의 표준화 작업내용을 반영함과 동시에 우리의 개발 작업에서 발견되는 개선 사항들을 ATM 포럼에 기고할 수 있도록 노력하고 있다. 우리는 이러

〈표 2〉 ATM API들 간의 기능 비교
〈Table 2〉 Functionality Comparison between Various ATM APIs

ATM API / Characteristics	XTI	WinSock 2	Interphase's	Fore's	Linux'	Ours
Signaling	UNI 3.x	UNI 3.x	UNI 3.x	SPANS	UNI 3.x	UNI 3.x
AAL Types	1, 5, user-defined	5, user-defined	5, user-defined	Null, 3/4, 5	Null, 5	5
Transport Data Transfer	Null, SSCOP	Null, SSCOP	Null	Null	Null	Null
ATM Address	ATM SAP	ATM SAP	ATM SAP	NSAP	ATM SAP	ATM SAP
QoS & Traffic Parameters	Q.2931	Q.2931, RFC 1363	Q.2931	Own	Q.2931	Q.2931

한 개발 및 표준화 참여의 노력을 통해 곧 전개될 수 있는 Native ATM API 관련 산업의 기술력 확보에 앞장설 수 있을 것으로 기대한다.

이상의 다른 ATM API들의 기능과 우리가 개발한 Native ATM API의 기능을 비교하면 표 2와 같다.

5. Native ATM API의 개발

먼저 Native ATM API의 개발시 고려해야할 사항들과 우리가 개발한 API에서의 제한사항에 대해서 알아본다. 구현 환경 및 소프트웨어 구조에 대해 설명하고 구현된 Native ATM 라이브러리 함수 그리고 개발된 Native ATM API를 이용한 응용 프로그래밍에 대해 기술한다.

5.1. 고려사항 및 제한사항

앞장에서 설명한 Native ATM 서비스를 지원하기 위해서 Native ATM API는 트래픽 특성과 QoS 요구사항을 나타낼 수 있는 매개 변수들을 정의하고 응용과 AAL의 매개 변수들 간의 매핑을 고려해야 한다. 또한 Native ATM API는 응용에게 ATM 프로시듀어와 매개 변수의 적절한 추상화를 제공하고 기존의 응용과 상호 운용될 수 있는 측면을 고려하여야 한다. 그럼으로써, Native ATM API를 이용하는 응용은 연결 설정을 위해 AAL과 그 아래 ATM 계층의 인터페이스 뿐만 아니라 시그널링 인터페이스의 세부사항을 고려하지 않고도 프로그래밍 할 수 있게 된다.

우리가 정의한 Native ATM API의 사양은 XTI와 WinSock 2 API 그리고 Linux ATM API 등을 참고하여 작성되었으며, 이 Native ATM API를 응용에게 제공하기 위한 소프트웨어인 Native ATM 라이브러리는 XTI를 주로 참고하여 개발되었다.

개발된 Native ATM API는 기본적으로 ATM 포럼의 UNI 3.x 버전[15]을 지원한다. 그러나, UNI 3.x의 모든 기능을 지원하지는 않으며, 다음과 같은 제한사항을 갖는다.

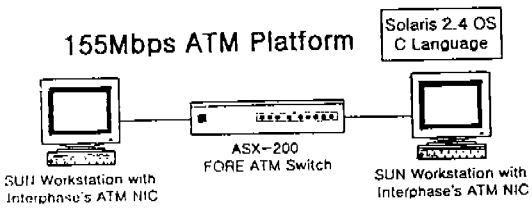
- UNI 3.x가 지원하는 ATM 베어러 서비스인 가상 경로(Virtual Path)와 가상 회선(Virtual Circuit) 서비스 중 가상 회선만을 지원하며 가상 경로는 향후 연구된다.
- UNI 3.x가 허용하는 AAL1, AAL3/4, AAL5와 사용자 정의 AAL 중에서 현재로는 AAL5만을 지원한다.
- UNI 3.x가 허용하는 두가지 AAL5 모드인 메세지 모드와 스트림 모드 중 메세지 모드만을 지원한다.

신뢰성있는 데이타 전송을 위해서 ATM 포럼에서는 AAL5 상에서 SSCOP (Service Specific Connection Oriented Protocol) 을 이용하도록 제안하고 있다. 현재 우리의 개발환경은 SSCOP을 포함하지 못하고 있어 비신뢰성 전송만을 지원하고 있다. 그러나 SSCOP을 이용하지 않고서도 신뢰성있는 전송을 지원하는 방안에 대한 연구를 현재 진행 중에 있으며, 이것은 Native ATM API와 AAL 사이에 새로운 신뢰성 제공

을 위한 프로토콜 (그림 1의 "Possible Stack"에 해당)을 삽입하는 방식이 될 것이다.

5.2. 구현환경

Native ATM 라이브러리 구현을 위한 하드웨어 환경은 그림 3과 같다. FORE사의 ASX200 스위치[11]를 중심으로 SUN 웍스테이션들이 연결되어 있으며 각각의 웍스테이션에 Interphase사의 SBus용 ATM 어댑터(adapter) 카드[12]가 장착되어 155Mbps급의 ATM 통신 플랫폼이 구축되어있다. 이 Interphase사의 ATM 어댑터 카드를 구동하는 디바이스 드라이버 상에서 Native ATM API를 개발하였다.

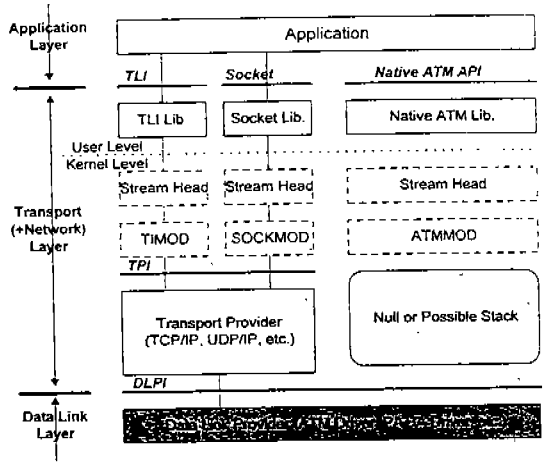


(그림 3) 하드웨어 환경
(Fig. 3) Hardware Environment

Native ATM API 개발을 위한 소프트웨어 환경으로는 Solaris 2.4를 운영체제로 이용하며 프로그래밍 언어로는 C 언어를 사용하였다. 앞에서도 설명한 바와 같이 ATM 계층의 디바이스 드라이버로는 Interphase사의 것을 가져와 약간의 수정을 거쳐 사용하였다.

5.3. 소프트웨어 구조

현재 개발된 Native ATM API는 그림 4의 오른쪽과 같은 소프트웨어 계층 구조를 갖는다. Native ATM 라이브러리 아래에 있는 ATMMOD는 프로토콜 계층의 형태는 아니며 단지 ATM API의 라이브러리 함수들을 아래의 ATM 드라이버의 프리미티브들로 연결 및 전환시켜주는 기능을 가지고 있다. 이것은 그림 4의 왼쪽 구조와 같이 기존의 TLI나 Socket과 같은 API들이 STREAMS 메카니즘으로 구현될때 TIMOD와 SOCKMOD를 두어 API를 하부 서비스제공자의 인터페이스로 연결/전환시켜주는 역할을 하도록 하는 것과 마찬가지이다. 하부 ATM 드라이버의 인터페이스 혹은 신뢰성있는 전송을 위하여 새로 정의하게 될지도 모르는 프로토콜 계층 (null or possible stack)의 인터페이스로 API를 연결시켜 주는 기능을 한다.



(그림 4) TLI, Socket, Native ATM API를 제공하는 소프트웨어 계층 구조
(Fig. 4) Software Layer Structures Supporting TLI, Socket, and Native ATM API

5.4. Native ATM 라이브러리 함수

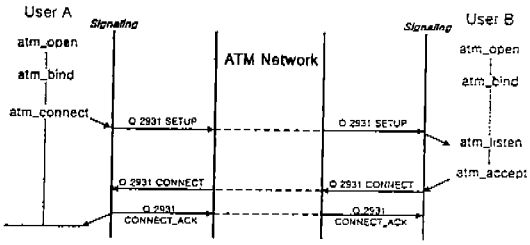
Native ATM API를 응용에게 제공하는 Native ATM 라이브러리 함수는 ATM 포럼의 구분 에에 따라 제어 단계, 데이타 단계, 관리 단계의 세가지 부류로 구분할 수 있다.

(1) 제어 단계

ATM 디바이스 드라이버는 원격지와의 연결 설정을 위해 시그널링 프로토콜을 사용한다. 시그널링 프로토콜은 ATM 포럼의 UNI 시그널링을 따른다. 연결 설정은 점대점 연결이나 점대다중점으로 이루어질 수 있다. 점대다중점 연결에서는 연결을 시작한 응용이 근원지 노드가 되며 수신 노드들을 첨가하거나 삭제할 수 있는 권한을 갖는다.

연결설정의 기본적인 절차는 그림 5와 같다. 이 절차는 XTI의 연결설정 기본절차와 동일하게 작성된 것이다. atm_open으로 API 끝점(endpoint)을 열고, atm_bind로 해당 API 끝점에 local ATM SAP 주소를 지정한다. 원격지의 ATM SAP 주소는 연결의 인지(indication)나 확인(confirmation)의 수신시에 지정된

다. 연결은 송신측의 atm_connect와 상대방의 atm_listen과 atm_accept로 이루어진다.

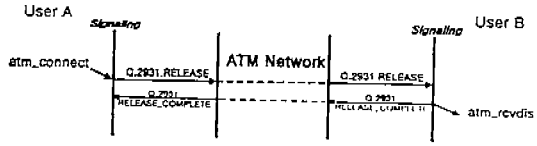


(그림 5) 연결설정의 기본 절차
(Fig. 5) Basic Procedure of Connection Establishment

연결 설정시 응용은 전송하고자 하는 트래픽의 특성 및 QoS에 대한 요구 사항을 제시할 수 있으며 원격지 호스트와 망과의 협상을 통해 적절한 수준으로 연결을 설정할 수 있다. 이는 atm_connect와 atm_listen, atm_accept의 세개의 연결설정 함수를 통해 이루어지며, 각 함수들의 call 매개인자 내에 QoS와 트래픽의 특성을 응용이 설정하여 이용할 수 있게 된다. 즉, 송신측 응용이 요구하는 QoS와 트래픽 특성을 atm_connect의 call 매개인자 자료구조 내부에 설정하여 수신측 응용에게 전달하게 되고, 수신측 응용은 전달된 특성값의 내용을 atm_listen을 통해서 받아본 후 수용의 여부를 판단하고, 그 응답을 역시 같은 형태의 call 매개인자에 담은 atm_accept(연결수락)나 atm_snddis(연결거부)를 통해 돌려주게 된다.

QoS와 트래픽 특성을 포함한 연결에 대한 여러 가지 특성은 atm_qryconattr와 atm_setconattr의 두 함수를 통해서 조정될 수 있다. 특히 PVC를 이용한 연결 설정의 경우 API 끝점을 연 이후에 atm_qryconattr와 atm_setconattr을 이용한 연결 특성의 설정 단계가 요구되며, 이 단계 이후에 atm_bind 만 수행하면 바로 데이타 송수신을 할 수 있는 상태로 접어들 수 있다.

연결의 해제는 응용으로부터의 atm_snddis에 의한 해제 요구로 이루어질 수 있고 망의 상황이나 원격지 호스트의 상황에 따라 해제당할 수 있다. 연결해제의 기본 절차는 그림 6과 같다. 점대다중점 연결에서 수



(그림 6) 연결해제의 기본 절차
(Fig. 6) Basic Procedure of Connection Release

〈표 3〉 연결 설정 및 해제를 위한 라이브러리 함수
〈Table 3〉 Library Functions for Connection Establishment and Release

함 수	기 능
atm_open()	API 연결끝점(connection endpoint)을 생성한다.
Atm_bind()	API 연결끝점에 ATM SAP 을 접합한다.
Atm_connect()	상대방 API 끝점과 연결을 설정한다.
Atm_listen()	API 연결 요구를 기다린다.
atm_accept()	API 연결 요구를 수락한다.
atm_unbind()	API 끝점에 접합된 ATM SAP 을 해제한다.
atm_close()	API 끝점을 닫는다.
atm_snddis()	API 연결해제를 요청하거나 API 끝점에 도착한 연결설정 요청을 거부한다.
atm_rcvdis()	연결 해제 및 거부를 받아들인다.
atm_add_party()	점대다중점 연결에 목적지 리프노드(leaf node) 하나를 첨가한다.
atm_drop_party()	존재하는 점대다중점 연결에 목적지 리프 노드 하나를 삭제한다.
atm_getinfo()	특정 응용에 대한 ATM 서비스 제공자의 정보를 얻는다.
atm_qryconattr()	API 연결의 특성을 확인한다.
atm_setconattr()	API 연결의 특성을 설정한다.

신 노드의 연결을 해제하고자 할 때 atm_drop_party를 사용한다.

■ atm_open(asp, flags, info): flags에 명시된 접근방식을(읽기/쓰기 모드 등)을 사용해 asp로 지정된 ATM 서비스 제공자에 대한 API 끝점으로 사용될 화일을 생성한다. 이 API 끝점을 나타내는 화일 지시자는 이 함수의 리턴값으로 주어진다. 이 함수는 ATM 포럼에서의 *ATM_associate_endpoint* 프리미티브를 구현한다.

■ atm_bind(fd, req_sap, ret_sap, qlen): ATM 프로토콜 주소인 ATM SAP 주소를 atm_open( )에 의해 얻어진 화일 지시자 fd로 지목되어지는 API 끝점과 접합시킨다. ATM 포럼의 *ATM_prepare_incoming_call*과 *ATM_wait_on_incoming_call*을 구현하며 outgoing 연결을 시작하려는 사용자에게는 *ATM_prepare_outgoing_call*을 구현한다.

■ atm_connect(fd, sndcall, rcvcall): 원격지 API 끝점과의 연결을 설정한다. atm_connect( )는 기본적으로 동기화(synchronous) 모드로 동작하여 연결이 설정되거나 연결 요청이 거부되기까지는 블럭화 상태로 지속된다. ATM 포럼에서의 *ATM_connect_outgoing_call*을 구현하며 시그널링 메세지 *Q.2931 SETUP*을 발생시킨다. 이후 상대편으로부터 *Q.2931 CONNECT* 메세지를 받아 점대점 연결이 성공되면 (동시에 *Q.2931 CONNECT_ACK* 메세지를 응답으로 되돌려 보냄), 이 함수의 리턴은 *ATM_P2P_call_active*를 구현하며, 점대다중점 연결이 성공된 것이면 *ATM_P2MP_call_active*를 구현한다.

■ atm_listen(fd, call): 원격지 API 끝점으로 부터 연결 요구를 기다린다. 구현된 atm_listen( )은 연결 인지를 받을 때까지 블럭(block)된다. 이 함수의 수행은 ATM 네트웍으로 부터의 시그널링 메세지 *Q.2931 SETUP*의 인지에 의해 결과가 나타나며, ATM 포럼의 *ATM_arrival_of_incoming_call*을 구현한다.

■ atm_accept(fd, fdnew, call): 연결 요청을 수락한다. ATM 포럼에서의 *ATM_accept_incoming_call*을 구현하며, 이 함수의 수행은 시그널링 메세지 *Q.2931 CONNECT*를 발생시킨다. 이후 네트웍으로 부터 *Q.2931 CONNECT_ACK*를 받아 점대점 연결이 성공되면 이 함수의 리턴은 *ATM_P2P_call_active*를 구현하며 점대다중점 연결의 성공이면 *ATM_P2MP_call_active*를 구현 한다.

■ atm_unbind(fd): fd에 접합되었던 ATM SAP 주소를 해제한다.

■ atm_close(fd): fd에의해 참조되는 API 끝점을 닫는다. API 끝점에 준비되어 있던 모든 자원을 삭제하고, 연계되어있던 화일을 닫는다.

■ atm_snddis(fd, call): fd로 나타내어지는 사용중이던 API 연결에 대해 연결해제를 요청하거나, fd에 도착한 새로운 연결 설정 요청 인지 (indication)에 대해 거부한다. 시그널링 메세지 *Q.2931 RELEASE*를 발생시키고 네트웍으로부터는 *Q.2931 RELEASE_COMPLEATE*을 되돌려 받는다. ATM 포럼에서의 *ATM_call_release (reqeust)*, *ATM_abort_connection*, 혹은 *ATM_reject_incomming_call*을 구현한다.

■ atm_rcvdis(fd, reason): fd에 대한 연결해제 요청 혹은 연결설정 거부를 받아들인다. 네트웍으로 부터 시그널링 메세지 *Q.2931 RELEASE*를 받은 후 *Q.2931 RELEASE_COMPLETE*을 되돌려 보낸다. 이 함수의 성공적인 리턴은 ATM 포럼에서의 *ATM_call_release (indication)* 프리미티브를 구현한다.

■ atm_add_party(fd, leaf_call): fd로 참조되는 점대다중점 연결에 하나의 리프노드를 첨가한다. ATM 포럼에서의 *ATM_add_party*를 구현하며 시그널링 메세지 ADD PARTY와 관련된다.

■ atm_drop_party(fd, leaf, dropcause): fd로 참조되는 점대다중점 연결에 하나의 리프 노드를 삭제한다. ATM 포럼에서의 *ATM_drop_party*를 구현하며 시그널링 메세지 DROP PARTY와 관련된다.

■ atm_getinfo(fd, info): 화일지시자 fd에 관련된 응용에 대한 ATM 서비스 제공자의 정보(address size, option size, 디바이스 드라이버에서의 SDU size, 서비스 형태 등)를 얻는다.

■ atm_qryconattr(fd, var_name, var_value): fd로 참조되는 API 연결의 특성을 확인한다. var_name에 의해 지정된 연결특성변수에 대한 값을 var_value에 담아 되돌려 준다. ATM 포럼에서의 *ATM_query_connection_attributes*를 구현한다.

■ atm_setconattr(fd, var_name, var_value): fd로 참조되는 API 연결의 특성을 설정한다. var_name에

의해 지정된 연결특성변수를 var_value 값으로 설정한다. ATM 포럼에서의 *ATM_set_connection_attributes*를 구현한다.

(2) 데이타 단계

응용은 atm_snd와 atm_rcv을 이용해 영구가상회선이나 교환가상회선으로 데이타를 송수신할 수 있으며, AAL을 선택할 수 있다. 현재 AAL5 방식이 지원된다.

〈표 4〉 데이타 전송을 위한 라이브러리 함수
〈Table 4〉 Library Functions for Data Transfer

함수	기능
atm_snd()	ATM 망을 통해 데이타를 송신한다.
atm_rcv()	ATM 망을 통해 데이타를 수신한다.

■ atm_snd(fd, buf, nbytes, flags): buf가 가리키는 버퍼로 부터 데이타를 읽어 fd에 의해 참조되는 API 연결에 len 바이트(bytes)만큼 전송한다. ATM 포럼에서의 *ATM_send_data*를 구현한다.

■ atm_rcv(fd, buf, nbytes, flags): fd에의해 참조되는 API 연결로부터 데이타를 수신한다. ATM 포럼에서의 *ATM_receive_data*를 구현한다.

(3) 관리 단계

지역적으로 오류 관련 정보를 얻거나 호스트 또는 스위치와 관련된 주소 정보를 요구하기 위해서 그리고 지역망의 운영 관리 및 동작 시험을 위해 다음과 같은 함수들을 사용한다.

〈표 5〉 지역적 처리 및 운영관리를 위한 라이브러리 함수
〈Table 5〉 Library Functions for Data Transfer

함수	기능
atm_error()	ATM 오류 메세지를 출력한다.
atm_getportaddr()	해당 포트 번호의 ATM 주소 리스트를 얻는다.
atm_initiate_loopback()	루프백 테스트를 수행 한다.
atm_query_mgmt_var()	지정된 관리 대상 변수의 값을 요구한다.
atm_set_mgmt_var()	지정된 관리 대상 변수의 값을 수정한다.

■ atm_getportaddr(port_num, addr_list, line_rate): 스위치의 포트 번호에 해당하는 ATM 주소 리스트를 찾는다. 이 함수는 UNI에서 명시된것처럼 ILMI (Interim Local Management Interface) 주소 등록의 결과를 보여준다.

■ atm_initiate_loopback(fd, extent, correlator): 한 연결의 루프백 테스트를 수행하는데 extent에 의해 가장 가까운 스위치로 또는 단으로 루프백 테스트를 수행한다.

■ atm_query_mgmt_var(var_name, table_index, var_value): var_name에 의해 지정된 관리 대상의 변수에 대한 값을 알기 위해 사용하며 table_index는 관리 변수가 테이블안에 위치할 때 그 테이블의 안에서의 위치를 알려준다.

■ atm_set_mgmt_var(var_name, table_index, var_value): var_name과 table_index에 의해 지정된 관리 대상 변수에 대한 값을 var_value로 수정하기 위해 사용한다.

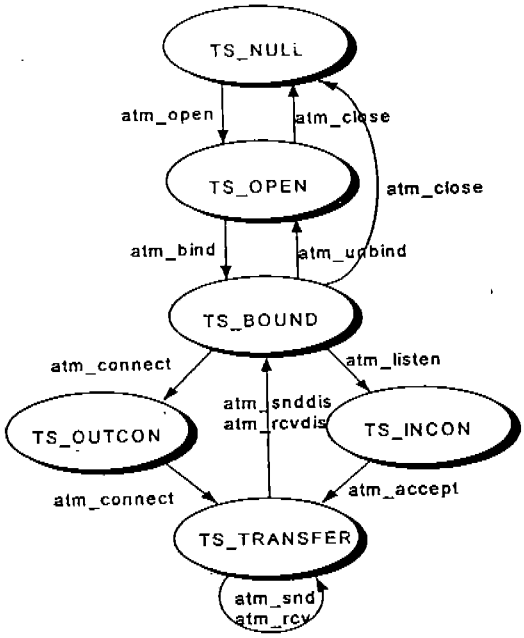
5.5. 응용 프로그래밍

구현된 Native ATM 라이브러리를 통해 제공되는 Native ATM API를 이용하여 통신 응용 프로그램을 작성하는 방법은 TLI 프로그래밍과 비슷하다. 이것은 우리가 개발한 Native ATM 라이브러리가 TLI의 확장 형태인 XTI를 기본적으로 참고하여 만들어졌기 때문이며, 기존의 TLI로 작성된 응용들을 쉽게 이식할 수 있다는 점에서 장점으로 생각할 수도 있다.

Native ATM 라이브러리의 기본적인 상태천이도를 나타내면 그림 7과 같다. API 연결의 설정과 데이타 전송 그리고 연결의 해제 단계를 천이함에 따라 그림 7과 같은 상태의 변화를 갖게된다.

개발된 Native ATM API를 이용하여 화일전송 프로그램과 "remote who" 프로그램을 포함한 몇가지 클라이언트-서버 통신 응용들을 제작해 보았다. 아직 전송속도에 대한 정확한 수치를 측정할 수 있는 성능 평가의 준비는 되어있지 않은 상태이지만 화일전송 프로그램을 이용한 전송능력의 평가는 대략적으로 이루어 질 수 있다. 즉 하나의 화일을 모두 전송하는데 걸리는 시간을 측정하여 대략의 전송률을 가늠해 볼 수 있다.

동일한 10Mbps 최대셀율 (peak cell rate) 을 할당한 가상선로에서 Native ATM API를 이용한 화일 전송의 전송률과 UDP/IP over ATM 상의 TLI를 통한 화일 전송의 전송률을 비교해본 결과 Native ATM API를 사용한 시범 응용이 IP over ATM을 이용한 응용



(그림 7) Native ATM 라이브러리의 상태천이도
(Fig. 7) State Diagram for Native ATM Library

에 비해 대략 2.6배 정도 빠른 전송률을 보였다. 또한 셀손실율 (cell loss rate) 에 있어서도 UDP와 IP의 기존 프로토콜 계층 구조를 이용한 경우 같은 환경하에서 10^{-5} 정도를 나타내었으나, Native ATM API를 이용한 경우 셀을 손실하지 않고 있다. 이것은 기존 프로토콜 계층 구조를 이용하는 경우 각 계층에서의 처리 지연으로 인한 수신 버퍼 용량의 초과가 원인일 것으로 분석된다. 동일한 셀손실율을 갖는 상황에서의 전송률를 비교하면 그 차이는 더 커지게 된다. 임의로 Native ATM API를 이용한 경우에서 IP over ATM을 이용한 경우와 같은 셀손실율 10^{-5}을 갖도록 조정할 경우 3배 이상의 전송률을 보였다.

전송능력의 향상과 함께 우리가 개발한 Native ATM API를 이용한 응용 프로그램의 장점은 atm_connect를 통한 연결설정의 단계에서 전송 선로의 최대셀율을 API 양 끝점 응용들의 합의에 의해 정할수 있게 되었다는 점이다. 즉 연결의 설정시 연결 설정을 요청하는 응용에 의해 원하는 연결의 최대셀율이 선택되어지고, 이것을 상대편 응용에게 atm_connect를 통하여 전달하면, 상대편에서는 전달된 최대셀율의 수치를 수용할 능력이 있는지를 판단하여 수용허락 혹은 원하는 수정치를 다시 연결 설정을 요청한 응용에게 전달 함으로서 최종적으로 합의된 최대셀율의 값으로 연결이 설정될 수 있게 된다. 이것은 응용에 의한 QoS 설정 및 협상의 기능을 보여주는 것으로 최대셀율 외에도 ATM 드라이버가 제공하는 여러가지 QoS 요소들에 대한 제어가 가능하도록 향상시키는 중이다.

결국 Native ATM API의 이용으로 응용이 ATM 망의 자원을 보다 효과적으로 사용하여 ATM의 서비스를 폭 넓게 이용할 수 있음을 알 수 있다.

6. 결론 및 향후 연구

ATM 통신망이 제공할 수 있는 고속의 데이타 전송 능력과 전송 서비스의 품질 제어 능력을 충분히 이용하기 위해서는 Native ATM 서비스에 직접 접근할 수 있는 Native ATM API가 필요하다.

본 논문에서는 ATM 포럼의 표준 지침인 "Native ATM Services:Semantic Description"을 따르는 Native ATM API의 사양을 정의하여 제시하였으며, 이 사양의 기본기능을 라이브러리의 형태로 구현한 내용을 기술하였다. 본 논문에서 정의한 Native ATM API 사양은 XTI, WinSock 그리고 Linux ATM API를 참조하여 작성하였으며, 구현은 Solaris 2.4 운영체제 상에서 Interphase사의 Sbus ATM 어댑터 카드와 그것의 디바이스 드라이버를 수정한 환경에서 이루어 졌다.

개발된 Native ATM API를 이용하여 전송 서비스의 품질을 제어할 수 있는 능력을 제공받을 수 있으므로 ATM 통신망 이용의 장점을 살릴 수 있게 되었다. 그리고 기존의 프로토콜 계층 구조를 이용하여 ATM 망에 접속하는 경우보다 셀전송률과 손실률 면에서 Native ATM API를 이용하는 경우가 보다 우수한 성능을 보였다.

추후의 연구로는 UNI 4.0의 내용을 반영하는 Native ATM API의 개발이 될것이다. 그리고 개발된 API에 대한 정확한 성능 평가가 수반되어야 할 것이다.

참 고 문 헌

[1] IETF, RFC 1577, 'Classical IP and ARP over

ATM', Jan., 1994.

[2] ATM Forum, 'LAN Emulation over ATM', version 1.0, af-lane-0021.000, Jan., 1995.

[3] T.Ross, "ATM APIs:The Missing Links," Data Communications, pp. 119-128, Sep., 1995.

[4] ATM Forum, 'Native ATM Services:Semantic Description', Version 1.0, af-saa-0048.000, Feb., 1996.

[5] WinSock Forum, 'Windows Sockets 2 Application Programming Interface', Revision 2.1.0, Jan., 1996.

[6] ATM Forum, 'WinSock 2 ATM Annex', ATM _Forum/96-0190, Feb., 1996.

[7] ATM Forum, 'ATM Protocol-Specific Appendix of X/Open's XTI API', ATM_Forum/95-1375R2, Dec., 1995.

[8] ATM Forum, 'XNET ATM API Specifications, Appendix Y', ATM_Forum/96-1169, Oct., 1996.

[9] W. Almesberger, 'Linux ATM API Draft', Version 0.3, Mar., 1996.

[10] Fore system, 'ATM Sbus Adapter User's Manual'.

[11] Fore system, 'ForeRunner ASX-200 ATM Switch User's Manual'.

[12] U. Gupta, 'ATM Sbus Adapter Solaris Driver Design Specification', Interphase Corp. 's Doc. No. ES00050, Dec., 1993.

[13] Naval Research Laboratory, 'VINCE 1.0.1:API Manual', Jan., 1995.

[14] ATML, 'The ATMSock ATM API', Dec., 1995

[15] ITU-T Recommendation Q.2931, 'UNI Layer 3 Specification for Basic Call/Connection Control'.

성 종 진

1990년 경북대학교 전자공학과 졸업(학사)

1992년 경북대학교 전자공학과 대학원 졸업(석사)

1992년~현재 한국전자통신연구원 근무

관심분야:ATM 통신망 및 서비스, 인터넷 서비스