碩 士 學 位 論 文

실시간 ATM 비디오 및 오디오 트래픽 분석

濟州大學校 大學院

電 氣 電 子 工 學 科

康 銀 成

2003 年 6月

실시간 ATM 비디오 및 오디오 트래픽 분석

指導敎授　　高　誠　澤

康　銀　成

이 論文을 工學 碩士學位 論文으로 提出함.

2003年 6月

康銀成의 工學 碩士學位 論文을 認准함.

審査委員長 _____

委　　員 _____

委　　員 _____

濟州大學校 大學院

2003年 6月

The Analysis of Real-Time ATM Video and Audio Traffic

Eun-Sung Kang
(Supervised by professor Sung-Teak Ko)

A thesis submitted in partial fulfillment of the requirement for the degree of Master of Engineering

2003. 6.

Department of Electrical and Electronic Engineering
GRADUATE SCHOOL
CHEJU NATIONAL UNIVERSITY

목 차

Summary

It is important to analyze characteristics of real-time ATM(Asynchronous Transfer Mode) traffic in order to control ATM traffic efficiently. VBR(variable bit rate) traffics such as real-time video and audio traffics are monitored and analyzed based on data obtained from LAN traffic at Cheju National university.

ATM traffic characteristics from the measured data are discussed. Cumulative density function(CDF) and traffic generation function(TGF) are obtained from the measured data. The traffic data generated from the TGF and the measured data are compared and the results show that the traffic data generated from the TGF is similar to the measured data.

The TGF can be used to design congestion control algorithms, to implement communication protocols, and to develop switching architectures for the ATM network.

Ⅰ. 서 론

음성, 영상, 데이터 등의 다양한 특성을 갖는 정보에 대한 서비스를 제공하기 위하여 1960년대에 별도의 전기 통신망이 구축되었다. 즉, 음성 서비스를 위해서는 공중전화망이 구축되었고, 컴퓨터 통신을 위해서는 데이터망이 출현되었으며, CATV를 위해서는 공중망이 나타났다. 반면, 이러한 통신망들은 각기 특정한 서비스를 제공하는데는 적합하나, 대역폭, 보류시간, 단말간 지연, 에러율 등의 통신망의 구성 요구 사항이 달라, 서로 다른 서비스를 전반적으로 수용하기에는 적절하지 못하다.

그리하여 하나의 통합된 망으로 모든 서비스들을 제공하는 단일 망에 대한 구상이 1970년대 초부터 거론되었다. 그 때 당시 반도체 기술의 발달로 컴퓨터의 도입이 확산되기 시작하였으며, 전송에 이어 교환 분야에서도 아날로그 방식에서 디지털 방식으로 변화가 일어났던 때였다. 단일 통신망으로 통합 개념은 자연스럽게 디지털 방식을 바탕으로 하는데 쉽게 합의되었다. 디지털 방식을 기반으로 모든 정보통신 서비스를 통합하는 개념인 종합정보 통신망(ISDN : Integrated Service Digital Network)이 1970년대 말에 확립되었으며, 필요한 세부 기술이 UN 산하 통신 전문 분야인 ITU-T(그 때 당시는 CCITT: International Telegraph and Telephone Consultative Committee, 1993년부터 ITU-T로 됨)에서 표준화 작업이 진행되어 1988년에 Blue Book으로 권고되면서 필요한 프로토콜 등이 대부분 완성되게 되었다.

1988년까지 완성된 것을 협대역 종합정보통신망(N-ISDN : Narrowband -ISDN)이라 부른다. 한편, N-ISDN 표준이 완료되어 가던 때인 1980년대 후반에 N-ISDN의 서비스 제공 측면에서 한계점이 드러나기 시작했다. N-ISDN의 서비스 통합 대상에는 천연색 동화상 정보 서비스도 포함되어 추진되었으나 N-ISDN의 기본 채널이 64kbit/s급으로 화상 신호를 코딩으로 축소하는데는 화면의 품질 면에서 성공적이지 못하였다. 이러는 가운데 HDTV(High Definition Television) 등 고품질의 화상 정보 단말의 연구에 박차가 가해지면서 N-ISDN으로 수용하는

것이 불가능하다는 결론에 도달하였고, 새로운 개념의 도출이 필요하게 되었다. 그래서 1980년대 중반부터 B-ISDN(Broadband-ISDN)의 개념이 구체화되기 시작하였고, 이를 실현하기 위한 방법으로 ATM이 출연하게 되었다.

N-ISDN은 통신방식에서 두 극단적인 회선 교환 방식과 패킷 교환 방식을 공존시키는 형태를 취하였다. N-ISDN에서는 전화와 같이 실시간성이 요구되는 서비스는 회선 교환 방식으로 처리하고 데이터 통신과 같이 대역폭의 유연성이 필요한 서비스는 패킷 교환 방식으로 처리하였다. 회선 교환 방식은 실시간성이 뛰어나지만 대역폭이 고정되어 있으므로 대역폭의 유연성은 떨어진다. 반면에 패킷 교환 방식은 시간 지연이 수반되므로 실시간성에서는 떨어지나 대역폭의 유연성 면에서는 가장 유리하다.

따라서 N-ISDN 교환기에서는 회선 교환 처리부와 패킷 처리부의 상이한 모듈이 공존하고 있는 형태로 구현되었다. 교환기의 구조로 미루어 볼 때 상이한 두 모듈을 한 시스템 내에 실현시키므로 유연성이 떨어지는 것은 당연하다고 볼 수 있다. 뿐만 아니라 N-ISDN의 경우 고정된 채널 구조로는 미래의 서비스에 요구되는 융통성을 제공 할 수 없어, 다양한 속도의 정보제공에 적합하지 않다는 비판 여론이 있었다. 그리하여 통합된 서비스의 전달 기술에 대한 논란이 있어 왔고, 융통성과 효율성을 제공할 수 있는 교환 기술이 무엇인지 의문이 제기되었다. 처음엔 패킷 교환 방식이 더 유연하므로 이 방식에 대한 검토가 있었다. 패킷 교환에서 제일 큰 약점인 실시간성을 확인하기 위하여, 패킷 교환된 음성에 대한 실험이 집중적으로 시도되었으나, 복잡한 프로토콜과 지연이 필수적으로 수반되는 패킷 교환 방식으로는 불가능하다는 결론에 도달했다. 그러나, 패킷 교환 방식의 개념적인 장점은 매력적인 것으로 남았다. 이러한 장점들은 매우 높은 처리 속도, 신속한 패킷 처리 및 최소 큐잉 지연을 목표로 하는 새로운 패킷 교환의 개발에 기여하였다. 이렇게 한 단계 발전된 것을 고속 패킷 교환(Fast Packet Switching)이라 한다.

1988년 ITU-T는 고속 패킷 교환과 유사한 ATM이라는 새로운 개념을 표준화하여 B-ISDN의 기반으로 합의하였다. ATM 현존 서비스와 미래에 나타날 서비스를 지원할 수 있는 융통성의 확보와 동적인 대역폭의 할당은 물론 모든 정보

의 통합화된 전달 기능과 통계적 다중화 등의 망 자원의 효율적인 이용이라는 점에서 가장 큰 장점을 가지고 있다. 따라서 다양한 정보를 고속으로 신뢰성 있게 전송하는 디지털 방식에 의한 종합정보통신망에 요구되는 대표적인 기술이 ATM이다.

ATM포럼에서는 ATM이 멀티미디어를 수용하기 위한 서비스 방식을 고정 비트율(CBR : Constant Bit Rate)서비스, 가변 비트율(VBR : Variable Bit Rate) 서비스, 가용 비트율(ABR : Available Bit Rate)서비스, 그리고 비 규정 비트율 (UBR : Unspecified Bit Rate)서비스로 구분하여 제안하고 있다.

사용자의 영상서비스, 또는 음성 등과 같은 실시간 서비스를 요구할 때 제한된 시간 내에 데이터가 목적지에 도착하지 못하면 전송의 의미가 없어진다. 그러므로 ATM망에서 네 종류의 서비스 중 CBR과 VBR 서비스는 ABR과 UBR 서비스보다 높은 대역폭과 우선 순위를 가진다. 따라서, 서비스별 품질을 보장하고, 망의 효율적인 구축은 무엇보다도 ATM의 특성을 분석하고 이를 바탕으로 하여 관리되어야 한다.

ATM 트래픽의 특성은 트래픽 모델을 입증하고, 폭주 제어 알고리즘을 설계하고, 통신 프로토콜을 충족시키고, 스위칭 아키텍쳐를 개발하는 등 여러 가지 측면에서 활용된다. 따라서 트래픽을 생성하는 트래픽 모델을 형성하고 이를 활용하는 것은 매우 중요하다.

전형적인 시뮬레이션은 Abdelnaser Adas(1997)등은 on-off 모델을 사용하여 트래픽을 발생하였고, Abdelnaser Adas(1997), J. Ni(1996)등은 마코프 변조 포아손 방법(MMPP: Markov Modulated Poisson Process)을 이용하여 트래픽을 발생시켰고, Ronald G. Addie, Moshe Zukerman(1995)등은 가우시안 프로세스 (Gaussion Process)을 이용한 모델링으로 트래픽을 발생시켰으며, 그 외에도 Train Model 등으로 모델링을 하여 트래픽을 발생시킨다. 이러한 트래픽 모델은 많은 기능적인 측면에서 유용하나, 필수 파라미터를 정의하는데 다른 가설 등에 의존한다.

트래픽 지시자는 최고율, 평균율, 평균 버스트 길이와 같은 네트워크 트래픽 특성을 정의하는 파라미터들을 포함한 각각의 트래픽 종류와 연관이 되어 있다.

트래픽 지시자는 트래픽 세이핑, CAC, 사용자 파라미터(UPC) 등의 기본적인 요소이다. 이러한 ATM 네트워크의 트래픽 특성을 효과적으로 연구하기 위해서는 트래픽 지시자를 정의하는 실험이 중요하다.

Lilly Cheng and Herman D.Hughes(1995)는 Wan과 ATM 환경 하에서 비디오, 오디오, FTP, Telnet, Rlogind 트래픽을 측정하여 workload model을 제안하였다. 이 workload model은 함수의 집합체로 구성되어 있으며, 트래픽의 종류에 따라 각각 트래픽의 생성 함수를 제안하였다. 트래픽 생성 함수에 의해 생성된 시뮬레이션 트래픽과 측정 트래픽의 특성을 비교·분석하였으나, workload model을 형성하는 데 복잡한 과정을 거친다.

본 논문에서는 실제 트래픽의 측정과 트래픽 생성 함수의 생성을 위하여 간단하면서도, 효율적인 트래픽 발생 함수를 제안하였다. 실제 트래픽의 측정과 트래픽 발생 함수의 생성을 위하여 제주대학교의 실제 트래픽의 측정을 기반으로 하고 있다. 제주대학교 LAN에서 데이터를 송수신 할 때, 가변 비트율(VBR)인 비디오, 오디오 트래픽을 실시간으로 측정하였다. ATM 네트워크에서 실시간 오디오, 비디오 트래픽을 측정하기 위하여 모니터링 프로그램을 작성하였다. 첫째로 ATM 드라이버에서 측정 프로그램을 작성하여 코딩하였다. 지정된 측정 시간에 따라 ATM 드라이버 메모리에서는 데이터를 순차적으로 저장하였다.

또한 ATM 어플리케이션에서는 ATM 드라이버로부터 측정된 데이터들을 일정 시간 간격으로 호출하였다. 어플리케이션 메모리는 호출 요청에 의해 보내진 데이터들을 저장한다. 반복적으로 ATM 어플레케이션에서는 ATM 드라이버에 일정 시간 간격으로 데이터를 호출하게 된다. 이렇게 얻어진 데이터들을 모니터에 나타내었으며 ATM 드라이버와 어플리케이션 프로그램을 상호 호출에 의해서 ATM 트래픽을 측정하였다.

우선 측정 시간 간격을 0.001초에서 1초까지 다양하게 측정하여 그 특성을 조사하여, 최적의 측정 시간 간격(비디오 0.1초, 오디오 1초)을 찾아내었다. 측정 데이터의 샘플을 1000개로 하여서 그래프로 도시하였다. 그리고 최적의 측정 시간 간격에 의해서 하루동안 시간대별로 24번을 측정하여 누적 확률 밀도 함수(CDF : cumulative distribution function)로 나타내었다. 이 누적 확률 밀도 함수들로

부터 표준 누적 확률 밀도 함수를 구하였다. 여기서 구해진 표준 확률 밀도 함수로부터 표준 함수를 찾아내었다. 표준함수에 의해서 발생된 트래픽과 실시간 측정하여 얻은 트래픽과 비교하여 상호간의 오차를 구하여 차이가 거의 없음을 검증하였다. 마지막으로 이렇게 생성된 트래픽의 표준 함수의 활용 범위에 대해 기술하였다.

Ⅱ. ATM 트래픽 및 트래픽의 측정

1. ATM 트래픽 서비스

ATM Forum에서는 고정 비트율(CBR) 서비스, 가변 비트율(VBR) 서비스, 비규정 비트율(UBR) 서비스, 그리고 가용 비트율(ABR) 서비스로 구분하여 서비스 방식을 제안하고 있다.

(1) CBR 서비스

CBR 서비스는 연결 시간동안 계속 이용할 수 있는 일정한 양의 대역을 요구하는 연결에 사용되며, 대역폭은 PCR값으로 설정된다. CBR은 대표적인 데이터 형태는 실시간 영상이나 음성 서비스, 회선 교환 서비스이다. 대역폭 등의 결정에 사용되는 트래픽 파라미터로는 PCR, CDVT가 있으며, QoS 파라미터인 셀 손실 및 셀 지연에 대해서는 낮은 허용 오차를 갖는다.

(2) VBR 서비스

VBR 서비스는 비디오와 가변성이 있는 음성이 주요 대상이며, 비 실시간 데이터의 전송에 사용된다.

VBR 서비스는 실시간형 가변율로서 rt-VBR(Real-Time Variable Bit Rate)과 비실시간형 가변율로서 nrt-VBR (Non-Real-Time Variable Bit Rate)로 나눌 수 있다. rt-VBR 서비스는 트래픽 발생률이 시간에 따라 변하는 트래픽을 수용하기 위한 것으로 대체로 버스트율이 높으며 셀 손실율이나 셀지연시간에 매우 민감한 실시간 응용 서비스로서 그 예는 패킷화, 압축된 음성/비디오 텔레컨퍼런싱, 멀티미디어 서비스 등이다. 또한 nrt-VBR 서비스는 트래픽 발생이 시간에 따라 변하는 트래픽을 수용하기 위한 것으로 버스티한 트래픽의 특성을 가지며 셀 전송 지연에는 민감하지 않으나 셀 손실율은 매우 낮은 값을 요구하는 비실시간 응용 서비스로서 프레임 릴레이와 같이 네트웍에서 제공되는 커넥션 지향

(connection-oriented) 서비스와 유사하다. 이 서비스에 사용되는 트래픽 파라미터로는 PCR, SCR, MBS, CDVT가 있으며, rt-VBR인 경우에는 nrt-VBR보다 QoS 파라미터인 셀 지연 및 셀 손실에 대한 허용오차가 낮다.

(3) UBR 서비스

UBR 서비스는 셀 지연 및 지연 변이에 민감하지 않은 비 실시간 응용 서비스를 위한 것으로 ATM망에서 가장 낮은 QoS을 제공하는 서비스로서 CBR이나 VBR 서비스가 사용하지 않은 가용 링크 대역폭을 활용하여 트래픽을 전송하는 방식이다. ATM망에서는 UBR 트래픽 전송에 대한 셀 전송지역 및 셀 손실율을 전혀 보장하지 못한다. 즉 빈번한 셀 유실로 인한 사용자 단말기간에 재전송이 필요한 어플리케이션에 유용하다. 이러한 응용서비스로는 파일전송, 전자우편과 같은 기존의 컴퓨터 통신 응용 서비스들이다. 일반적으로 ATM 백본에 연결되는 대부분의 워크스테이션은 ATM NIC을 장착하지 않고 있다. 대신 이들 시스템은 ATM 업링크를 사용한 LAN 스위치로 연결되는데 이때 제공되는 서비스가 바로 UBR이다.

(4) ABR 서비스

ABR 서비스는 UBR과 마찬가지로 CBR이나 VBR 서비스가 연결 설정 후에 일어나는 변화로서 망에 의해 제공되는 제한된 ATM 계층 전달 특성을 제공하는 ATM 계층 서비스이다. UBR과의 가장 큰 차이점은 UBR은 전송상에 일어나는 일에 대해 보장을 하지 않으나, ABR은 셀 손실에 대한 보장을 위해 흐름 제어 메카니즘을 제공한다. 흐름 제어 메카니즘은 ATM 계층 전달 특성 변화에 따라 소스의 셀 율을 제어하기 위해 피드백 제어 기능이 적용되는데, 이 피드백 제어를 위한 정보들은 RM(Resource Management) 셀이라 불리우는 특정 제어 셀을 통해 소스에 전달한다. 현재 존재하는 대부분의 LAN Application에 서비스한다. TCP/IP, 또는 기타 Data Application을 수용하기 위하여 대역 예약을 하지 않고 Data를 전송한다. 링크가 되어 있으면 비어있는 링크 대역 전부를 Connection으로 사용 할 수 있으나 밀집되면 자동적으로 낮은 속도로 전송한다.

지연 조건은 CLP에 대한 요구가 강력하게 요구된다. 이를 Best Effort Service

라 부른다. CBR, VBR, UBR과 다른 점은 PCR만 연결시 신고하고 실제의 Rate는 Feed Back 제어에 의하여 네트워크의 폭주 상황에 따라 대응하여 PCR 이하의 속도로 규제된다.

2. ATM 트래픽 측정 방법

ATM 네트워크 환경하에서 다양한 형태의 서비스를 제공하기 위하여, ATM 트래픽을 수학적으로 분석하거나 근사화된 트래픽을 발생시키고자 할 때 트래픽의 모델이 필요하게 된다.

(1) ARMA(Autoregressive Moving Average)

ARMA 모델은 30초분의 1초를 프레임 단위로 하여 n번째 프레임의 비트 속도 λ(n)이 이전 프레임의 비트 속도와 다음과 같은 관계를 갖는 트래픽의 모델이다.

$\lambda(n)=a\lambda(n-1)+bw(n)$

이러한 1차 모델로서, 1차 모델만으로 비교적 정확한 화상 정보를 나타낼 수 있다. 그러나, 해석이 복잡하므로 수학적인 분석에는 사용되기 어렵고 시뮬레이션 등에서 화상 신호원의 발생 패턴으로 주로 사용된다.

(2) MMPP(Markov Modulated Poission Process)

여러 개의 중첩된 트래픽에 대하여 시간에 따라 트래픽의 비트 속도를 구하여, 연속 상태를 갖는 각각의 속도를 랜덤한 포아송 과정의 도달 시점에서 샘플링하여 근사화시킨다. 여기에서 속도는 여러 가지로 계층적으로 포아송 과정을 거쳐 근사화된다. 이러한 과정을 반복함으로써 정확한 근사값을 갖는다.

(3) workload model에 의한 방법

측정된 트래픽을 기반으로 하여 함수의 집합체로 형성된 workload model을 만든다. workload model에 정의된 트래픽의 종류에 따라 각각 트래픽의 생성 함수를 제안한다. 그리고 측정 트래픽과 시뮬레이션 트래픽의 누적 밀도 함수와 트래픽의 특성을 비교, 분석하였으나, workload model을 형성하는 데 복잡한 과정을 거친다는 단점이 있다. Lilly Cheng and Herman D.Hughes(1995)은 Wan과

ATM 환경에서 비디오, 오디오, FTP, Telnet, Rlogind 트래픽을 측정하여 workload model을 제안하였다.

(4) 제안된 간략화된 TGF 생성 방법

트래픽을 반복 측정하여 트래픽의 표준 확률 밀도 함수(CDF)를 찾아낸다. TGF를 생성하기 위하여 최소 자승 추정 방식(Marquardt, 1963)에 의하여 측정한 트래픽의 표준 확률 밀도 함수(CDF)와 근사한 최적의 트래픽 발생 함수를 생성한다. 이 방법은 workload model에 의한 방법과 유사하나, 간단하고 효율적으로 트래픽 발생 함수를 찾아낼 수 있다.

3. 실시간 ATM 트래픽 측정

그림1은 제주대학교 LAN에서의 ATM 트래픽을 분석하기 위한 순서도를 나타낸 것이다. 우선 실시간(real time) 비디오, 오디오 트래픽을 측정 시간에 따라 다양하게 측정해서 최적의 측정 시간(비디오 0.1초, 오디오 1초)을 찾아낸다. 찾아낸 최적의 측정 시간으로 비디오, 오디오 트래픽을 측정한다. 측정된 트래픽들의 확률 밀도 함수(CDF : cumulative distribution function)를 도시한다. 하루동안 24번씩 시간대별로 확률 밀도 함수들을 구하고, 그 함수들로부터 표준 확률 밀도 함수를 찾아낸다. 표준 확률 밀도 함수를 curve fitting을 실행하여 표준 함수를 구한다. 표준 함수에 의해 발생된 트래픽과 측정 트래픽을 상호 비교, 분석한다.

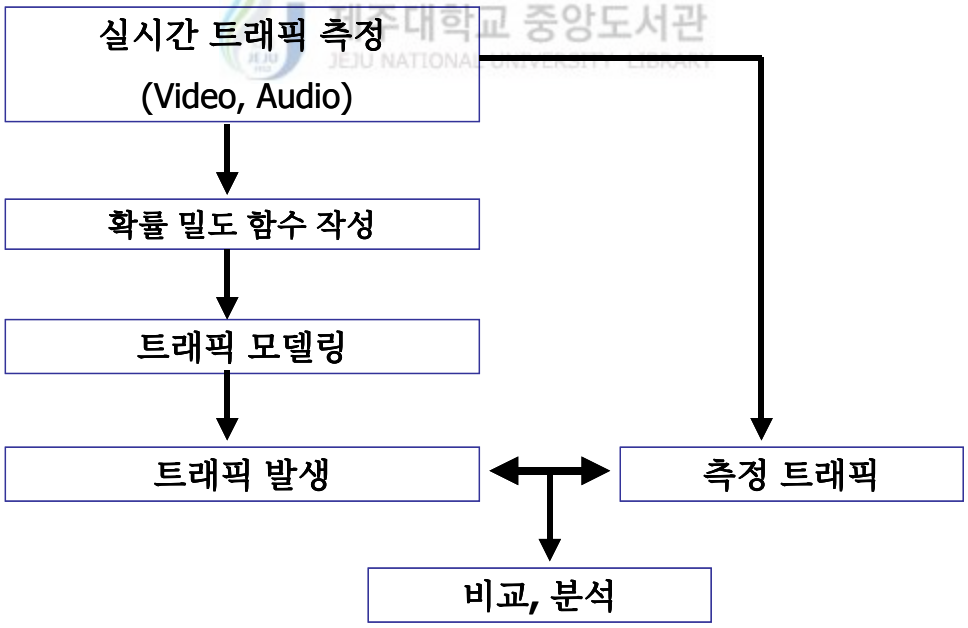


Fig. 1. The flow chart of ATM traffic.

ATM 트래픽을 측정하기 위하여 그림 2와 같이 네트워크를 구성하였다. 제주대학교의 이더넷(Ethernet) 네트워크와 ATM 네트워크와의 상호 연결은 ATM 스위치(Omni Switch)에 의해서 이루어진다. ATM 스위치는 이더넷에서 전송되는 패킷을 ATM에서 전송되는 셀로 바꾸어 준다. ATM 스위치와 ATM 155bps의 ATM 카드를 장착하고 있는 컴퓨터와의 물리적인 연결은 상호간의 송수신을 위해 쌍방향 광케이블을 사용하였다. 또한 ATM 카드에 ATM 드라이버를 설치하고, ATM 어플리케이션에서 프로그래밍하여 ATM 드라이버와 어플리케이션의 상호간에 데이터를 송수신하면서, 비디오, 오디오신호를 실시간으로 측정하였다.

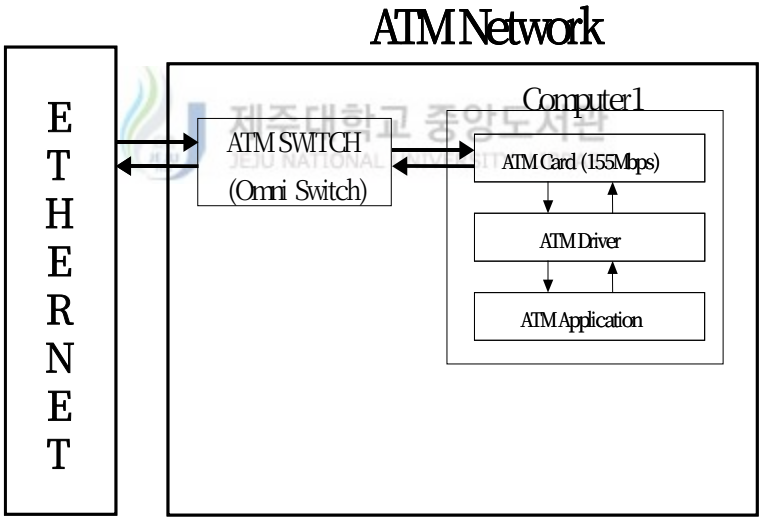


Fig. 2. Network for measuring ATM traffic.

그림 3은 ATM 드라이버 부분이 ATM 카드와 ATM 어플리케이션 부분에서의 상호 작동하는 과정을 보여준다. ATM 드라이버의 수신부분에서는 ATM 카드에서 수신한 트래픽을 받아들이는 부분이며, 받아들인 수신 데이터는 일정 시간 단위로 상위 드라이버 호출에 의해 ATM 어플리케이션 부분으로 데이터가 전송되며, 어플리케이션 부분에서 데이터가 처리된다. 연속적으로 ATM 어플리케이션 부분에서는 ATM 드라이버 부분에 데이터를 호출하고, ATM 드라이버는 ATM 어플리케이션 부분으로 전송된다. 이러한 과정을 반복 수행한다. 트래픽을 측정하고 난 후, ATM 어플리케이션 부분에서는 수신 완료 메시지를 ATM 드라이버 부분에 보내면, ATM 드라이버 부분에서는 설정된 자원(Memory)을 해제한다.

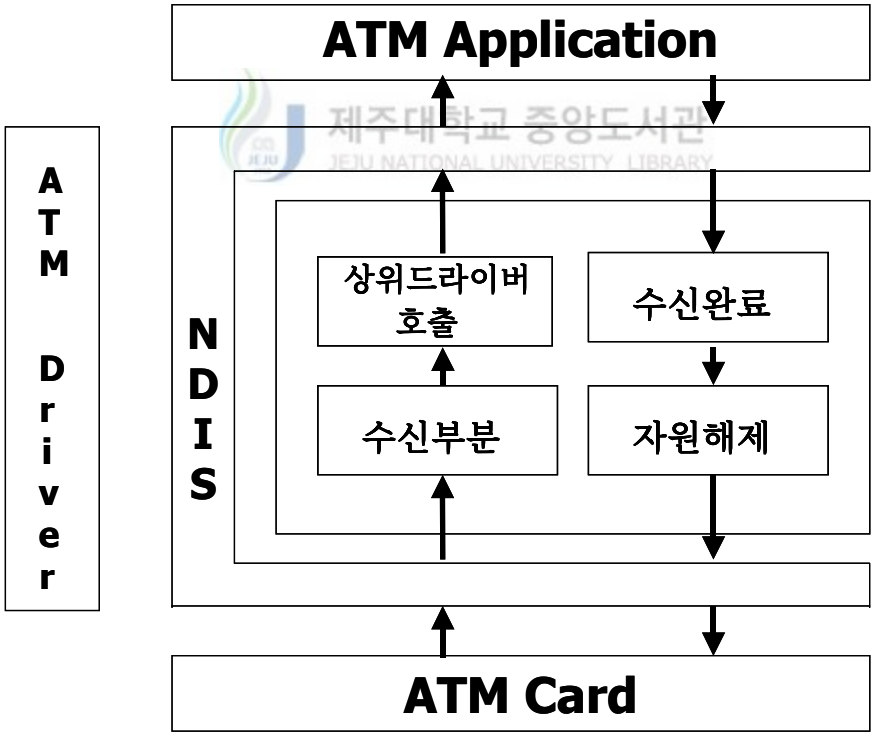


Fig. 3. The structure of network driver.

그림 4는 ATM 드라이버 부분과 어플리케이션 부분에서 쓰이는 함수들을 나타낸 것이다. ATM 드라이버부분에서 쓰이는 대표적인 함수는 Receive()함수이며, 이 함수에서 대표적으로 쓰이는 함수는 데이터를 수신하는 GetData()함수와 수신한 데이터를 분석하는 AnalyzeData() 함수로 구성되어 있다. ATM 어플리케이션 부분에서 쓰이는 대표적인 함수는 ReadData()이며, ReadData()함수에는 측정된 트래픽을 컴퓨터로 보기 위한 작업을 하여 주는 함수들(Meacell(), Cellcount(), Celltime())이 있다.

ATM 어플리케이션 부분에서 지정한 시간에 의해 주기적으로 ATM 드라이버에 호출하고, ATM 드라이버는 호출될 때마다 데이터를 ATM 드라이버에 전송한다. 이와 같은 과정을 계속 반복 수행한다.

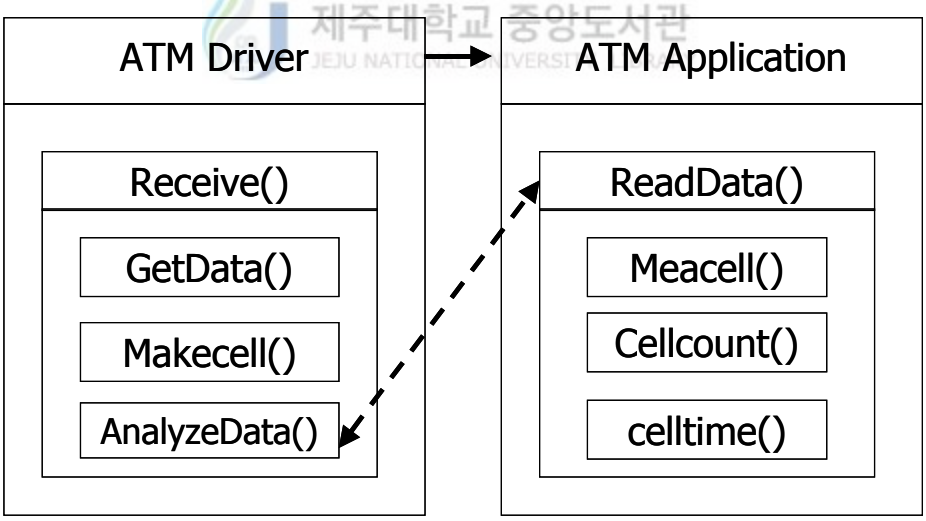
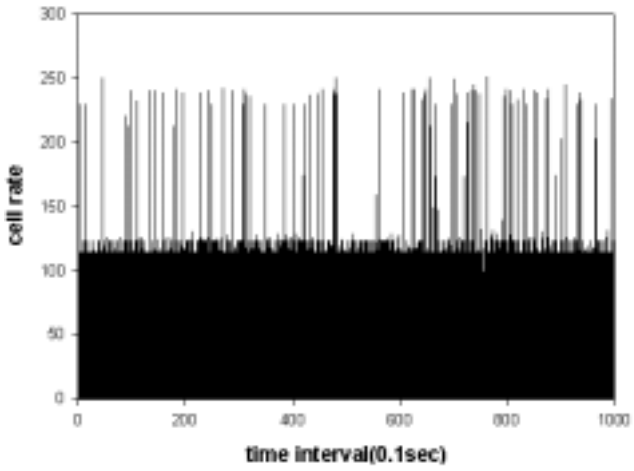


Fig. 4. The structure of measuring program.

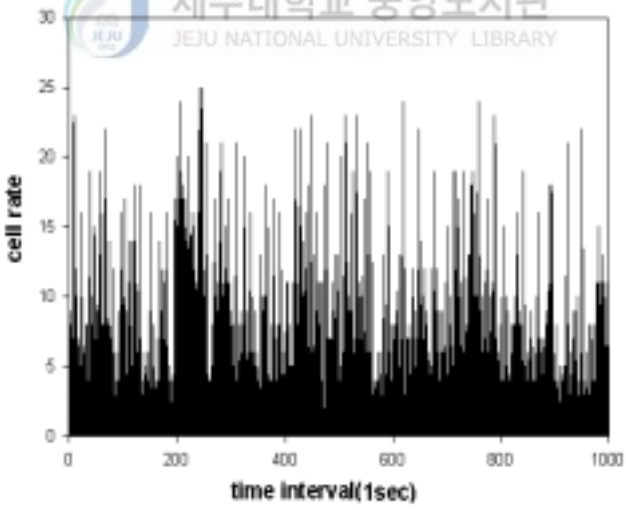
4. 제주대학교 ATM망에서 측정된 트래픽

그림 5는 제주대학교의 이더넷 네트워크와 ATM 네트워크가 상호 연계된 경우, 실제의 비디오 트래픽과 오디오 트래픽을 측정하여 보았다. 측정 시간 간격은 비디오신호는 0.1초이고, 오디오신호는 1초이다. 샘플의 수는 1000개이고, 측정시간은 비디오 트래픽인 경우 1분 40초이고, 오디오 트래픽인 경우 16분 40초(비디오 트래픽인 경우 : 0.1초 × 1000 = 100초, 오디오 트래픽인 경우 : 1초 × 1000 = 1000초)이다.

비디오 트래픽의 최고 전송율은 0.1초당 250cell이고, 평균 전송율은 0.1초당 146.06cell이다. 또한 오디오 트래픽의 최고 전송률은 1초당 25cell이고, 평균 전송률은 6.35cell이다.



(a) video



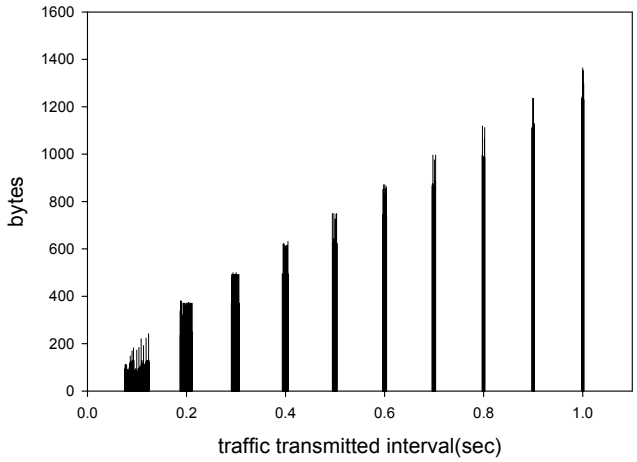
(b) audio

Fig. 5. Measured real traffic.

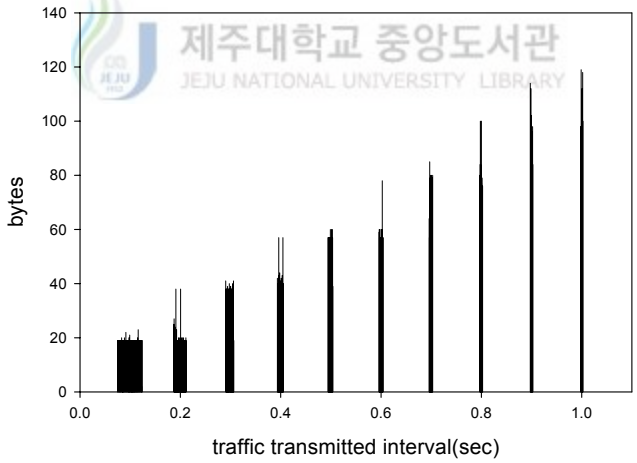
5. 대역폭 계산

트래픽의 측정시간간격 (Δt)에 대한 의존성을 조사하기 위해서 비디오 신호와 오디오 신호를 각각 10(0.01, 0.02, 0.03, ..., 0.09, 0.1sec) 단계로 나누어서 조사하였다. 비디오 신호의 최고셀율은 620.41Mbps, 최저셀율은 526.43Mbps이고, 평균셀율은 559.15Mbps이며, 오디오 신호의 최고셀율은 4.62Mbps, 최저셀율은 4.14Mbps이고, 평균셀율은 4.32Mbps이다.

그림 6은 측시간에 따른 트래픽의 전송량을 그림으로 나타낸 것이다(측정시간 10초). 비디오 트래픽과 오디오 트래픽의 최고율은 측정시간간격(Δt)에 의존함을 알 수 있다.



(a) video



(b) audio

Fig. 6. The transferring traffic by different time scale.

표 1은 비디오 트래픽과 오디오 트래픽의 초당 평균 전송량을 나타낸 것이다. 비디오 트래픽의 경우, 시간 간격(Δt)이 0.1초일 때, 1초당 전송되는 평균 셀은 1413.10cell 이고, 전송 속도는 (1413.10cell/sec) × (53byte/cell) × (8bit/byte) = 599.15Kbps 이다. 측정 시간이 0.1초부터 1초 동안 10단계(0.1초, 0.2초, ..., 0.9초, 1초)로 나누어서 전송량을 측정하여 정리하였다.

초당 평균 전송률은 비디오 트래픽의 경우 최고 620.41Kbps(Δt=0.5초)이고, 최저 526.43Kbps(Δt=0.7초)이며. 평균전송률은 564.05Kbps 이고, 오디오 트래픽의 경우 최고 4.62Kbps(Δt=0.2초)이고, 최저 4.14Kbps(Δt=0.7초)이며. 평균 전송률은 4.32Kbps이다.

그림 7은 측정 시간에 따른 평균 전송률을 나타낸 것이다. 비디오 트래픽과 오디오 트래픽의 평균 전송률은 측정 시간 간격(Δt)에 선형적으로 나타난다. 따라서 트래픽의 평균 전송률과 측정 시간 간격(Δt)과의 상호 의존성이 있음을 알 수 있다.

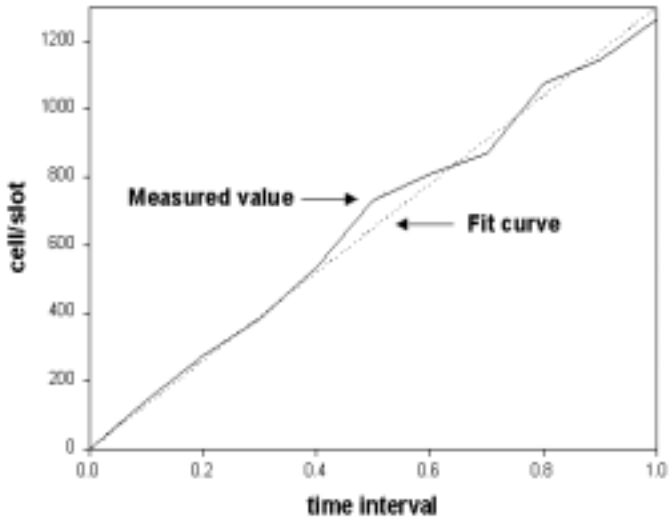
Table 1. Mean rate measurement per different time scale.

(a) video

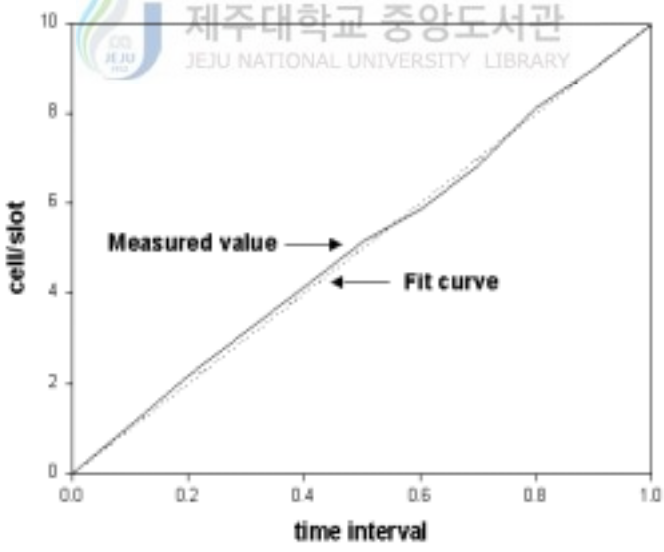
Δt	cell/slots	cells/sec	Kbps
0.1	141.31	1413.10	599.15
0.2	272.27	1361.35	577.21
0.3	381.37	1271.23	539.00
0.4	531.90	1329.75	563.81
0.5	731.62	1463.24	620.41
0.6	806.45	1344.08	569.89
0.7	869.10	1241.57	526.43
0.8	1074.19	1342.74	569.32
0.9	1144.49	1271.66	539.18
1.0	1264.41	1264.41	536.11

(b) audio

Δt	cell/slots	cells/sec	Kbps
0.1	1.07	10.07	4.25
0.2	2.18	10.90	4.62
0.3	3.16	10.53	4.46
0.4	4.14	10.35	4.39
0.5	5.16	10.32	4.38
0.6	5.87	9.78	4.15
0.7	6.84	9.77	4.14
0.8	8.13	10.16	4.31
0.9	9.00	10.00	4.24
1.0	9.96	9.96	4.22



(a) video

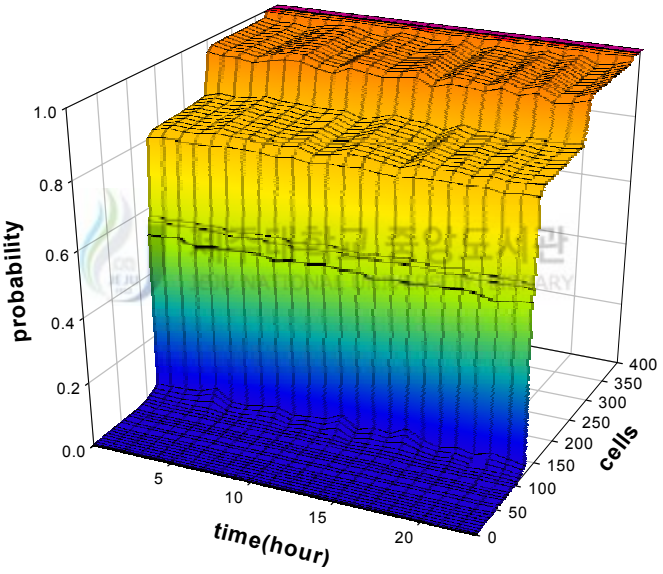


(b) audio

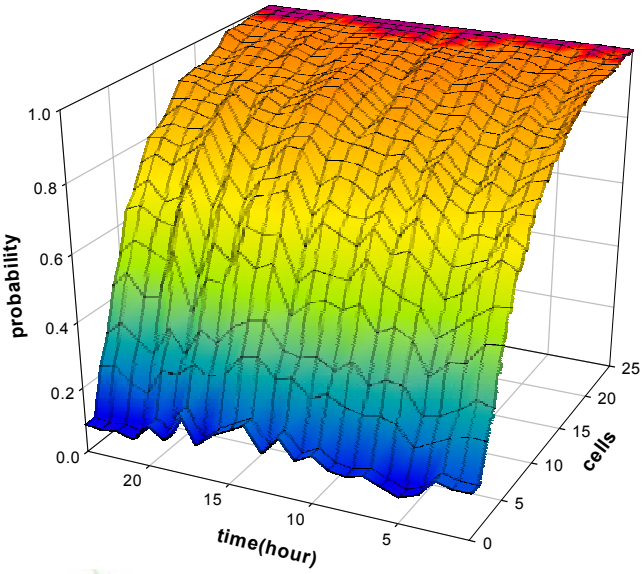
Fig. 7. Mean rate measurement per time scale.

6. 시간대별 트래픽의 측정

그림 8은 비디오 트래픽과 오디오 트래픽을 하루동안 1시간 간격으로 24번 측정하여 셀 전송률을 확률 밀도 함수와 시간대별의 상관 관계를 그래프로 나타낸 것이다. 하루동안 측정된 비디오 트래픽과 오디오 트래픽의 확률 밀도 함수의 차이가 거의 없음을 알 수 있다.



(a) video

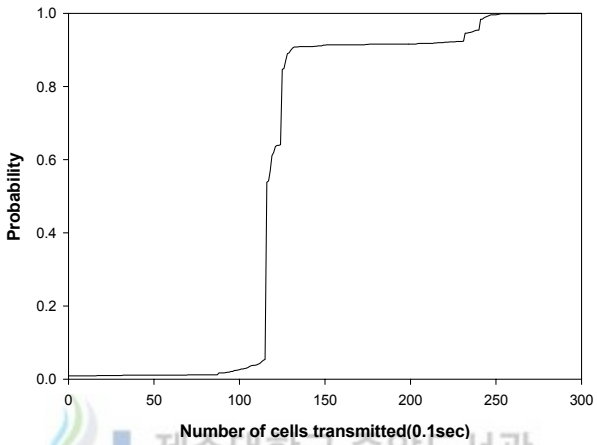


(b) audio

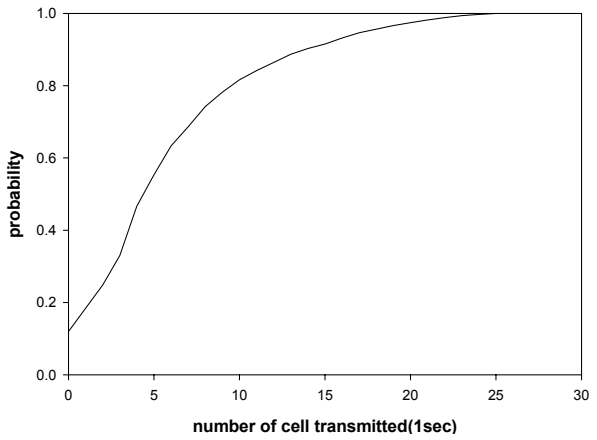
Fig. 8. Measurement of traffic every hour.

7. 측정 트래픽의 CDF 형성

그림 9는 측정된 제주대학교 트래픽으로부터 평균적인 CDF 특성을 나타낸 것이다.



(a) video



(b) audio

Fig. 9. The CDF of measured traffic.

III. 모델의 형성 및 시뮬레이션

1. 모델의 구조 및 형성

측정한 데이터들에서 트래픽의 특성을 가지고 트래픽 파라미터들을 계산한다. ATM 모델에 의해서 측정 데이타의 누적 분포들이 입증된다. 그리고 이것은 ATM 네트워크를 관리하고 제어하기 위한 유용한 자료가 된다.

트래픽 발생 함수(TGF)를 생성하기 위하여, 측정된 트래픽에서 표준 확률 밀도 함수를 생성한다. 생성된 확률 밀도 함수를 최소 자승 추정 방식(Marquardt, 1963)에 의하여 최적의 트래픽 발생 함수(TGF)를 생성한다. 측정된 비디오 트래픽과 오디오 트래픽의 최소 자승 추정 방식(Marquardt, 1963)에 의하여 생성된 함수는 exponential rise to max 함수의 한 종류이다.

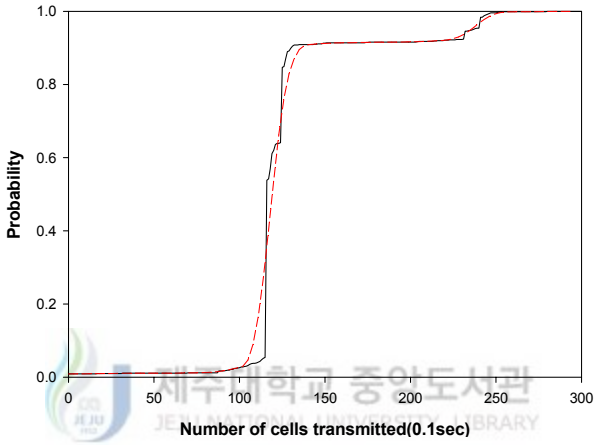
식 3.1은 비디오 트래픽과 오디오 트래픽을 발생하기 위해서 최소 자승 추정 방식(Marquardt, 1963)에 의해 생성된 트래픽 발생 함수이며, 이 방정식 역시 측정된 비디오 트래픽과 오디오 트래픽에 대한 분석을 기반으로 하고 있다.

$$v(t) = a(1 - \exp(-bx))^c \tag{3.1}$$

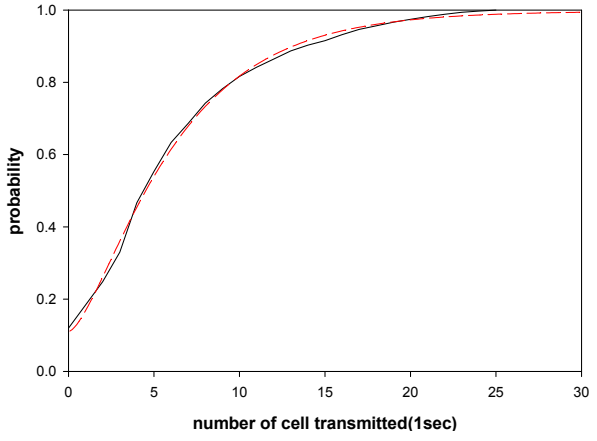
측정된 비디오 트래픽의 표준 그래프에서 얻어낸 값은 a = 0.951826, b = 0.158299, c =9.414874이며, 오디오 트래픽의 표준 그래프에서 얻어낸 값은 a = 0.886829, b = 0.203531, c = 1.618960이다.

2. 성능평가

다음은 제주대학교 ATM 트래픽과 함수 모델에 의해서 생성된 트래픽과의 비교를 해 보았다. 그림 10은 비디오 트래픽과 오디오 트래픽의 확률 밀도 함수와 표준 그래프를 상호 비교한 것이다.



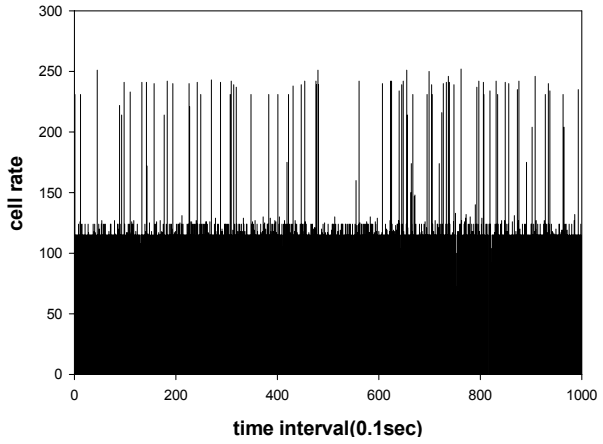
(a)video



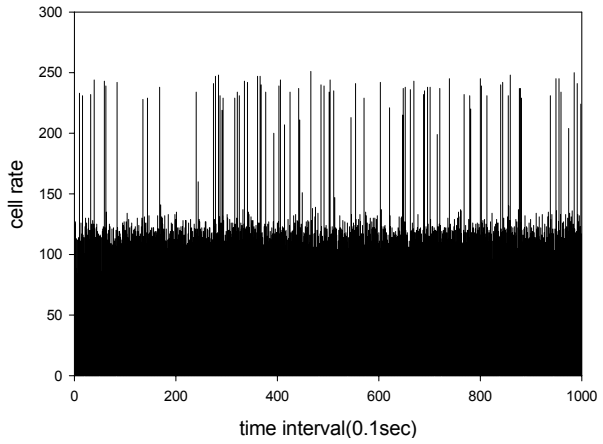
(b) audio

Fig. 10. Comparison of measuared traffic CDF and generated traffic CDF.

그림 11은 비디오 트래픽의 경우, 실제 측정 트래픽과 TGF에 의해 발생시킨 시뮬레이션 트래픽을 비교하여 나타내었다.



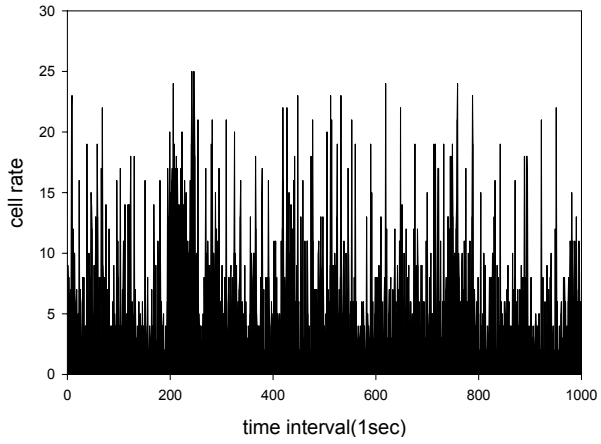
(a) measured video traffic



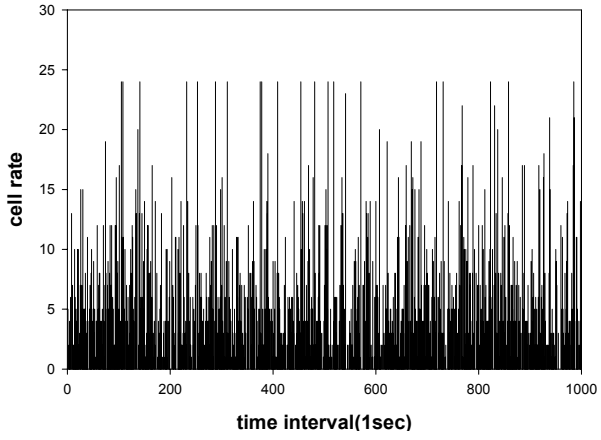
(b) simulated video traffic

Fig 11. Comparison video traffic.

그림 12는 오디오 트래픽의 경우, 실제 측정 트래픽과 TGF에 의해 발생시킨 시뮬레이션 트래픽을 비교하여 나타내었다.



(a) measured audio traffic



(b) simulated audio traffic

Fig 12. Comparison Audio traffic.

표 2는 비디오 트래픽과 오디오 트래픽의 측정치와 발생 함수에 의해 발생된 트래픽과의 차이를 나타내었다. 비디오 트래픽의 경우, 측정 트래픽과 트래픽 발생 함수에 의해 생성된 트래픽과의 평균치의 차이는 0.09이고, 표준 편차의 차이는 2.28이다. 그리고 오디오 트래픽의 경우, 측정 트래픽과 트래픽 발생 함수에 의해 생성된 트래픽과의 평균치의 차이는 0.44이고, 표준 편차의 차이는 0.27이다. 따라서 비디오, 오디오 측정 트래픽과 발생함수에 의해서 생성된 트래픽과의 차이가 거의 없음을 알 수 있다.

Table 2. The comparison of measured traffic and generated traffic.
(a: measured traffic, b: generated traffic)

	Video	Audio
$Ea(x)=\frac{1}{n_a}\sum_{\alpha=1}^{n_a} Xa$	146.06	6.35
$E_b(x)=\frac{1}{n_b}\sum_{b=1}^{nb} X_b$	145.97	5.91
$\lvert Ea(x)-E_b(x)\rvert$	0.09	0.44
$S_a=(\frac{\sum_{a=1}^{n_a} X_a-n_a\times E_a(x)}{n_a-1})^{0.5}$	161.19	5.83
$Sb=(\frac{\sum_{a=1}^{n_b} X_b-n_b\times E_b(x)}{n_b-1})^{0.5}$	161.88	5.30
$S=\sqrt{\frac{Sa^2}{n_a}+\frac{Sb^2}{n_b}}$	2.28	0.27

Ⅴ. 결 론

본 논문에서는 제주대학교 ATM 비디오, 오디오 트래픽을 측정하여, 측정된 트래픽의 특성을 조사하고, 새로운 트래픽 생성 함수를 제안하였다.

측정 프로그램은 Visual C++로 프로그래밍하여, 트래픽의 측정시간(Δt)과 하루 24시간동안에 전송되는 트래픽을 측정하였다.

제주대학교 ATM 비디오 및 오디오 트래픽의 측정시간에 따른 트래픽의 변화는 측정 시간 간격(Δt)에 의존한다.

그리고 측정 트래픽과 측정 데이타로부터 생성된 트래픽 모델에 의해 발생된 트래픽과의 차이를 비교하였다. 초당 평균 전송량의 차이는 비디오 트래픽의 경우 1% 이하, 오디오 트래픽인 경우 7% 이하의 차이가 있다. 또한 표준편차에 의하여 구해진 차이는 비디오의 경우 2.28, 오디오인 경우 0.27로서 차이가 거의 없음을 알 수 있었다.

따라서, 생성된 트래픽 생성 함수는 제주대학교의 ATM 네트워크의 해석과 관리 및 제주대학교 ATM 네트워크에서 최적의 폭주제어 알고리즘 개발, 통신프로토콜의 개선 및 스위칭 아키텍처의 개발에 쓰일 것으로 사료된다.

참 고 문 헌

Abdelnaser Adas, 1997. 7, "Traffic Models in Brandband Network", IEEE Communications Manazine

Othmar Kyas, 1997, ATM Network, 홍릉과학출판사

ATM망에서의 트래픽 제어에 관한 연구, 1999. 8, 제주대학교 정보통신부

Black, Uyless D., 1995, ATM, Vol. Ⅰ : Foundation for broadband networks, Prentice Hall

Black, Uyless D., 1995, ATM, Vol. Ⅱ : Signaling in broadband networks , Prentice Hall

Black, Uyless D., 1995, ATM, Vol. Ⅲ : Internetworking with ATM,, Prentice Hall

Carrie Carter, David Freeman, 1996, "Test traffic generation equipment and algorithms evaluating ATM networks", Computer Communications 19, pp962~971

D.S. Holtainger, H. G. Perros, and A. A. Nilsson, July 1993. "Analysis of traffic measurements in the VISTnet Gigabit networking tested," tech. rep. Center for Communications and signal processing

Ioannis Stavrakakis, 1996, " Characterization and multiplexing of correlated traffic from sources with different time constants", Computer Communications '96, pp1112~1123

Jorge-A., sANCHEZ-p.,Gerd Keiser, Iakovos S. Venieris, Emmanuel N. Protonotraios, 1996, "On the Definition of ATM Traffic Descriptorsfor LAN Sources", processings of the 1996 IEEE International conference on communications vol.3

Lilly Cheng and Herman D. Hughes, June 1995, "An ATM traffic model based on empirical traffic measurement", IEEE International Conference on Communications'95.

Lilly Cheng and Herman D. Hughes, June 1995, "A connection admission control algorithm based on empirical traffic measurements", in processing of IEEE International Conference on Communications'95.

Lilly Cheng and Herman D. Hughes, March 1995, "Testing application traffic in an atm tested" in 7th IEEE Workshop on Local and Metropolitan Area Networks.

Qiang Ren, G. Ramamurthy, February 2000, "A real-time dynamic connection admission controller based on traffic modeling, measurement, and Fuzzy logic control", IEEE iournal on selected areas in communications. vol. 18. no.2.

Ronald G. Addie, Moshe Zukerman and Tim Neame, 1995, "Ftactal Traffic: Measurement, Modelling and Performance Evaluation", processing of the IEEE Infocom '95 The conference on computer communications, vol. 3.

Victor S. Frost and Benjamin Nelamed, March 1994. "Traffic Modeling for Telecommunications Networks", IEEE Commuications Magazine, pp.70~81.

홈페이지 www.cinewel.com, 영화 "달마야 놀자" 실시간 측정

홈페이지 www.imbc.com, 음악 박효신의 "좋은사람"을 실시간 측정

김상철, 고성택, 2000, ATM망에서의 효율적인 문턱값 기반 호 수락 제어“, 한국신호처리・시스템학회 학회지, Vol. 1, pp.33-36.

김지관, 1998, 표준 ATM, 교보문고

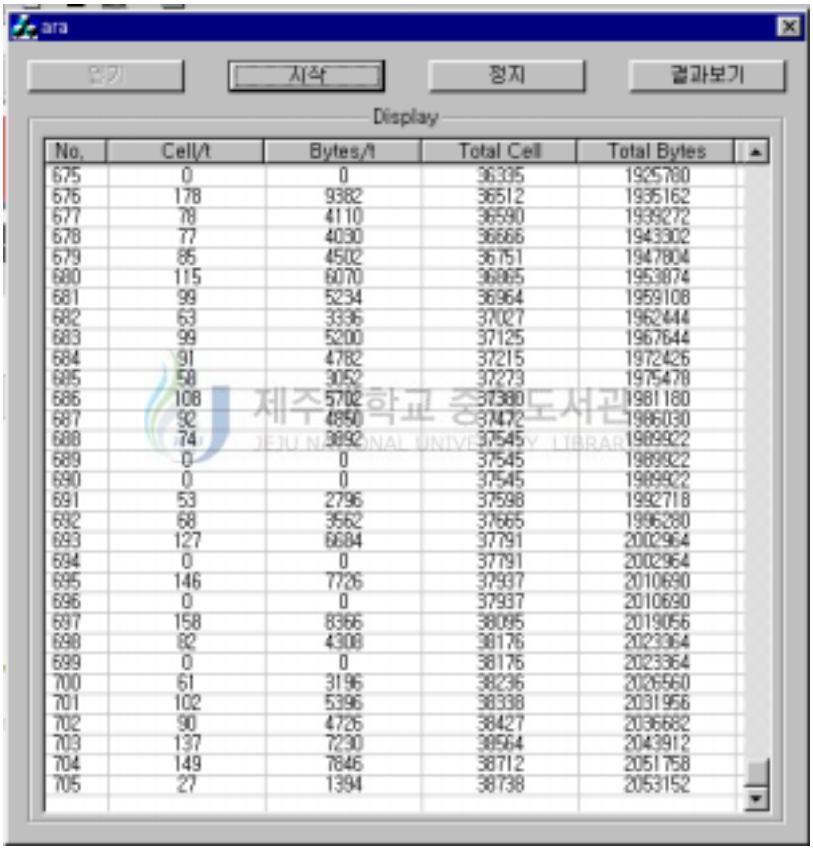
심덕주, 최상훈, 곽경섭, 1999, “ATM망의 가상경로 상에서 효과적인 대역할당 알고리즘 역구”, 한국통신학회 논문지 pp.2065-2070

이문호, 장성현, 1999, “ATM망의 가상 경로를 이용한 효율적인 호 수락 제어”, 한국통신학회 논문지 ‘96-11, Vol. 21, pp. 2897-2907

임주환, 성단근, 한치문, 김영선, 1997, ATM 교환, 홍릉과학출판사

부 록

1. 측정 프로그램 실행시 결과 윈도우창



2. 측정 프로그램

```
void CAraDlg::OnOpen()
{
	CString str;
	PCHAR SYSNAME = "\\\\.\\SAHIT";
	if(!hDevice)
	{
		hDevice = CreateFile(SYSNAME,

GENERIC_READ|GENERIC_WRITE,

FILE_SHARE_READ|FILE_SHARE_WRITE, 0, OPEN_EXISTING, FLAG_OVERLAPPED, 0);
		if(hDevice == INVALID_HANDLE_VALUE)
		{
			str.Format("%d",GetLastError());
			AfxMessageBox(str);
		}
		else
		{
			m_bIsOpen = TRUE;
			m_bViewResult = FALSE;
			CButton* pB = (CButton*)GetDlgItem(IDC_OPEN);
			pB->EnableWindow(FALSE);
			AfxMessageBox("OPEN SUCCESS");
		}
	}
	else
	{
		AfxMessageBox("Already OPEN!");
		return;
	}
}

#include "filename.h"

void CAraDlg::OnInit()
{
```

```
CString str, Next;
ULONG Result;
STATISTIC Data;
PULONG returnbyte =0;
OVERLAPPED Overlapped;
CFileName fName;

m_List.DeleteAllItems();

if(m_bIsInit)
{
        AfxMessageBox("processing");
        return;
}

if((fName.DoModal() == IDOK) || m_bIsOpen == TRUE || m_bIsInit == FALSE)
{
        if(fName.m_FileName.IsEmpty())
        {
                AfxMessageBox("input filename");
                return;
        }
        else
        {
                memset(&Data, 0, sizeof(STATISTIC));
                memset(&Overlapped, 0, sizeof(OVERLAPPED));
                Overlapped.hEvent = CreateEvent(NULL, FALSE, FALSE, NULL);
                Result = DeviceIoControl(hDevice, IOCTL_STATISTIC_START, 0,
                0, Data, sizeof(STATISTIC), returnbyte, &Overlapped);

                if(Result == ERROR_IO_PENDING)
                {
                Result = GetOverlappedResult(hDevice, &Overlapped, turnbyte,
                        TRUE);
                }

                m_bIsInit = TRUE;
                m_bIsStop = FALSE;
                m_bViewResult = FALSE;
```

```
                    SetTimer(1, 1000, 0);
                    m_Index = 0;
                    m_TempPacketNum = 0;
                    m_TempBytesNum = 0;

                    str.Format(fName.m_FileName);
                    m_File.Open(str, CFile::modeCreate | CFile::modeWrite);
                    str.Format("NO.\t cell per sec \tnumbyte\t Total cells\t
                            totalbytes;");
                    m_File.Write(str, str.GetLength());
                    str.Format("\r\n;\r\n;\r\n");
                    m_File.Write(str, str.GetLength());
                }
        }
}

void CAraDlg::OnReceive()
{
        int a = 0;
        float b = 0;
        float c = 0;
        CString str;
        ULONG Result;
        STATISTIC Data;
        PULONG returnbyte =0;
        CSize Size(0,0);
        OVERLAPPED Overlapped;

        if(m_bIsInit)
        {
                memset(&Overlapped, 0, sizeof(OVERLAPPED));

                Overlapped.hEvent = CreateEvent(NULL, FALSE, FALSE, NULL);

                Result = DeviceIoControl(hDevice, IOCTL_STATISTIC_GET, 0, 0, &Data,
                                izeof(STATISTIC), returnbyte, &Overlapped);

                if(Result == ERROR_IO_PENDING)
                {
```

```
                Result = GetOverlappedResult(hDevice, &Overlapped, returnbyte,
                        TRUE);
        }

        str.Format("%d", m_Index+1);
        AddItem(m_Index, 0, str);

        m_File.Write(str, str.GetLength());
        str.Format("\t       ");
        m_File.Write(str, str.GetLength());

        a = (Data.RcvTotalBytes - m_TempBytesNum);
        b = a/53.0;
        c = a/53;

        if(b-c > 0)
        {
                c = c + 1;
        }
        str.Format("%d", Data.RcvTotalPackets - m_TempPacketNum);
        str.Format("%d", (int)c);
        AddItem(m_Index, 1, str);

        m_File.Write(str, str.GetLength());
        str.Format("\t\t         ");
        m_File.Write(str, str.GetLength());

str.Format("%d", Data.RcvTotalBytes - m_TempBytesNum);
        AddItem(m_Index, 2, str);

        m_File.Write(str, str.GetLength());
        str.Format("    \t\t");
        m_File.Write(str, str.GetLength());

str.Format("%d", Data.RcvTotalPackets);
        str.Format("%d", Data.RcvTotalBytes/53);
        AddItem(m_Index, 3, str);
```

```
                m_File.Write(str, str.GetLength());
                str.Format("\t          ");
                m_File.Write(str, str.GetLength());

                str.Format("%d", Data.RcvTotalBytes);
                AddItem(m_Index, 4, str);

                m_File.Write(str, str.GetLength());
                str.Format("\t;\r\n");
                m_File.Write(str, str.GetLength());

                m_TempPacketNum = Data.RcvTotalPackets;
                m_TempBytesNum = Data.RcvTotalBytes;

                Size.cy = m_Index++;
                m_List.Scroll(Size);
        }
}

void CAraDlg::OnStop()
{
        CString str;
        if(!m_bIsStop)
        {
                KillTimer(1);
                m_File.SeekToEnd();
                m_File.Write("",0);
                m_File.Close();
                m_bIsInit = FALSE;
                m_bViewResult = TRUE;
        }
        m_bIsStop = TRUE;
}

void CAraDlg::OnTimer(UINT nIDEvent)
{
        OnReceive();
```

```
	if(m_TimeCount > 240)
	{
		OnStop();
		AfxMessageBox("The End");
		return;
	}
	ReadData();
	m_TimeCount++;
	CDialog::OnTimer(nIDEvent);
}

void CAraDlg::OnClose()
{
	if(hDevice)
	{
		CloseHandle(hDevice);
	}
	CDialog::OnClose();
}
```



```
#include "ResultDlg.h"
void CAraDlg::OnResult()
{
	if(!m_bViewResult)
	{
		AfxMessageBox("cannot open");
		return;
	}
	CString str;
	CFileDialog FileDlg(TRUE);
	FileDlg.m_ofn.lpstrInitialDir = (LPSTR)("C:\\");
	FileDlg.m_ofn.lpstrFilter = (LPSTR)("*.*(allfile)");
	if(FileDlg.DoModal() == IDOK)
	{
		if(AnalyzeFile(FileDlg.m_ofn.lpstrFileTitle))
		{
			m_ResultDlg.DoModal();
		}
```

```
		}
}

#define PACKET_PER_TIME	1
#define BYTE_PER_TIME		2
#define TOTAL_PACKET		3
#define TOTAL_BYTE		4

BOOL CAraDlg::AnalyzeFile(CString FileName)
{
		UINT Static[200] = {0,};
		int Start = 0, itemp;
		int Value = 0;
		char *Data;
		char Line1[100];
		CString Line2;
		CString str, Temp;

		m_File.Open(FileName, CFile::modeRead);
		m_File.SeekToBegin();

		while(m_File.ReadString(Line2))
		{
			int length = Line2.GetLength();
			m_File.Seek(-(length+2), CFile::current);
			m_File.ReadString(Line1, 100);
			m_Offset = Line2.Find(";", m_Offset);

			if(m_Offset == -1)
			{
				break;
			}
			else
			{
				m_File.Seek(0, CFile::current);
				m_Offset = 0;
				if(Start++ >= 3)
				{
					Data = strtok(Line1, " ;\t");
					while((Data != NULL) && (m_SelectedData < 4))
```

```
                                        {
                                                m_SelectedData++;
                                                switch(m_SelectedData)
                                                {
                                                case PACKET_PER_TIME:
                                                        Data = strtok(NULL, " ;\t");
                                                        str.Format("%s", Data);
                                                        Value = atoi(str);

                                                        itemp = (Value-1) / 10;
                                                        Static[itemp] ++;

                                                        break;
                                                case BYTE_PER_TIME:
                                                        Data = strtok(NULL, " ;\t");
                                                        break;
                                                case TOTAL_PACKET:
                                                        Data = strtok(NULL, " ;\t");
                                                        break;
                                                case TOTAL_BYTE:
                                                        Data = strtok(NULL, " ;\t");
                                                        break;
                                                }
                                        }
                                        m_SelectedData = 0;
                                }
                        }
                        Line2.Empty();
                }
                m_ResultDlg.m_Result.Format("resuitnumcell\r\n\r\n\r\n");

                for(itemp=0; itemp <200; itemp ++)
                {
                        Temp.Format("%d.  %d  ~  %d  :  %d\r\n\r\n",  itemp+1,  itemp*10+1,
itemp*10+10,            Static[itemp] );
                        m_ResultDlg.m_Result += Temp;
                }

                m_File.Close();
                return TRUE;
```

```
}

void CAraDlg::ReadData()
{
	UCHAR pData[sizeof(MONITORINGDATA)*100] = {0,};
	MONITORINGDATA MonitoringData;
	memset(&MonitoringData, 0, sizeof(MONITORINGDATA));
	ULONG returnbyte = 0;
	CString str;
	CSize Size(0,0);
	OVERLAPPED Overlapped;

	memset(&Overlapped, 0, sizeof(OVERLAPPED));
	Overlapped.hEvent = CreateEvent(NULL, FALSE, FALSE, NULL);

	ULONG Result = ReadFile((HANDLE)hDevice, pData, sizeof(MONITORINGDATA)*100,
					&returnbyte,  &Overlapped);
	ULONG Error = GetLastError();

	if(Error == ERROR_IO_PENDING)
	{
		Result = GetOverlappedResult(hDevice,
			&Overlapped,
			&returnbyte,
			TRUE);
	}
	if(returnbyte)
	{
		UINT BufferCount = 0;
		UINT ListCount = 0;
		UINT ByteSize = 0;

		BufferCount = returnbyte/sizeof(MONITORINGDATA);
		if(returnbyte > sizeof(MONITORINGDATA)*900)
		{
			AfxMessageBox("overflow data");
		}
		while(BufferCount)
```

```
{
        memcpy(&MonitoringData, pData+(ListCount*sizeof(MONITORINGDATA)),sizeof(MONITORINGDATA));

        str.Format("%d", m_Index+1);
        AddItem(m_Index, 0, str);
        m_File.Write(str, str.GetLength());
        str.Format("\t\t      ");
        m_File.Write(str, str.GetLength());

        if(MonitoringData.TimeStamp.Hour >= 15)
        {
                MonitoringData.TimeStamp.Day += 1;
        }

        MonitoringData.TimeStamp.Hour += 9;
        MonitoringData.TimeStamp.Hour = MonitoringData.TimeStamp.Hour%24;

        str.Format("%d/%d/%d/%d/%d/%d/%d",MonitoringData.TimeStamp.Year,MonitoringData.TimeStamp.Month, MonitoringData.TimeStamp.Day, MonitoringData.TimeStamp.Hour, MonitoringData.TimeStamp.Minute, MonitoringData.TimeStamp.Second, MonitoringData.TimeStamp.Milliseconds);
        AddItem(m_Index, 1, str);
        m_File.Write(str, str.GetLength());
        str.Format("\t\t      ");
        m_File.Write(str, str.GetLength());

        str.Format("%x", MonitoringData.SystemTimeStamp);
        AddItem(m_Index, 2, str);
        m_File.Write(str, str.GetLength());
        str.Format("\t\t      ");
        m_File.Write(str, str.GetLength());

        str.Format("%d",MonitoringData.CellCounts);
        AddItem(m_Index, 3, str);
        m_File.Write(str, str.GetLength());
```

```
                    str.Format("\t;\r\n");
                    m_File.Write(str, str.GetLength());

                    BufferCount--;
                    ListCount++;

                    Size.cy = m_Index++;
                    m_List.Scroll(Size);
                }
            }
}


void CAraDlg::ReadData()
{
            UCHAR pData[sizeof(MONITORINGDATA)*100] = {0,};
            MONITORINGDATA MonitoringData;
            memset(&MonitoringData, 0, sizeof(MONITORINGDATA));
            ULONG returnbyte = 0;
            CString str;
            CSize Size(0,0);
            OVERLAPPED Overlapped;

            memset(&Overlapped, 0, sizeof(OVERLAPPED));
            Overlapped.hEvent = CreateEvent(NULL, FALSE, FALSE, NULL);

            ULONG Result = ReadFile((HANDLE)hDevice, pData, sizeof(MONITORINGDATA)*100,
                        &returnbyte,  &Overlapped);
            ULONG Error = GetLastError();

            UINT OutputCellCount = 0;
            CSTIMEFIELDS OutputTimeStamp = {0, };

            if(Error == ERROR_IO_PENDING)
            {
                Result = GetOverlappedResult(hDevice,
                        &Overlapped,
                        &returnbyte,
```

```
                    TRUE);
        }
        if(returnbyte)
        {
                UINT BufferCount = 0;
                UINT ListCount = 0;
                UINT ByteSize = 0;

                BufferCount = returnbyte/sizeof(MONITORINGDATA);
                if(returnbyte > sizeof(MONITORINGDATA)*900)
                {
                        AfxMessageBox("overflow data");
                }
                while(BufferCount)
                {
                        memcpy(&MonitoringData,
pData+(ListCount*sizeof(MONITORINGDATA)),sizeof(MONITORINGDATA));

                        if(MonitoringData.TimeStamp.Hour >= 15)
                        {
                                MonitoringData.TimeStamp.Day += 1;
                        }
                        MonitoringData.TimeStamp.Hour += 9;
                        MonitoringData.TimeStamp.Hour =
MonitoringData.TimeStamp.Hour%24;

                m_SourceTime = MonitoringData.TimeStamp;

                if((m_CompareTime.Year != 0) && (m_SourceTime.Year != 0) )
                {
                        if(m_SourceTime.Hour < m_CompareTime.Hour)
                        {
                                m_TempTime.Hour = (m_SourceTime.Hour + (24 -
                                        m_CompareTime.Hour));
                                m_SourceTime.Minute += (60 * m_TempTime.Hour);

                                m_SourceTime.Second += (60 * (m_SourceTime.Minute
-                                       m_CompareTime.Minute));
```

```
                m_SourceTime.Milliseconds += (1000
*(m_SourceTime.Second -
m_CompareTime.Second));
        }
        else if(m_SourceTime.Minute < m_CompareTime.Minute)
        {
                m_TempTime.Minute = (m_SourceTime.Minute + (60 -
                        m_CompareTime.Minute));
                m_SourceTime.Second += (60 * m_TempTime.Minute);
                m_SourceTime.Milliseconds += (1000 *(m_SourceTime.
                        Second- m_CompareTime.Second));
        }
        else if(m_SourceTime.Second < m_CompareTime.Second)
        {
                m_TempTime.Second = (m_SourceTime.Second + (60 -
                        m_CompareTime.Second));
                m_SourceTime.Milliseconds += (1000 *
                        m_TempTime.Second);
        }
        else if(m_SourceTime.Second > m_CompareTime.Second)
        {
                m_TempTime.Second = (m_SourceTime.Second -
                        m_CompareTime.Second);
                m_SourceTime.Milliseconds += (1000 * m_TempTime.
                        Second);
                m_SourceTime.Second -= m_TempTime.Second;
        }
        else if(m_SourceTime.Minute > m_CompareTime.Minute)
        {
                m_TempTime.Minute = (m_SourceTime.Minute -
                        m_CompareTime.Minute);
                m_SourceTime.Second += (60 * m_TempTime.Minute);
                m_SourceTime.Minute -= m_TempTime.Minute;
                m_TempTime.Second = (m_SourceTime.Second -
                        m_CompareTime.Second);
                m_SourceTime.Milliseconds += (1000 *
                        m_TempTime.Second);
                m_SourceTime.Second -= m_TempTime.Second;
```

```
                    }
                    else if(m_SourceTime.Hour > m_CompareTime.Hour)
                    {
                              m_TempTime.Hour = (m_SourceTime.Hour -
                                        m_CompareTime.Hour);
                              m_SourceTime.Minute += (60 * m_TempTime.Hour);
                              m_SourceTime.Hour -= m_TempTime.Hour;
                              m_TempTime.Minute = (m_SourceTime.Minute -
                                        m_CompareTime.Minute);
                              m_SourceTime.Second += (60 * m_TempTime.Minute);
                              m_SourceTime.Minute -= m_TempTime.Minute;
                              m_TempTime.Second = (m_SourceTime.Second -
                                        m_CompareTime.Second);
                              m_SourceTime.Milliseconds += (1000 *
                                        m_TempTime.Second);
                              m_SourceTime.Second -= m_TempTime.Second;
                    }
          }

if(m_CompareTime.Year == 0)
{
          m_OutputCellCounts = MonitoringData.CellCounts;
          m_CompareTime = MonitoringData.TimeStamp;
          BufferCount--;
          ListCount++;
          continue;
}

else if((m_SourceTime.Milliseconds - m_CompareTime.Milliseconds) <= 100)
{
          m_OutputCellCounts += MonitoringData.CellCounts;
          BufferCount--;
          ListCount++;
          continue;
}

else if((m_SourceTime.Milliseconds - m_CompareTime.Milliseconds) > 100)
{
```

```
if(m_OutputCellCounts)
{
        str.Format("%d", m_Index+1);
        AddItem(m_Index, 0, str);
        m_File.Write(str, str.GetLength());
        str.Format("\t\t        ");
        m_File.Write(str, str.GetLength());
```

str.Format("%d/%d/%d/%d/%d/%d/%d",MonitoringData.TimeStamp.Year,MonitoringData.TimeStamp.Month, MonitoringData. TimeStamp. Day, MonitoringData. TimeStamp. Hour, MonitoringData. TimeStamp. Minute, MonitoringData. TimeStamp. Second, MonitoringData. TimeStamp. Milliseconds);

```
        AddItem(m_Index, 1, str);
        m_File.Write(str, str.GetLength());
        str.Format("\t\t        ");
        m_File.Write(str, str.GetLength());

        str.Format("%d",m_OutputCellCounts);
        AddItem(m_Index, 2, str);
        m_File.Write(str, str.GetLength());
        str.Format("\t;\r\n");
        m_File.Write(str, str.GetLength());

        BufferCount--;
        ListCount++;

        Size.cy = m_Index++;
        m_List.Scroll(Size);
        ////////////출력////////////

        m_OutputCellCounts = MonitoringData.CellCounts;

        CSTIMEFIELDS m_NullTimeStamp = {0, };
        UINT LoopCount = 0;
        m_NullTimeStamp.Milliseconds = (m_SourceTime.Milliseconds -
                100)-m_CompareTime.Milliseconds;
```

```
                while(m_NullTimeStamp.Milliseconds >= 100)
                {
                        str.Format("%6d", m_Index+1);
                        AddItem(m_Index, 0, str);
                        m_File.Write(str, str.GetLength());
                        str.Format("\t\t          ");
                        m_File.Write(str, str.GetLength());

                        str.Format("0000/00/00/00/00/00/000");
                        AddItem(m_Index, 1, str);
                        m_File.Write(str, str.GetLength());
                        str.Format("\t\t          ");
                        m_File.Write(str, str.GetLength());

                        str.Format("000");
                        AddItem(m_Index, 2, str);
                        m_File.Write(str, str.GetLength());
                        str.Format("\t;\r\n");
                        m_File.Write(str, str.GetLength());

                        m_NullTimeStamp.Milliseconds -= 100;

                        Size.cy = m_Index++;
                        m_List.Scroll(Size);
                        }
}
else
{
                CSTIMEFIELDS m_NullTimeStamp = {0, };
                UINT LoopCount = 0;
                m_NullTimeStamp.Milliseconds = (m_SourceTime.Milliseconds -
                        100)-m_CompareTime.Milliseconds;
                while(m_NullTimeStamp.Milliseconds >= 100)
                {
                        str.Format("%6d", m_Index+1);
                        AddItem(m_Index, 0, str);
                        m_File.Write(str, str.GetLength());
```

```
                                        str.Format("\t\t          ");
                                        m_File.Write(str, str.GetLength());

                                        str.Format("0000/00/00/00/00/00/000");
                                        AddItem(m_Index, 1, str);
                                        m_File.Write(str, str.GetLength());
                                        str.Format("\t\t          ");
                                        m_File.Write(str, str.GetLength());

                                        str.Format("000");
                                        AddItem(m_Index, 2, str);
                                        m_File.Write(str, str.GetLength());
                                        str.Format("\t;\r\n");
                                        m_File.Write(str, str.GetLength());

                                        m_NullTimeStamp.Milliseconds -= 100;

                                        Size.cy = m_Index++;
                                        m_List.Scroll(Size);

                                        m_OutputCellCounts += MonitoringData.CellCounts;
                                        BufferCount--;
                                        ListCount++;
                                }
                        }
                        m_CompareTime = MonitoringData.TimeStamp;
                }
                        }
                }
}
*/

void CAraDlg::ReadData()
{
        UCHAR pData[sizeof(MONITORINGDATA)*1000] = {0,};
        MONITORINGDATA MonitoringData;
        memset(&MonitoringData, 0, sizeof(MONITORINGDATA));
        ULONG returnbyte = 0;
```

```
CString str;
CSize Size(0,0);
OVERLAPPED Overlapped;

memset(&Overlapped, 0, sizeof(OVERLAPPED));
Overlapped.hEvent = CreateEvent(NULL, FALSE, FALSE, NULL);

ULONG Result = ReadFile((HANDLE)hDevice, pData,
        sizeof(MONITORINGDATA)*1000, &returnbyte,  &Overlapped);
ULONG Error = GetLastError();

UINT OutputCellCount = 0;
CSTIMEFIELDS OutputTimeStamp = {0, };

if(Error == ERROR_IO_PENDING)
{
        Result = GetOverlappedResult(hDevice,
                &Overlapped,
                &returnbyte,
                TRUE);
}
if(returnbyte)
{
        UINT BufferCount = 0;
        UINT ListCount = 0;
        UINT ByteSize = 0;

        BufferCount = returnbyte/sizeof(MONITORINGDATA);
        if(returnbyte > sizeof(MONITORINGDATA)*900)
        {
                AfxMessageBox("overflow data");
        }
        while(BufferCount)
        {
                memcpy(&MonitoringData,
                data+(ListCount*sizeof(MONITORINGDATA)),
                sizeof(MONITORINGDATA);

                if(MonitoringData.TimeStamp.Hour >= 15)
```

```
{
        MonitoringData.TimeStamp.Day += 1;
}
MonitoringData.TimeStamp.Hour += 9;
MonitoringData.TimeStamp.Hour = MonitoringData.TimeStamp.Hour%24;

m_SourceTime = MonitoringData.TimeStamp;

if((m_CompareTime.Year != 0) && (m_SourceTime.Year != 0) )
{
        if(m_SourceTime.Hour < m_CompareTime.Hour)
        {
        m_TempTime.Hour = (m_SourceTime.Hour + (24 - m_CompareTime.Hour));
                m_SourceTime.Minute += (60 * m_TempTime.Hour);
                m_SourceTime.Second += (60 * (m_SourceTime.Minute -
                                m_CompareTime.Minute));
                m_SourceTime.Milliseconds += (1000 * (m_SourceTime.Second -
                                m_CompareTime.Second));
        }
        else if(m_SourceTime.Minute < m_CompareTime.Minute)
        {
                m_TempTime.Minute = (m_SourceTime.Minute + (60 -
                                m_CompareTime.Minute));
                m_SourceTime.Second += (60 * m_TempTime.Minute);

                m_SourceTime.Milliseconds += (1000 * (m_SourceTime.Second -
                                m_CompareTime.Second));
        }
        else if(m_SourceTime.Second < m_CompareTime.Second)
        {
                m_TempTime.Second = (m_SourceTime.Second + (60 -
                                m_CompareTime.Second));
                m_SourceTime.Milliseconds += (1000 * m_TempTime.Second);
        }

        else if(m_SourceTime.Second > m_CompareTime.Second)
        {
                m_TempTime.Second = (m_SourceTime.Second
                        -m_CompareTime.Second);
```

```
                m_SourceTime.Milliseconds += (1000 * m_TempTime.Second);
                m_SourceTime.Second -= m_TempTime.Second;
        }
        else if(m_SourceTime.Minute > m_CompareTime.Minute)
        {
                m_TempTime.Minute = (m_SourceTime.Minute
                        -m_CompareTime.Minute);
                m_SourceTime.Second += (60 * m_TempTime.Minute);
                m_SourceTime.Minute -= m_TempTime.Minute;
                m_TempTime.Second = (m_SourceTime.Second
                        -m_CompareTime.Second);
                m_SourceTime.Milliseconds += (1000 * m_TempTime.Second);
                m_SourceTime.Second -= m_TempTime.Second;
        }
        else if(m_SourceTime.Hour > m_CompareTime.Hour)
        {
                m_TempTime.Hour = (m_SourceTime.Hour -
                        m_CompareTime.Hour);
                m_SourceTime.Minute += (60 * m_TempTime.Hour);
                m_SourceTime.Hour -= m_TempTime.Hour;

                m_TempTime.Minute = (m_SourceTime.Minute
                        -_CompareTime.Minute);
                m_SourceTime.Second += (60 * m_TempTime.Minute);
                m_SourceTime.Minute -= m_TempTime.Minute;

                m_TempTime.Second = (m_SourceTime.Second
                        -m_CompareTime.Second);
                m_SourceTime.Milliseconds += (1000 * m_TempTime.Second);
                m_SourceTime.Second -= m_TempTime.Second;
        }
}

if(m_CompareTime.Year == 0)
                {
        m_OutputCellCounts = MonitoringData.CellCounts;
        m_CompareTime = MonitoringData.TimeStamp;
        BufferCount--;
```

```
                ListCount++;
                continue;
        }

        else if((m_SourceTime.Milliseconds - m_CompareTime.Milliseconds) <= 10)
        {
                m_OutputCellCounts += MonitoringData.CellCounts;
                BufferCount--;
                ListCount++;
                continue;
        }

        else if((m_SourceTime.Milliseconds - m_CompareTime.Milliseconds) > 10)
        {
                if(m_OutputCellCounts)
                {
                        str.Format("%6d", m_Index+1);
//                      AddItem(m_Index, 0, str);
                        m_File.Write(str, str.GetLength());
                        str.Format("\t\t ");
                        m_File.Write(str, str.GetLength());

str.Format("%4d/%2d/%2d/%2d/%2d/%2d/%3d",MonitoringData.TimeStamp.Year,MonitoringData.TimeStamp.Month, MonitoringData .TimeStamp. Day, MonitoringData. TimeStamp. Hour, MonitoringData. TimeStamp.Minute, MonitoringData. TimeStamp.Second, MonitoringData.TimeStamp.Milliseconds);
                        AddItem(m_Index, 1, str);
                        m_File.Write(str, str.GetLength());
                        str.Format("\t\t ");
                        m_File.Write(str, str.GetLength());

                        str.Format("%3d",m_OutputCellCounts);
                        AddItem(m_Index, 2, str);
                        m_File.Write(str, str.GetLength());
                        str.Format("\t;\r\n");
                        m_File.Write(str, str.GetLength());
                        BufferCount--;
```

```
                ListCount++;

                Size.cy = m_Index++;
                m_List.Scroll(Size);

                m_OutputCellCounts = MonitoringData.CellCounts;

                CSTIMEFIELDS m_NullTimeStamp = {0, };
                UINT LoopCount = 0;
                m_NullTimeStamp.Milliseconds = (m_SourceTime.Milliseconds -
                10)-m_CompareTime.Milliseconds;
                while(m_NullTimeStamp.Milliseconds >= 10)
                {
                        str.Format("%6d", m_Index+1);
                        AddItem(m_Index, 0, str);
                        m_File.Write(str, str.GetLength());
                        str.Format("\t\t        ");
                        m_File.Write(str, str.GetLength());
                        str.Format("0000/00/00/00/00/00/000");
                        AddItem(m_Index, 1, str);
                        m_File.Write(str, str.GetLength());
                        str.Format("\t\t        ");
                        m_File.Write(str, str.GetLength());
                        str.Format("000");
                        AddItem(m_Index, 2, str);
                        m_File.Write(str, str.GetLength());
                        str.Format("\t;\r\n");
                        m_File.Write(str, str.GetLength());
                        m_NullTimeStamp.Milliseconds -= 10;

                        Size.cy = m_Index++;
                        m_List.Scroll(Size);
                }
}
else
{
                CSTIMEFIELDS m_NullTimeStamp = {0, };
```

```
                    UINT LoopCount = 0;
                    m_NullTimeStamp.Milliseconds = (m_SourceTime.Milliseconds -
                    10)-m_CompareTime.Milliseconds;
                    while(m_NullTimeStamp.Milliseconds >= 10)
                    {
                              str.Format("%6d", m_Index+1);
                              AddItem(m_Index, 0, str);
                              m_File.Write(str, str.GetLength());
                              str.Format("\t\t          ");
                              m_File.Write(str, str.GetLength());

                              str.Format("0000/00/00/00/00/00/000");
                              AddItem(m_Index, 1, str);
                              m_File.Write(str, str.GetLength());
                              str.Format("\t\t          ");
                              m_File.Write(str, str.GetLength());

                              str.Format("000");
                              AddItem(m_Index, 2, str);
                              m_File.Write(str, str.GetLength());
                              str.Format("\t;\r\n");
                              m_File.Write(str, str.GetLength());
                              m_NullTimeStamp.Milliseconds -= 10;

                              Size.cy = m_Index++;
                              m_List.Scroll(Size);

                              m_OutputCellCounts += MonitoringData.CellCounts;
                              BufferCount--;
                              ListCount++;
                    }
          }
          m_CompareTime = MonitoringData.TimeStamp;
          }
          }
}
```

감사의 글

2년여간의 대학원 생활을 마무리 지으며, 많은 조언과 격려를 아끼지 않으셨던 분들께 이렇게 감사의 글을 올립니다.

대학원 생활동안 항상 아낌없는 가르침을 주신 고성택 지도교수님께 존경과 감사의 마음을 드립니다. 배움의 이정표가 되어 주시고, 논문이 완성될 수 있도록 좋은 가르침을 주신 김경식 교수님, 김경연 교수님과 대학원 생활동안 많은 가르침과 충고를 아끼시지 않으신 이광만 교수님, 도양회 교수님, 강민제 교수님께 감사를 드립니다.

함께 대학원 생활을 하면서 친구, 선·후배로서 많은 도움을 주신 경훈이 형, 봉석이 형, 유신이 누나, 영균, 성윤, 숙인, 영아에게 감사를 전합니다. 또한 직장생활에 바쁜 용완이 형과 종수 형에게도 감사의 말을 전합니다.

함께 조교 생활을 하면서, 많은 충고와 격려와 많은 배려를 해 준 경희 누나, 올해 함께 조교 생활하고 있는 미나 선생님께 감사를 드립니다.

논문을 쓰는데 많은 도움과 충고를 주신 HanTechSnS 식구들(승학이 형, 철규 형, 영기 형, 두혁, 보승)에게 감사의 말을 전하며, 함께 연구실에서 생활했던 후배 성수, 용석, 경희, 은미, 창언에게도 고맙다는 말을 전하고 싶습니다.

늘 곁에서 사랑과 관심을 가지고 지켜봐 준 누나, 매형, 은심, 은규 그리고 귀여운 조카 지훈, 지원, 지민에게 감사를 드립니다.

마지막으로, 세상에서 가장 존경하고 사랑하는 부모님께 이 논문을 바칩니다.