목 차

그림 목차

1. 서론

1980 년대 초반 컴퓨터간의 네트워크를 구축하는 사례가 늘어났다. 컴퓨터간을 연결하는 네트워크를 구축하면서 발생하는 관리 비용 절감과 생산성 향상 등을 기대하고 각 기업체에서는 네트워크 분야의 신기술이 제품으로 발표되면 즉시 도입하여 네트워크를 구축하거나 기존의 네트워크를 확장하였다. 이러한 급진적인 확장과정에서의 호환성에 관한 문제가 1980 년대 중반에 이르러서 나타났으며 특히 서로 다른 기술표준을 수용한 회사일수록 이 문제가 심각하게 다가왔다. 이러한 과정에서 발생한 문제는 다양한 장비의 구성을 가진 네트워크를 운영 및 관리하는 것과 네트워크 수요의 증대에 따라 확장할 때 일어나는 장비들간의 호환성 문제였다. 이러한 문제를 해결하기 위해서 네트워크 운영과 전체적인 관리 계획 수립의 중요성이 부각되었고, 자동적 네트워크 관리에 대한 요구를 야기 시켰다. 따라서 증가하는 네트워크 세그먼트를 제한된 인력으로 효율적으로 관리할 수 있는 네트워크 관리 시스템이 각광받게 되었다. 네트워크 단위의 관리 시스템이 호스트 단위의 관리 시스템보다 구현 및 문제 발생시 원인 분석에 있어 훨씬 더 많은 노력을 요구하지만, 그럼에도 불구하고 컴퓨터 네트워크 시스템의 중요성은 계속 증가하고 있고, 현재 기업체등에서 필수 불가결한 요소로 받아들여 지고 있다. 보다 많은 네트워크 어플리케이션을 통한 편한 사용자 환경을 제공하기 위하여 점점 더 복잡해지고 다양한 컴퓨터 네트워크가 구축이 되면서 시스템과 네트워크 관리라는 측면에서 최근 기업들은 컴퓨터 네트워크를 구성하는 모든 자원들이 정상적으로 동작할 수 있도록 이들을 통제(Control)하고 관찰(Monitor)하는 응용 소프트웨어인 NMS(Network Management System)탑재를 선호하고 있다. 현재 출시되는 네트워크 장비의 대부분이 기본적으로 SNMP[39] 에이전트(Agent)를 탑재하여 SNMP 프로토콜에 의한 관리를 가능하게 하고 있기 때문에 대부분의 NMS는 SNMP를 기본적인 네트워크 관리 프로토콜로 사용하

고 있고 새로 개발되는 대부분의 NMS에서도 SNMP를 기본적인 프로토콜로 채택한다.

SNMP 기반의 NMS개발은 관리 대상의 선정, 관리 대상을 MIB으로 기술, 기술된 MIB파일을 기반으로한 SNMP 에이전트와 SNMP 매니저(Manager) 모듈 개발, 개발된 SNMP 매니저 모듈을 기반으로 통합된 UI를 적용, 설치 및 운영 중 테스트, 기능 수정 및 보완 등의 과정을 거치게 된다. 이러한 개발 과정을 거치면서 대부분의 경우 아래와 같은 문제점들을 경험하게 된다[1].

첫번째, 개발 환경 구축의 문제이다. NMS를 개발하는 경우, 먼저 구축된 네트워크 환경에 NMS를 적용하는 경우와 신규 네트워크 도입과 동시에 NMS를 구축하는 경우가 있다. 이미 구현되어 있는 네트워크에 신규 NMS를 적용하는 경우는 기존에 구축되어 있는 네트워크를 개발환경으로 사용해야 한다. 이러한 경우 개발과정에서 기존 네트워크의 운영에 지장을 초래할 위험이 있기 때문에 주의를 요하는 경우가 많다. 그리고 신규 네트워크 설치와 NMS의 개발을 병행하는 경우엔 개발 환경으로 사용될 네트워크의 구축이 끝난후 NMS 개발을 착수하는 것이 비용적인 측면 ,개발 일정상의 문제, 네트워크 구축전 개발 환경 부재 등으로 불합리한 경우가 많다. 따라서 네트워크 구축이 완료될 때까지 NMS자체의 개발이 지연되거나 연기되는 경우가 많았다.

두번째, NMS개발이나 운영중의 특정 오류 상황 재현이 어렵다. 개발이나 운영 중에 발생하는 NMS 자체의 문제나 네트워크 상의 문제 현상을 해결하기 위해서 문제가 되는 상황을 반복해서 재현할 필요가 있다. 이러한 상황 재현을 실제 사용중인 네트워크 환경에서 수행하기는 현실적으로 많은 어려움을 가지고 있다.

세번째, 신규 SNMP 에이전트의 개발, 적용 시 전체적인 개발 일정을 지키기 어렵다. SNMP를 이용한 NMS개발을 완료하기 위해서는 SNMP 에이전트 개발이 먼저 이루어져야 한다. 따라서 SNMP 에이전트

개발이 지연될 경우 NMS개발 프로젝트 전체가 지연되게 된다.

이러한 문제점을 해결하기 위하여 실제의 네트워크 장비에 탑재되어 있는 SNMP 에이전트를 가상적인 공간 안에서 시뮬레이션 할 수 있는 SNMP 에이전트 시뮬레이터 들이 개발되어 사용되고 있다[2, 3, 4]. 이러한 시뮬레이터의 사용은 NMS개발 기간의 단축, 네트워크 운영 방법 교육의 효율성 극대화 및 유지 비용 절감 등의 효과를 가져 온다.

본 논문에서는 기존 SNMP 에이전트 시뮬레이터의 구조를 분석하고, 개선된 SNMP 에이전트 시뮬레이터의 구조를 제시하고자 한다. 본 연구에서는 동시에 시뮬레이션 될 수 있는 SNMP 에이전트의 수를 최대화 하기 위하여 최소한의 메모리를 사용하는 최적화된 SNMP 에이전트 class를 설계하였다. 이로 인해 SNMP 에이전트 시뮬레이터에서 동시에 시뮬레이션 할 수 있는 SNMP 에이전트의 개수가 증가되었다. 또한 대부분의 SNMP 에이전트가 채택하고 있는 Tcl/Tk를 사용하지 않고 C/C++언어를 사용하였기 시뮬레이터의 동작 속도가 증가 하였으며 다수의 SNMP 에이전트 시뮬레이션 결과를 종합 분석하기 위하여 global event log handler를 통해 각 SNMP 에이전트에서 발생하는 이벤트들을 종합해서 직렬화(serialization)시켜서 저장하기 때문에 시뮬레이션 결과 분석을 보다 용이하게 하였다. 따라서 이 SNMP 시뮬레이터는 NMS개발과정에서 test-bed로 사용할 수 있고, SNMP 에이전트개발 과정에서 MIB 설계를 검증하는 용도로 사용할 수 있다. 또한 마켓팅 과정에서 NMS제품의 동작 시연시에 유용하게 사용할 수 있다. 또한 아주 작은 메모리를 사용하기 때문에 Embeded OS 에서도 사용이 가능하다. 그리고 외부에서 MIB compile 결과와 back end process definition table 만 지정해주는 것 만으로도 SNMP 에이전트 개발이 가능하므로 Instant SNMP agent development toolkit으로 사용할 수 있다.

논문의 구성은 다음과 같다. 제 2 장에서는 NMS의 개념과 SNMP 프로토콜, 기존 SNMP 에이전트 시뮬레이터들의 특징 및 적용된 시뮬레

이션 지원 기능들에 대해서 알아보고 제 3 장에서는 SNMP 에이전트 시뮬레이터를 개발 할 때 고려해야 할 사항들을 정리하고 제 4 장에서는 구현한 SNMP 에이전트 시뮬레이터의 구조에 대하여 설명한다. 제 5 장은 실제 구현 내용을 설명하고 제 6 장은 결론과 앞으로의 과제에 대하여 설명한다.

2. 관련 연구

제 2 장에서는 네트워크 관리의 기본 개념을 설명하고, SNMP 기반의 네트워크 관리에 대하여 고찰해 본 뒤, 현재 사용되고 있는 SNMP 에이전트 시뮬레이터들의 종류와 기능에 대하여 분석한다.

2.1 네트워크 관리의 기본 개념

소규모의 네트워크에서는 관리 대상이 한정되어 있기 때문에 NMS 도입의 필요성을 느끼지 못했다. 그러나 네트워크가 확대 되고 통신장비 및 시스템의 수가 증가 함에 따라서 이들을 효과적으로 관리할 수 없는 경우에는, 향후 망의 확장, 유지보수 및 서비스 제공에 커다란 장애요인으로 등장하게 된다. 이는 네트워크 운영 및 관리가 비효율적으로 이루어 진다는 측면 뿐만 아니라 투자 비용의 낭비가 일어날 확률이 많기 때문이다. 각 네트워크 장비 제조회사에서 제작된 많은 종류의 네트워크 장비들이 각각 독자적인 관리 프로토콜을 가지고 있다면 이를 일관성 있게 관리한다는 것은 거의 불가능 할 것이다. 따라서 국제 표준화 그룹은 네트워크 관리를 위한 표준 프로토콜을 제정하고 있으며, 이러한 표준 프로토콜을 지원하는 통신 장비를 사용하도록 권장하고 있다. 현재 보편적으로 쓰이고 있는 네트워크 관리 프로토콜에는 CMIP[11], 본 논문에서도 다루고 있는 SNMP(Simple Network Management Protocol) [5]등이 있다.

2.1.1 네트워크 관리 시스템의 구조

현재 사용중인 대부분의 네트워크 관리 시스템 (Network Management System, 이하 NMS로 표기)의 구조들은 그림 1 과 유사한 기본 구조를 사용한다. 에이전트(Agent)는 관리 대상 장비 내부에 존재하는 프로그램이며, NMS에서 보내는 관리 대상 정보에 대한 정보 전송 요청을 수신

한 뒤, 그 요청 받은 정보를 수집해서 그 결과를 NMS로 전송하는 역할을 한다. 관리 대상(Managed Device)은 NMS가 관리하는 네트워크 장비이며 내부에 NMS가 질의를 하고 값을 가져올 수 있는 관리 대상으로 선정된 정보를 가지고 있다. 일반적으로 각종 Hosts, Routers, switches, Terminal Servers, Printers 등을 말한다.

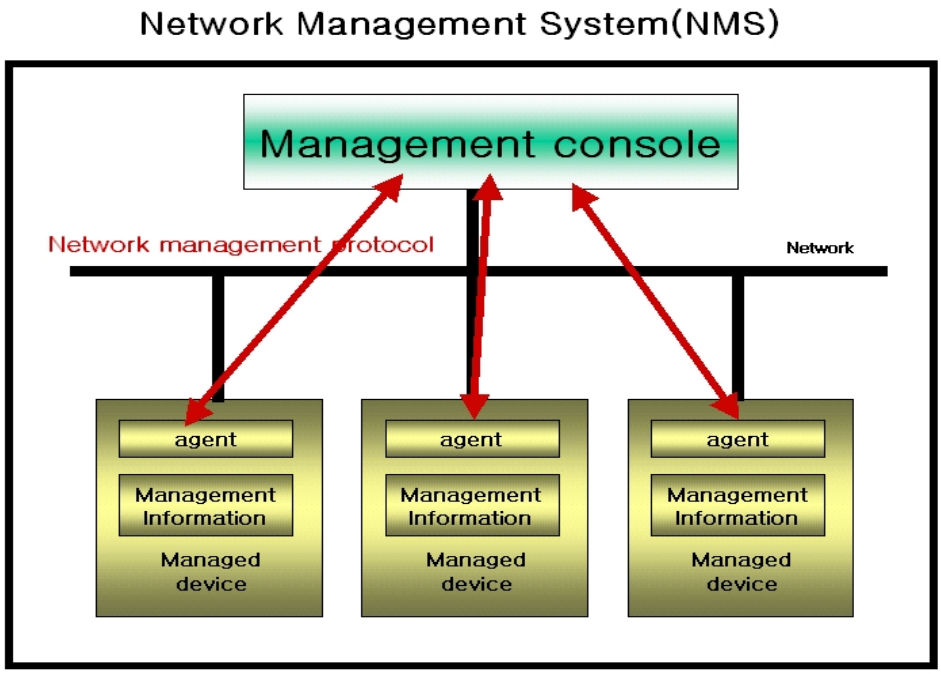

그림 1 Network Management System(NMS)의 구조

보편적으로 NMS는 일반적인 서버나 워크스테이션 상에서 동작하는 네트워크 관리 프로그램을 말하며, 네트워크 관리 프로토콜을 통하여 에이전트와 통신을 하고, 관리자에게 그 정보를 표시해 주고, 관리자의 지령을 에이전트에 전달해주는 역할을 한다. 네트워크 관리 프로토콜(Network Management Protocol)은 NMS와 에이전트의 구성 요소끼리 주고받는 통신규약이다. 에이전트는 컴퓨터 시스템이나 네트워크 장비

등의 관리 대상 장비에서 문제가 발생했을때 NMS로 경보(Alarm)를 전송할 수 있는 기능을 가지고 있으며 관리대상에 대한 정보를 관리하는 관리대상정보 데이터베이스(Management Information DB)를 관리하고 있다. 네트워크 장비 내부에서 발생한 문제가 어떤 정해진 한계 수치(threshold)를 넘어서면, NMS에게 네트워크 관리 프로토콜을 통하여 경보를 전송하게 된다. 경보를 수신한 NMS는 운영자에게 이를 알리거나, 이 사실을 기록하거나, 문제를 해결하기 위한 절차를 수행하는 등의 역할을 수행하게 된다. NMS는 자동적으로 수행되거나 또는 관리자의 요청에 의해 각각의 관리 대상들에게 특정한 관리 대상 정보에 대해 정보 제공 요청(Poll)을 할 수가 있는데, 이때 관리 대상들은 NMS의 요청(Request)을 분석하고 정보를 수집해서 NMS에게 알려주게 된다.

2.1.2 Network Management Model

ISO[6]에서 제시한 Network management model은 장애 관리(Fault Management), 설정 관리(Configuration Management), 계정 관리(Accounting Management), 성능 관리(Performance Management), 보안 관리(Security Management)로 이루어져 있다.

장애 관리는 네트워크에서 발생하는 문제점들을 검출하고, 기록하고, 사용자에게 알려주거나 혹은 수리를 하는 역할을 수행한다. 구체적으로는 장애 노드의 발견(Detection), 장애 노드의 통보(Notification), 장애 노드의 분리(Isolation), 장애 노드에 관한 정보 획득(Monitoring) 및 장애 노드의 복구(Recovery) 등의 네트워크 장비의 오류사항 및 상태의 관리 기능, 장애 이력 보관 및 장애 알람(Alarm) 자동화 및 이벤트 설정 기능 등이 있다.

설정 관리는 네트워크의 동작과 관련된 여러 하드웨어 및 소프트웨어들에 대한 설정 정보들의 관리, 네트워크의 구성 정보 및 시스템 설정 상황을 감시하는 역할을 수행한다. 구체적인 기능은 논리적인

Topological MAP 구성, 네트워크 관리 대상 장비에 대한 Auto discovery, 소프트웨어의 Version 관리 및 장비의 시스템 정보, 설정 정보 및 모듈 관리 등이다.

계정 관리는 사용자나 혹은 사용자 그룹에게 네트워크 자원을 균등하게 분배하여 사용할 수 있도록, 네트워크 자원의 사용 정도를 측정하고 이를 분석하는 역할과 효과적인 자원의 관리 및 과금, 회계처리 기능을 가진다.

성능 관리는 네트워크의 성능이 어떤 주어진 수준 이상으로 유지될 수 있도록 네트워크의 성능과 관련된 정보를 수집하고 분석, 관리하는 역할을 수행하며, 네트워크 정보 표시, 성능 모니터 기능, 장비의 현재 상태 및 사용현황 모니터링, 네트워크 트래픽 모니터링등의 기능을 가지고 있다. 또한 장비 및 인터페이스별 트래픽 상황 분석, Capacity Planning 기능, 자원의 재분배 및 효율성 극대화를 위한 성능 보고서 제공 기능도 있다.

보안 관리는 허가된 사용자만이 네트워크 자원을 접근하여 사용할 수 있도록 하는 역할을 수행하기 위한 것으로 사용자별 접근 권한 설정, NMS 구성요소별 접근 권한 설정 등의 기능이 있다.

2.1.3 NMS 의 도입 효과

신속한 장애 처리와 네트워크 모니터링을 통한 장애 예측 기능으로 네트워크 운영 중단 시간을 최소화 할 수 있다. 따라서 기업 생산성을 극대화 시킬수 있다. 네트워크 자원의 사용량과 운영 상태에 대한 다양한 정보를 수집해서 효율적인 관리 및 운영 계획 수립을 할 수 있다. 또한 많은 수작업을 통해 이루어지던 네트워크 관리를 자동화 할 수 있어서 적은 유지 보수 인력을 유지하고도 전체 네트워크의 운영이 가능하며, 이를 통해서 유지 보수비용의 절감을 이룰 수 있다. 이러한

결과로 인하여 비용 절감 효과와 기업 정보 활용의 극대화를 이루어 기업의 경쟁력 강화를 이룰수 있다.

2.2 SNMP 기반의 네트워크 관리

표준 네트워크 관리 프로토콜이 정해지기 전에는 각 네트워크 장비 업체들은 자신들의 장비끼리만 통용이 되는 각기 다른 Network management protocol을 지원하고 있었다. 따라서 각 업체들은 NMS제품에 대한 지원 소프트웨어는 물론 이러한 NMS와 연동할 수 있는 장비 자체의 Firmware까지도 제공해야만 했다. 일부 규모가 큰 회사들은 서로 제휴를 맺고 독자적인 네트워크 관리 프로토콜 정보를 교환하여 상호 지원하는 방법을 사용하였다. 하지만 이것은 근본적으로 호환성 문제를 해결한 것이 아니었기 때문에 이것은 널리 보급되지 못하였다.

TCP/IP 환경의 인터넷 망 관리를 위해서 초기에는 ICMP(Internet Control Message Protocol)[7]를 이용하여 end-to-end 네트워크 노드간의 연결 상태를 파악하였다. 하지만 ICMP를 사용하여 얻을 수 있는 정보는 단순하게 상대방 HOST가 작동하고 있는지 여부와 응답시간 등으로 한정되어 있었다. 그러나 인터넷에 접속되어 있는 컴퓨터의 수가 엄청나게 증가하고 네트워크의 구성이 복잡해짐에 따라 이러한 단순한 기능만으로는 관리에 많은 어려움을 겪게 되었고 보다 정밀하고 기능이 좋은 네트워크 관리 프로토콜이 필요하게 되었다. 이러한 요구에 따라 1988 년 초 IAB(Internet Architecture Board)[8]에서는 표준화 작업을 시작했다. 이때까지 연구가 진행되어오던 HEMS(High Level Entity Management system)[9], SGMP[10](Simple Gateway Monitoring System), CMIP/CMIS(Common Management Information Protocol/Services)[11]중에서 SGMP를 발전시킨 SNMP를 표준으로 채택하기로 결정하였다.. 그 당시 HEMS는 연구는 상당히 이루어져 있었으나 실제 적용 사례가 없었고, CMIP/CMIS는 너무 방대하고 구현이 전혀 되어있지 않은 상태였다.

SGMP는 NYSERNET[12] 및 SURAnet[13]에서의 실제적인 요구에 의해서 개발된 것으로 Proteon에서 작업중이었고 MERIT, IBM, MCI에 의해 NFSNET에서 사용하기 위한 작업을 진행 중이었으며 실제로 필드에 적용해서 유용하게 사용되고 있었다. 결국 IAB는 단기적으로 기본적으로 SNMP를 채택하고, IAB와 업체들은 ISO CMIS/CMP를 기반으로 한 NMS를 발전시키고, SNMP와 관련된 작업은 IETF[14]가 책임지며, HEMS를 위해서 정의된 MIB을 포함한 이전의 연구 작업 결과를 적극 수용하자는 결정을 내렸다. 이렇게 출발한 SNMP는 구현이 쉽고 간편해서 오늘날 가장 일반적인 네트워크 관리 프로토콜이 되었다.

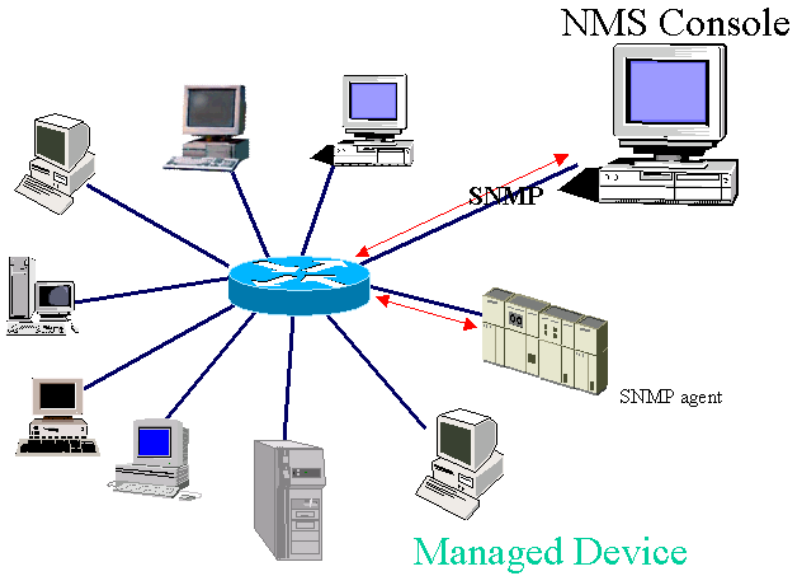

그림 2 SNMP 기반의 네트워크 관리

그림 2 는 NMS 콘솔과 Managed Device 에 내장된 SNMP 에이전트가 SNMP 프로토콜을 사용해서 통신하는 SNMP 기반의 네트워크 관리에 대한 개념적인 구조를 보인 것이다. SNMP 프로토콜의 동작에 대한 구체적인 구조는 그림 3 에 나와 있다.

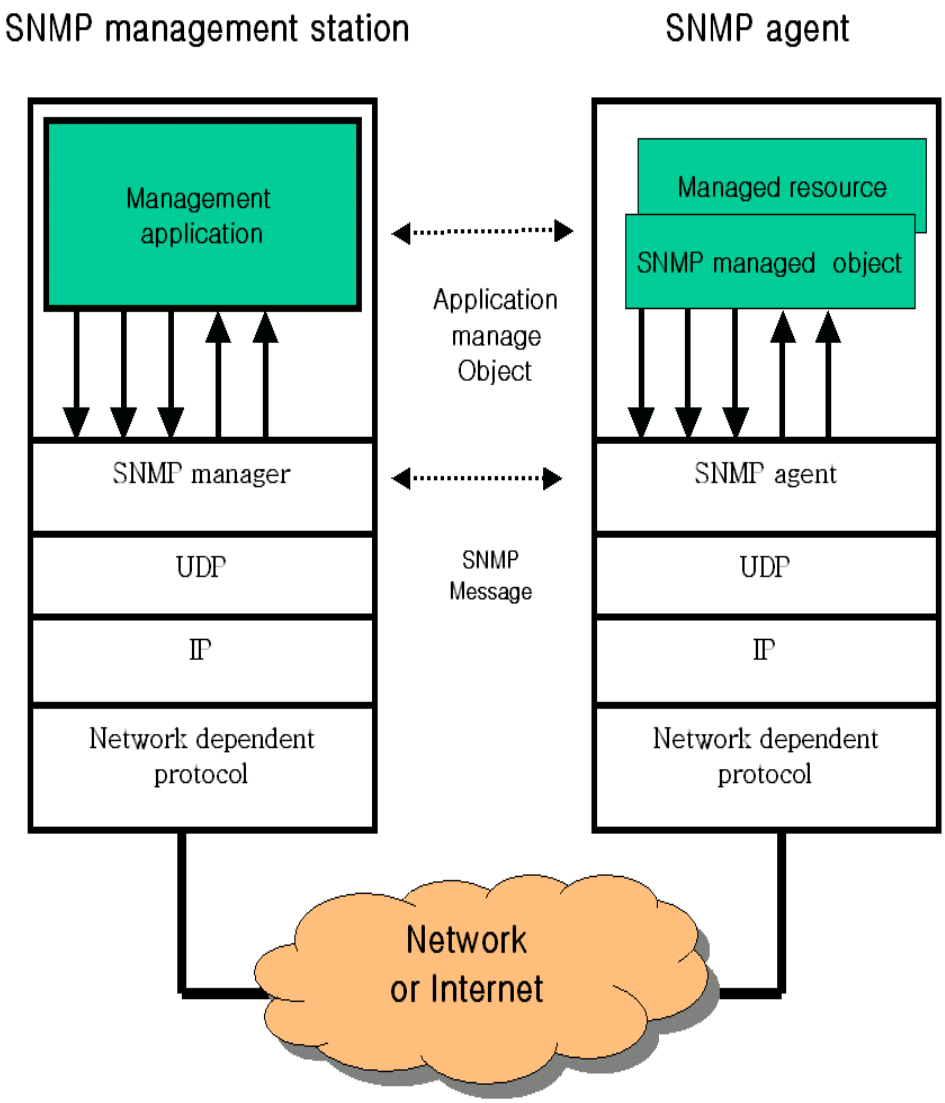

그림 3 SNMP 기반의 네트워크 관리 시스템의 동작

2.2.1 MIB (Management Information Base)

네트워크 관리는 관리자(Manager)와 에이전트(Agent)사이에 특정한 정보를 주고 받으면서 이루어 진다. 이 과정에서 주고 받는 특정 정보

를 Management Information이라 한다. 이러한 정보를 모아 놓은 집합체를 MIB(Management Information Base)이라고 한다. 네트워크 관리의 의미는 관리 대상인 장비들이 제공하는 MIB중에서 특정 정보를 얻어와서 그 장비의 상태를 파악하거나 그 값을 변경하는 것을 말한다. 값의 변동을 통하여 MIB안에 존재하는 정보의 변경과 정보의 변경에 의한 간접적인 장비의 상태 변경을 지시할 수 있다.

MIB를 정의하고 구성하는 뼈대(Framework)인 SMI는 RFC-1155 에 아래와 같은 사항을 정의해 놓고 있다. MIB의 각 객체들은 ISO와 ITU-T[15]에 의해 표준화되고 개발된 언어인 ASN.1(Abstract Syntax Notation One)[16]을 사용해서 정의한다. 정의할 모든 객체는 반드시 이름, Syntax , 부호화를 갖고 있어야 한다. 여기서 이름은 해당 객체를 식별하는 객체 식별자 이며, Syntax는 정수,문자열 등과 같은 객체의 데이터 타입을 말하는 것이고, 부호화는 객체의 데이터가 어떤 BIT패턴으로 전송되는가에 관한 것이다. MIB에는 다음의 세 종류가 있다.

- MIB-1 : MIB-1 은 원래는 MIB라고 불렀으나 MIB의 확장판인 MIB-2 가 발표됨에 따라 MIB-2 와 구별하기 위해서 MIB-1 이라 불리게 되었다. MIB -1 은 네트워크 관리에 필요한 최소한의 관리대상(114 개 항목)을 정의하고 있다.
- MIB-2 : MIB-2 는 MIB-1 의 확장판으로 MIB-1 의 모든 MO들을 포함하여 총 171 개의 MO를 포함하고 있다. 현재 판매되고 있는 대부분의 네트워크 장비들에서 MIB-2 를 지원하고 있다.
- Enterprises MIB : MIB-1, MIB-2 에서는 규정되어 있지 않으나, 장비 제조 업체가 가지고 있는 독자적 기능을 SNMP에서 관리할 수 있도록 정의한 것이다.

2.2.2 MIB 의 구조

모든 관리 대상 정보 객체(Managed Object, 이하 MO라 표기) 는 그림 4 와 같은 트리 형식의 계층적 구조를 이루고 있다. 계층적 트리에서의 Leaf Object가 실제의 MO이며 각각의 MO는 Object Identifier(OID)를 가지고 있다.

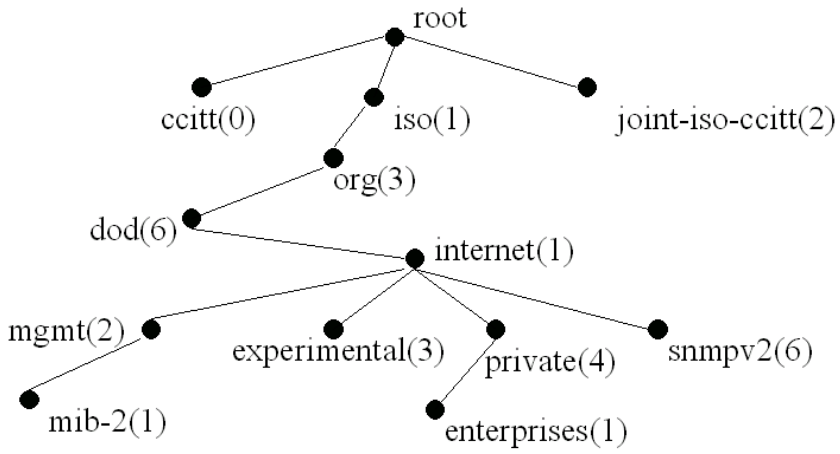

그림 4 SNMP OID tree 구조도

Object Identifier(OID)는 MIB에 기술된 MO에 부여되는, 고유하고 유일한 ID이다. OID의 표기 방법의 예는 아래와 같다.

internet OBJECT IDENTIFIER : = { iso(1) org(3) dod(6) 1}

위의 OID는 {1 3 6 1} 이나 1.3.6.1 이라고 표기될 수 있다. 예를 들어 TcpConnTable은 1.3.6.1.2.6.13 이라고 표시가 되는데 이것을 풀어서 해석을 해보면 아래와 같다.

Iso org dod internet mgmt mib-2 tcp tcpConnTable

1 3 6 1 2 1 6 13

그림 4 에서 보이는 것과 같이 SNMP SMI는 "iso(1) org(3) dod(6)"의 서브트리로 "internet(1)"을 명시하고 "internet(1)"의 서브트리로 directory(1) , mgmt(2) , experimental(3), private(4)과 private(4)의 subtree로 enterprises(1)를 정의해놓고 있다. SNMP MIB에 표준으로 정의되어 있지 않은 임의의 비 표준 MIB은 이 private(4) enterprises(1)에 정의할 수 있다. 이 이하의 subtree는 각 업체에서 임의로 사용할 수 있다. 이런 경우 각각 서로 다른 네트워크 장비 제조사를 구별하는 OID가 있어야 하는데 이는 IANA(Internet Assigned Number Authority)[17]에서 관리하고 있다. 그러므로 private enterprises MIB를 구현하기 위해서는 그림 4 에 보이는 것처럼 IANA에서 OID를 부여 받아야 한다. 그래야 그 하위 MIB을 구성하는 객체의 유일성이 보장되고 그래야 다른 업체의 사설 MIB와 구별이 가능하다. IANA에 등록하면(iana-mib@isi.edu) 번호(OID)를 부여 받을 수 있다. MIB 트리의 access는 Lexicographical Ordering방법을 사용한다(그림 5 참조).

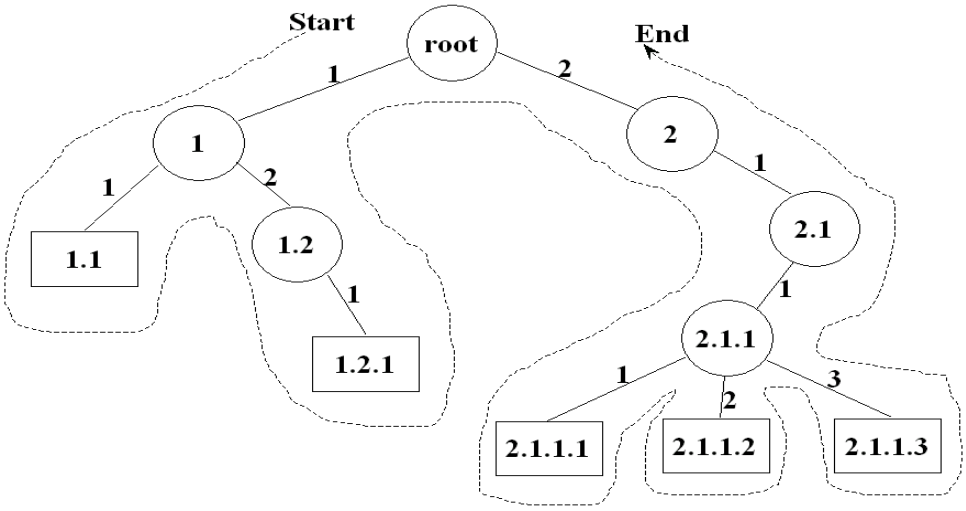

그림 5 MIB 트리의 Lexicographical Ordering 에 의한 access

2.2.3 ASN.1

SNMP에서 사용하는 메시지 형식이 길이가 유동적이고 복잡하기 때문에, SNMP 메시지 구조를 서술하기 위하여 ASN.1(Abstract Syntax Notation One)[16]을 사용한다.

2.2.3.1 ASN.1 의 표기법

예를 들어, ABC 라는 MO의 구조를 구조를 ASN.1 표기법을 이용하여 서술하면 아래와 같다.

```
ABC ::= SEQUENCE {
        source    OCTET STRING (SIZE(8)),
        dest      OCTET STRING (SIZE(8)),
      Number   INTEGER (3000..4000),
        data    ANY (SIZE(2-4500)),
        crc     OCTET STRING (SIZE(8))
          }
```

'::=' 의 오른쪽은 식을 정의하고 왼쪽 (위의 예시에서는 " ABC") 에는 변수가 정의된다. SEQUENCE {} 는 {} 안에 들어가는 항목의 순서 목록을 정의하는 것이다. 위에 제시된 예에서의 source, dest, Number, data, crc등은 MO ABC의 필드 이름이다. 각 필드의 유형은 필드 변수의 이름 바로 뒤에 적는다

예를 들어, source와 dest는 각각 그 값으로 0 개 이상의 octet을 갖는 데이터 유형을 정의하는 OCTET STRING으로 정의된다. Octet string을 구성하는 octet은 8bit로 구성되어 있으므로 각 octet은 0 부터 255 까지

값을 가질 수 있다. Number는 정수 형태의 값을 가질수 있는 INTEGER로 정의가 되어 있다. INTEGER 뒤의 ()안의 숫자는 정수 형태의 변수가 가질 수 있는 최소값과 최대값이다. 이 예에서는 3000 이상 4000 이하의 값을 가질 수 있다. Data는 임의의 형(ANY)을 가지며, 길이는 2 에서 4500 octet까지 가질 수 있다.

2.2.3.2 ASN.1 에서 지원하는 변수 타입

ASN.1 에서는 크게 두 가지 부류의 데이터 형식이 지원되는데, Universal 타입과 Application-wide 타입으로 나눌 수 있다. Universal 타입은 INTEGER, OCTET STRING, NULL, OBJECT IDENTIFIER, SEQUENCE, SEQUENCE-OF가 있고, Application-wide 타입은 Network address, IP address, Counter, Gauge, Time ticks, Opaque가 있다.

2.2.4 SNMP 프로토콜의 동작 방법

이 단원에서는 SNMP 프로토콜에 정의된 세가지 동작(Get, Set, Trap)에 대해서 알아보고 SNMP message 와 SNMP PDU 에 대해서 설명한다.

2.2.4.1 SNMP 의 작동

SNMP는 기본적으로 Get, Set, Trap 이렇게 3 가지 동작으로 Network을 관리한다. 그림 6 은 SNMP의 기본적인 동작 3 가지의 발생 형태를 나타낸다.

- GET : SNMP 매니저 에서 SNMP 에이전트에 있는 MO의 값을 가져온다.
- SET : SNMP 매니저 에서 SNMP 에이전트에 있는 MO의 값

을 변경한다.

- TRAP : SNMP 에이전트에 있는 특정 상황 발생을 SNMP 매니저에게 알린다.

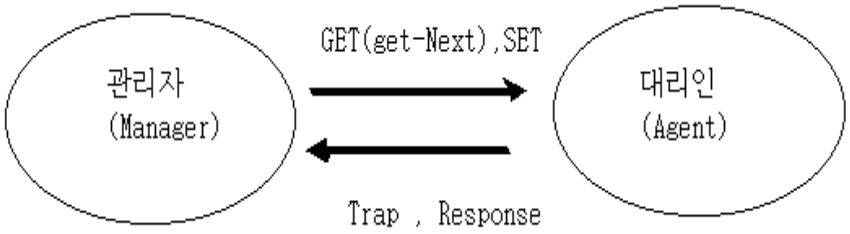

그림 6 SNMP 의 기본적인 통신 방법

2.2.4.2 SNMP 메시지 와 SNMP PDU

SNMP 매니저와 SNMP 에이전트는 SNMP 메시지를 사용하여 상호 통신을 하게 된다. SNMP 메시지는 SNMP 버전 번호, SNMP 커뮤니티 문자열, 그리고 SNMP 데이터 유닛 (PDU) 타입 정보를 포함하고 있다. SNMP 버전 번호는 0 은 SNMPv1, 1 은 SNMPv2 을 나타낸다. SNMP 커뮤니티(Community)는 SNMP 에이전트와 NMS간의 관계성(Relationship)을 나타낸다. SNMP 인증, 접근통제 등의 용도로 사용되며, default값은 문자열 " public" 을 사용한다. SNMP PDU 타입 필드에는 SNMPv1 에서 정의한 GetRequest, GetNext-Request, SetRequest, GetResponse, Trap 이렇게 5 종류를 나타내는 ID가 들어간다. SNMP PDU 의 타입별 구조는 그림 7 에 나와 있다. PDU Type별 용도는 아래와 같다.

- GetRequest : NMS 매니저에서에서 SNMP 에이전트내에 있는 특정 MO의 값을 읽어 올 때 사용한다.
- GetNextRequest : NMS 매니저에서 지정한 SNMP 에이전트 내의 MO의 다음 Mo의 값이나 지정한 MO가 테이블인 경우

다음 인덱스 값을 가져올 수 있게 한다. 지정된 MO의 다음 MO는 앞에서 설명한 MIB트리의 Lexicographical Ordering 방법에 의하여 결정 된다.

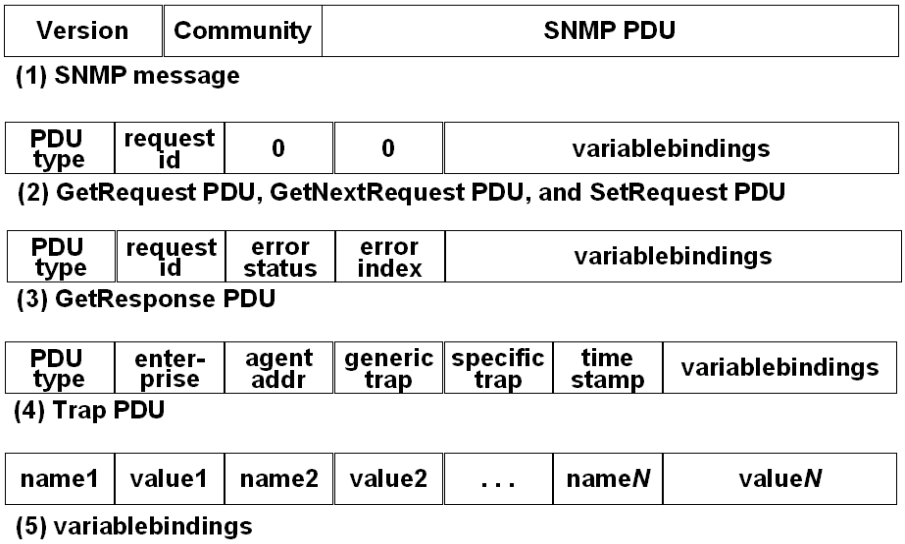

그림 7 SNMP Message 와 PDU 의 타입

- GetResponse : NMS 매니저의 요구에 반응하여 SNMP 에이전트에서 해당 MO의 값을 NMS 매니저에게 되돌려 줄 때 사용한다.
- SetReuest : NMS 매니저에서 SNMP 에이전트 내부에 있는 MO의 값을 변경할 때 사용한다.
- Trap : SNMP 에이전트에서 특정 상황이 발생했음을 NMS 매니저에게 알릴 때 사용한다.

그림 7 에서 “ error status”는 SNMP 에이전트가 SNMP 메시지를 해석하고 해당 MO에 대한 요구를 과정에서 해당 MO 또는 다른 곳에

서 문제가 발생한 경우를 표시한다. SNMPv1 에서는 noError(0), tooBig(1) (해당 객체의값을 채워서 관리자에게 전송하기 위해서 만든 메시지가 MTU보다 크다), noSuchName(2), badValue(3) (SNMP 에이전트 내부에 있는 객체 값을 변경할 때 변경하려는 데이터의 값이 부적절한 값인 경우, 즉 데이터의 타입이 MIB에 정의된 것과 다르거나 값의 범위를 벗어난 경우), readOnly(4), genError(5)으로 정의되어 있다. " error-index"는 NMS에서 SNMP 에이전트에게 variable bindings에 의해 동시에 여러 MO에 대한 정보를 요청할 때 variable bindings의 몇 번째 OID에서 오류가 발생한 것인지를 나타내며 NMS에서는 이것을 보고 다시 정보를 요청할 수 있다. Variable bindings는 1-N개 까지의 OID가 이어질 수 있음을 의미한다. 각 MO의 이름, 즉 OID와 그 MO의 값을 한 셋으로 한다. 이것은 MTU가 허용하는 범위까지 가능하다. 그래서 (ID+VALUE) + (ID+VALUE).....(ID+VALUE) 형태로 메시지가 만들어지고 전달된다. 이러한 SNMP 에이전트와 NMS간의 통신이 이루어 지기 위해서는 SNMP 매니저와 SNMP 에이전트 간의 SNMP버전이 일치해야 하고, 양쪽의 Community가 일치하여야 하며 이름이 일치하여야 한다. 이때 Community는 최소한의 인증절차를 위한 암호 역할을 한다. 또한 SNMP PDU에 대한 응답 SNMP PDU의 타입이 SNMP PDU sequence에 따라야 한다.

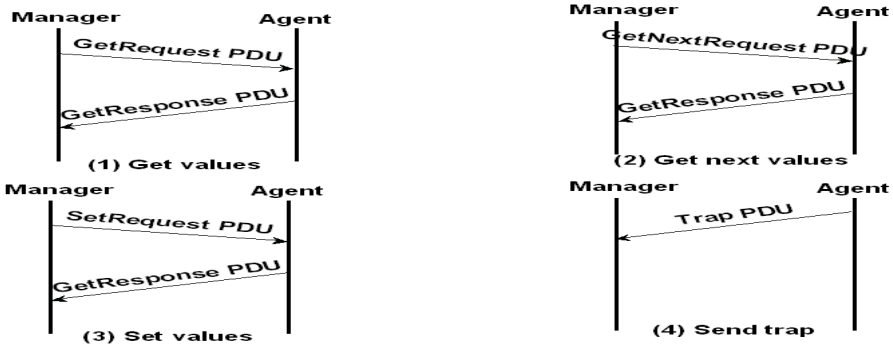

그림 8 SNMP PDU exchange sequence

그림 8 은 위에서 설명한 SNMP PDU exchange sequence를 나타낸 것이다. 그림 8 에서 알수 있는 바와 같이 NMS에서 SNMP 에이전트로 보내는 SNMP PDU는 Get/ GetNext/ GetBulk 이외의 PDU 타입이 될 수 없고, 마찬가지로 Get/ GetNext/ GetBulk에 대한 응답은 GetResponse PDU타입만 올 수 있다.

2.2.5 BER Encoding

NMS와 SNMP 에이전트 사이의 데이터를 주고 받을 때 ASN.1 타입의 변수를 octet string 형태로 변환해서 주고 받기 위하여 BER Encoding[18]을 사용한다. BER Encoding은 기본적으로 Type-Length-Value 구조를 사용하여 encoding을 하게 된다.

2.2.5.1 Type-Length-Value Encoding

SNMP 프로토콜에서 NMS와 SNMP 에이전트간에 교환되는 데이터 포맷의 표시법을 정의하기 위해, BER은 전송되는 octet내에서의 각 비트의 위치를 지정한다. 각 octet은 최상위 비트(MSB: Most Significant Bit)를 먼저 전송하고 그것을 octet 왼쪽에 있는 bit 8 로 정의하고 제일 오른쪽에 있는 bit 1 을 최하위 비트(LSB: Least Significant Bit)로 정의한다. 데이터 인코딩 구조에는 Type, Length, Value (TLV) 의 세가지 구성 요소가 있다. 이를 또는 Identifier-Length-Contents (ILC, ISO 8825-1) 이라고도 한다. SNMP에서 쓰이는 TLV 인코딩의 구조는 그림 9 와 같다.

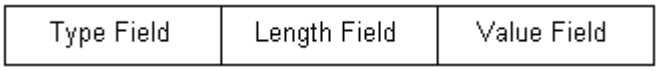

그림 9 Type-Length-Value(TLV) 인코딩

BER은 전송되는 데이터를 구성하는 octet의 구성 bit들의 순서와 구조를 정의해서(그림 10) 통신 채널의 양단에서 bit stream을 똑같이 해석할 수 있도록 해 준다. 이로 인한 이기종간의 통신에서도 전송되는 데이터의 잘못된 해석을 막을 수 있다.

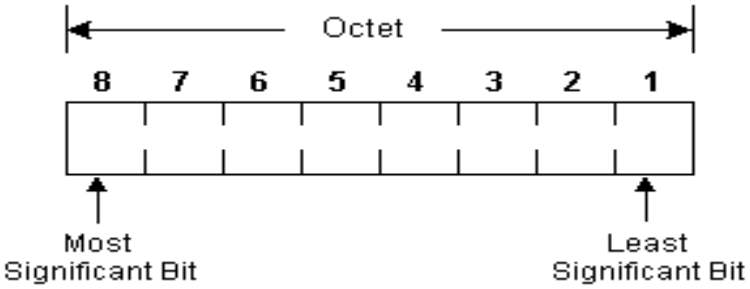

그림 10 BER 의 비트 순서 (ISO 8825-1 에 정의된 방식을 따름)

2.2.5.1.1 Type 필드

Type 필드는 TLV인코딩에서 맨 처음에 나오며, 뒤따르는 필드를 해석하는데 필요한 변수 타입 정보를 포함하고 있다. Type 필드 내의 하위 필드에는 P/C로 불리는 하나의 비트가 들어있다. P/C는 코딩방법을 Primitive(P/C = 0)나 Constructed (P/C = 1)중 어떤 것을 사용하였는지 나타낸다. 이것은 그림 11 과 같이 나타낼 수 있다. Type 필드에는 두 가지 유형이 있는데 Tag 번호의 크기에 따라 달라진다. Tag 번호가 0 과 30

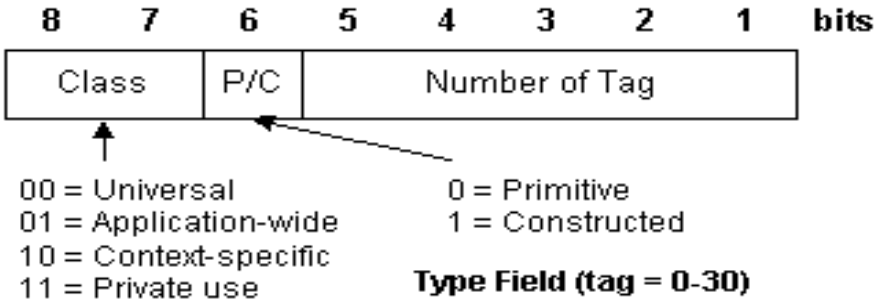

그림 11 Type 필드의 하위 필드

사이일 때는 Tag 필드에는 하나의 octet이 들어가고 Tag 번호가 31 이상일 때는 Type 필드에는 여러 개의 octet이 들어간다. 두 가지 유형 모두 첫번째 octet에는 세 가지 하위 필드 (클래스와 P/C bit 및 Tag번호) 가 들어있다. 그림 12 는 Tag 번호가 31 이상일 때는 type 필드의 인코딩을 나타낸것이다.

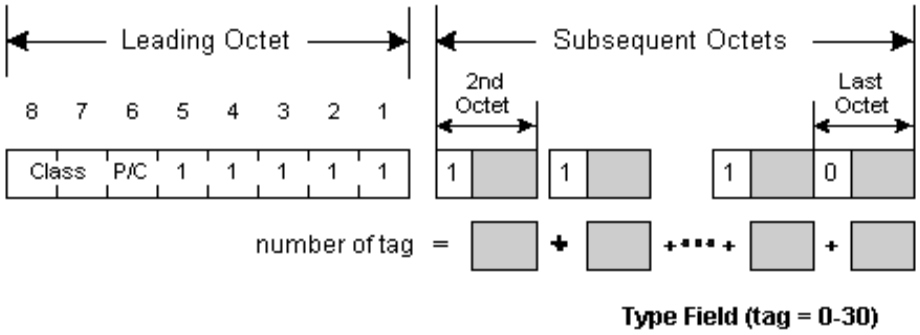

그림 12 Type 필드의 인코딩 (ISO 8825-1)

두 가지 유형 모두 클래스 하위 필드는 사용 중인 Tag 의 클래스 타입을 나타낸다. Tag의 클래스 타입은 그림 13 에 나와 있다.

Class	Bit 8	Bit 7
Universal	0	0
Application	0	1
Context – specific	1	0
Private	1	1

그림 13 Tag 의 class type

SNMP 어플리케이션은 처음의 세 클래스인 Universal, Application및 Context-specific을 이용한다. Universal 클래스는 INTEGER 형과 OCTET STRING 형을, Application 클래스는 defined타입(IpAddress, Counter 타입), Context-Specific 클래스는 다섯 개의 SNMP 프로토콜 데이터 단위(PDU)를 인코딩 한다. P/C 하위 필드 (bit 6) 는 데이터 요소의 형식을 나타낸다. Primitive 인코딩(P/C = 0) 은 value 필드의 octet 이 그 값을 직접적으로 나타낸다는 것을 뜻하며, Constructor 인코딩(P/C = 1) 은 value 필드의 octet이 하나 이상의 추가적인 데이터 값(SEQUENCE 와 같은) 들을 인코딩 한다는 것을 뜻한다. SNMP 는 0 과 30 사이의 tag 번호를 사용한다. Tag번호는 세 번째 하위 필드에 있으며, 2 진수로 표현된다. Bit 5 는 tag 의 MSB 이며, Bit 1 은 LSB 이다. Universal 클래스에 대한 tag번호는 ISO 8825-1, Application 클래스에 대한 tag번호는 RFC 1155, Context-specific 클래스에 대한 tag번호는 RFC 1157 에 정의되어 있다. 그림 14 는 SNMP에서 쓰이는 Type 필드의 세 가지 클래스와 그 필드들(Class, P/C, Tag 번호)에 대한 인코딩을 정리한 것이다. 이 인코딩은 2 진수와 16 진수 표시법 둘 다로 표시된다. 여기서, H는 16 진수 표시법일 때 쓰인다. BER에서 31 이상의 tag 번호를 제공하기는 하지만, SNMP에서는 사용되지 않는다. 31 이상의 tag 번호에 대해서 Type 필드는 다른 형식을 이용한다. 첫번째 octet내의 tag 번호는 2 진수 11111 로 설정된다. 그 이후에 추가되는 octet들은 tag번호를 나타낸다. octet의 Bit 8 = 1 은 그

뒤에 연이어서 더 많은 octet이 따른다는 것을 나타내며, octet의 Bit 8 = 0 은 그 octet이 마지막 것임을 나타낸다. 그 이후의 각 octet의 Bit 7 부터 1 까지는 tag 번호의 부호없는(unsigned) 2 진 정수이다. 그 이후의 첫번째 octet의 Bit 7 은 tag 번호의 MSB를 나타낸다.

Universal class	Type field value
INTEGER	00000010 = 02H
OCTET STRING	00000100 = 04H
NULL	00000101 = 05H
OBJECT IDENTIFIER	00000110 = 06H
SEQUENCE	00110000 = 30H
SEQUENCE-OF	00110000 = 30H

Application class	Type field value
IPAddress	01000000 = 40H
Counter	01000001 = 41H
Gauge	01000010 = 42H
TimeTicks	01000011 = 43H
Opaque	01000100 = 44H

Context-specific Class	Type field value
GetRequest	10100000 = A0H
GetNextRequest	10100001 = A1H
GetResponse	10100010 = A2H
SetRequest	10100011 = A3H
Trap	10100100 = A4H

그림 14 TYPE 필드의 encoding 결과

2.2.5.1.2 Length 필드

Length 필드는 Type필드 다음에 나오며, Value 필드에 들어갈 octet의 갯수를 나타낸다. Length 필드는 그림 15 에서 볼 수 있듯이, Short

definite 형식이나 Long definite형식을 취한다(indefinite 형식은 SNMP에서는 쓰이지 않는다). Definite형식은 encoding된 데이터의 길이가 데이터가 전송되기 전에 미리 알려진다는 것을 나타내며, indefinite형식은 그 반대를 나타낸다.

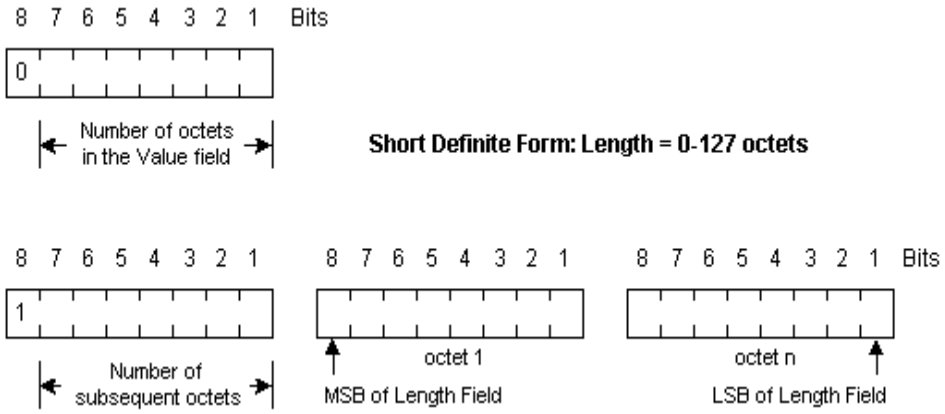

그림 15 Length 필드의 encoding

Short definite형식은 value 필드에 담겨져 있는 내용의 길이가 0 에서 127 octet 사이임을 나타내며, Long definite형식은 128 octet 이상인 것을 나타내며, Long definite 형식을 사용하여 128 octet 미만의 길이를 나타낼 수도 있다. Long definite 형식은 전체 길이를 나타내는데 여러 개의 octet을 사용한다. Long definite 형식에서는 length 필드의 첫번째 octet의 Bit 8 = 1 이다. Bit 8 다음의 비트들은 뒤따르는 octet이 몇 개인지를 나타낸다. 이 숫자는 1 과 126 사이에 있는 값이어야 한다. 숫자 127 은 앞으로 확장할 것을 대비해서 예비로 마련해 둔 것이다. 두 번째 octet의 Bit 8 은 Length 필드의 MSB로 간주되며, 그 다음의 octet들은 그 길이의 나머지 부분을 구성한다. 이로써, Long definite형식은 최대 2^{1008} - 1 octet의 길이를 나타내게 된다.

2.2.5.1.3 Value 필드

Value 필드에는 데이터 값을 전달하는 0 개 이상의 contents octet 이 들어간다.

(a) INTEGER 형의 인코딩

INTEGER 형은 양수와 음수 및 0 의 값을 갖는 Simple 타입이다. 이 데이터형은 하나 이상의 contents octet을 포함하는 Value 필드로 encoding된 Primitive형이다. Contents octet은 정수 값과 같은 값을 가지는 "2 의 보수" 2 진수이며, 필요한 만큼의 octet을 가질 수 있다.

(b) OCTET STRING형의 인코딩

OCTET STRING 은 Simple 형으로서, 그의 고유 값들은 0, 1, 또는 그 이상의 octet(각각은 8 비트의 곱이어야 한다)을 순서대로 늘어놓은 배열이다. OCTET STRING 값에 대한 인코딩은 Type 필드 = 04H 인 Primitive이다. Length 필드와 Value 필드는 인코딩된 정보에 따라 달라진다.

(c) OBJECT IDENTIFIER형의 인코딩

OBJECT IDENTIFIER 는 SNMP에서는 관리되는 객체의 고유 ID이다. 이 데이터 형의 Value 필드에는 하위 식별자들을 순서대로 늘어놓은 목록이 들어있다. 인코딩과 전송에 필요한 노력을 줄이기 위해, 첫 번째 하위 식별자가 0 이나 1 또는 2 와 같이 작은 수라는 사실을 이용하고, 그것을 두 번째 하위 식별자(보다 큰 수일 수 있다) 와 수학적으로 결합하도록 한다. 따라서, 하위 식별자들의 총 숫자는 인코딩되고 있는 OID 값 내의 객체 식별자의 숫자보다 작다. 이렇게 숫자가 줄어든 것(하나 작게)은 다른 수식을 이끌어 내기 위해 처음의 두 OID 구성

요소들을 사용하는 수식 때문이다.

Given X is the value of the first OID, and Y is the second:

First subidentifier = (X * 40) + Y

이 하위 식별자들의 값은 인코딩된 뒤 Value 필드 내에 위치하게 된다. 각 octet의 Bit 8 은 그 octet이 그 값을 완전하게 기술하는데 필요한 octet들의 배열 내에서 마지막에 있는지의 여부를 나타낸다. Bit 8 = 1 이라면 적어도 하나의 octet이 뒤따르며, Bit 8 = 0 이면 그것이 마지막의 (또는 유일한) octet임을 나타낸다. 각 octet의 Bit 7 에서 1 까지는 하위 식별자들을 인코딩한다. MIB-II 객체 트리 구조에 있는 System 그룹을 이용하여, OBJECT IDENTIFIER 가 다음 값을 갖는다고 가정하자.

{ iso org(3) dod(6) internet(1) mgmt(2) mib-2(1) 1 }

객체 트리에서, 이것은 { 1 3 6 1 2 1 1 } 와 같이 표현된다. 첫 번째 하위 식별자 값에 대하여 X = 1 과 Y = 3 의 값과 위의 수식을 이용하면, 다음과 같이 표현할 수 있다.

(1 * 40) + 3 = 43

이 때문에 첫 하위 식별자의 값은 43, 두 번째 하위 식별자의 값은 6, 세 번째 것은 1 이 된다. 인코딩을 하려면 첫 번째 값 (43) 에는 6 비트 또는 1 octet이 필요하다. (00101011) 두 번째 값 (6) 을 인코딩 하려면 3 비트가 필요하며 (110), 따라서 하나의 octet만 있어도 된다. 그 이후의 값들 또한 하나의 octet을 필요로 한다. 따라서 인코딩은 다음과 같이 된다.

Type 필드 = 06H (OBJECT IDENTIFIER, Tag = 6),

Length 필드 = 06H,

Value 필드 = 2B 06 01 02 01 01 H

(d) NULL형의 인코딩

NULL 타입은 정보가 없다는 것을 알려주기 위한 것이다. NMS에서 특정 OID에 해당하는 변수를 요구할 때 SNMP 에이전트에서 보낼 데이터가 없을 경우에 공백을 채우기 위하여 NULL 타입을 사용한다. NULL 타입에 대한 인코딩은 Primitive 이다. Type 필드 = 05H 이며, Length 필드 = 00H 이다. Value 필드는 비어있다(값이 없는 octet) .

2.3 SNMPv2 의 특징

SNMPv1 이 안고 있는 가장 큰 문제점은 보안 기능이 거의 없다는 것이다. 그리고 단 한 개의 scalar 데이터로 이루어진 객체에서는 큰 문제가 발생하지는 않지만, 라우팅 테이블처럼 많은 열(row)이 존재하는 경우에는 전체 테이블을 읽고 싶을 때 수많은 Request/response를 반복해야 하므로 NMS구동에 따른 네크워크 부하가 증가 하였다. 또한 여러 SNMP manager가 공존하고 있을 때 SNMP manager간의 통신 기능이 정의되어 있지 않다. 이런 문제점을 수정하기 위해서 93 년에 SNMPv2 가 등장했다. SNMP manager사이의 통신을 위한 InformRequest PDU가 정의되었고, 이를 위한 SNMP manager간의 MIB를 정의해 놓았다. SNMP 매니저 사이의 통신이 가능해짐으로 분산 처리 방식에 의한 관리 방법이 가능해 졌다. 그리고 테이블을 구성하고 있는 객체들의 값을 손쉽게 읽어오기 위해서 GetBulkRequest PDU를 도입했다. 이에 따라 한번의 요청

으로 여러 테이블 값들의 읽어오는 것이 가능해졌고 NMS 도입에 따른 불필요한 네트워크 대역폭을 줄일 수 있게 됐다. 또한 가장 문제였던 보안 기능이 추가되었다. 전송되는 테이블 보안 기능을 위해서 DES(Data Encryption Standard)알고리즘을, 인증절차에 대한 보안을 위해서는 MD5(Message Digest 5) 알고리즘을 사용한다. 그리고 party MIB를 이용해서 특정한 사용자에게만 특정 객체를 이용하거나 못하게 하는 접근통제(Access control) 기능을 추가하였다. 또한 SNMPv1 은 UDP를 기본으로 이용하는데 반해서 SNMPv2 는 전송 계층에 대해서 UDP뿐만 아니라 OSI CLNS(Connection Network Services), 애플토크 DDP(Datagram Delivery Protocol), 그리고 노벨의 IPX(Internet Packet Exchange)등도 구현할 수 있도록 정의해 놓았다. 그리고 MIB으로 표현할 수 있는 데이터 타입이 추가가 되었다. 그림 16 은 SNMP 버전에 따른 데이터 타입의 지원을 정리한 것이다.

Data Type	SNMPv1	SNMPv2,v3
INTEGER	O	O
UNSIGNED32	X	O
COUNTER32	O	O
COUNTER64	X	O
GUAGE32	O	O
TimeTicks	O	O
Octet String	O	O
IP address	O	O
Object Identifier	O	O
Opaque	O	O

그림 16 SNMP 버전에 따른 지원 데이터 타입

2.4 SNMPv3 의 개선점

SNMPv3[19]는 SNMPv2 에 보안기능을 보완한 것이다. SNMP PDU를 주고 받을 때 인증 기능과 암호화를 지원하여 SNMP 매니저와 SNMP 에이전트간의 secure한 통신을 보장하기 위한것이다. 그림 17 은 SNMP 버전별로 제공하는 보안기능에 대한 비교표이다.

모델	지원	인증방법	암호화	비고
SNMPv1	No Authentication, No privacy	Community String	No	Uses a community sting match for authentication.
SNMPv2	No Authentication, No privacy	Community String	No	Uses a community sting match for authentication
SNMPv3	No Authentication, No privacy	Username	No	Uses a username match for authentication.
SNMPv3	Authentication, No privacy	MD5 or SHA-1	No	Provides authentication based on the HMAC-MD5 or HMAC-SHA algorithm
SNMPv3	Authentication, privacy Priv	MD5 or SHA-1	DES	Provides authentication based on the HMAC-MD5 or HMAC-SHA algorithm. Also provides DES 56-bit encryption based on the CBC-DES (DES-56) standard

DES : (Data Encryption Standard)
MD5, SHA-1 : Secure Hash Function
HMAC : Message Authentication Code

그림 17 SNMP 버전별 보안 기능 지원 비교

2.5 기존의 SNMP 에이전트 시뮬레이터

위에서 살펴본 바와 같이 SNMP 에이전트는 SNMP 프로토콜을 통하여 SNMP 매니저로부터 요구받은 MO에 관한 정보를 자신이 가지고 있는 MIB 구조에 속해 있는 MO정보들 중에서 찾아서 다시 SNMP 프로토콜을 사용해서 SNMP 매니저에게 전달하는 기능을 한다. SNMP 에이전트 시뮬레이터도 기본적으로는 SNMP 에이전트를 가상적으로 구현한 것이 때문에 핵심 루틴은 SNMP 에이전트와 많은 유사점을 가지고 있다.

	SNMP agent	SNMP agent simulator
SNMP MIB support	Compile time	Run time
Back-end process	Hard coded	Dynamic, User configurable
User Interface	No	Yes
Changing MIB on runtime	No	Yes
Flexibility	No	Yes

그림 18 SNMP 에이전트와 SNMP 에이전트 시뮬레이터의 차이

하지만 그림 18 에서와 같이 SNMP 에이전트와 SNMP 에이전트 시뮬레이터의 동작 방법에 차이가 있게 된다. SNMP 에이전트와 SNMP 에이전트 시뮬레이터는 기본적으로 MIB 데이터 로딩 방식과 MO의 값을 읽거나 쓰는 루틴의 동작 방식이 다르다. 일반적으로 SNMP 에이전트는 SNMP 에이전트에서 사용될 MIB이 결정이 되면 SNMP 에이전트 소스 코드 안에 포함시키게 된다. 따라서 MIB의 변경이 일어날 경우 다시 소스코드 수정 및 컴파일 과정이 필요하게 된다. SNMP 에이전트 시뮬레이터는 SNMP 에이전트에서 사용되는 MIB이 시뮬레이션 시에 결정이 되게 된다. 따라서 MIB이 SNMP 에이전트 소스 코드 안에 MIB

을 포함시킬 수 없고, 프로그램 외부에 위치하게 된다. 그리고 SNMP 에이전트 내부에서 NMS로부터의 요청이 있었을 때 MO의 값을 읽어서 전달하거나 MO의 값을 수정하는 루틴(일반적으로 Back-End-Process 라고 부른다)이 SNMP 에이전트는 개발시에 결정되게 되지만(Hard coded), SNMP 에이전트 시뮬레이터에서는 실행중에 유저 인터페이스로부터 입력을 받아서 결정되게 된다. 이러한 두가지 다른 관점이 SNMP 에이전트 시뮬레이터를 개발하는데 있어서 중요한 고려 사항이다..

현재 개발되어 사용되고 있는 SNMP 에이전트 시뮬레이터들은 Gambit[2], SimpleSoft[3], MileStone[4]등의 회사에서 출시된 것 들이다. 먼저 각각 제품들의 특징을 살펴 보고 지원하는 SNMP 에이전트 시뮬레이션 방법들에 대해 알아 본다.

2.5.1 MIMIC

MIMIC[2]은 Gambit사의 SNMP 에이전트 시뮬레이터로, 한 개의 시뮬레이터에서 10000 개까지의 SNMP 에이전트를 동시에 시뮬레이션 할 수 있고, 자체 내장 MIB 컴파일러와, 실제 동작중인 네트웍 장비로부터 데이터를 수집해서 시뮬레이션 할 수 있는 MIMIC recorder등을 제공한다. MIMIC은 MIMICView, MIMIC Simulator, MIMIC Recorder, MIMIC Compiler, MIMIC Shell, Topology Wizard, MIB Wizard, Discovery Wizard, Simulation Wizard, Device Library, Networks Library 및 MIB Library 로 구성되어 있다. MIMICView모듈은 시뮬레이션 과정을 표시하는 사용자에게 친숙한 GUI를 제공한다. MIMIC Simulator모듈은 인텔 기반의 PC나 Sun Sparc에서 SNMP 에이전트를 시뮬레이션 할 수 있도록 하는 핵심 모듈이며 SNMP v1[37], v2[2] 및 v3[8]를 지원한다. MIMIC Recorder모듈은 네트워크 장비나 네트워크의 snapshot을 작성하는 기능을 한다. MIMIC Compiler 모듈은 시뮬레이션에 사용될 수 있도록 SMI based MIB을 컴파일 하는데 사용된다. MIMIC shell모듈은 MIMIC 프로그램 자체를 제어할

수 있는 Command-line 인터페이스를 제공한다. Topology Wizard, MIB Wizard, Discovery Wizard, Simulation Wizard 모듈들은 사용자가 시뮬레이션을 실행시키는 과정을 쉽게 할 수 있도록 도와준다. Device Library, Networks Library, MIB Library 모듈들은 시뮬레이션 환경 구축을 쉽게 해주는 역할을 한다. 실제 사용되는 네트워크 장비와 네트워크, 그리고 MIB 데이터에 관한 예제를 사용자에게 제공하는 기능을 한다. 그림 19는 MIMIC의 동작 화면이며 다수의 네트워크 장비에 탑재된 SNMP 에이전트들이 MIMIC안에서 시뮬레이션 되고 있는 것을 볼 수 있다.

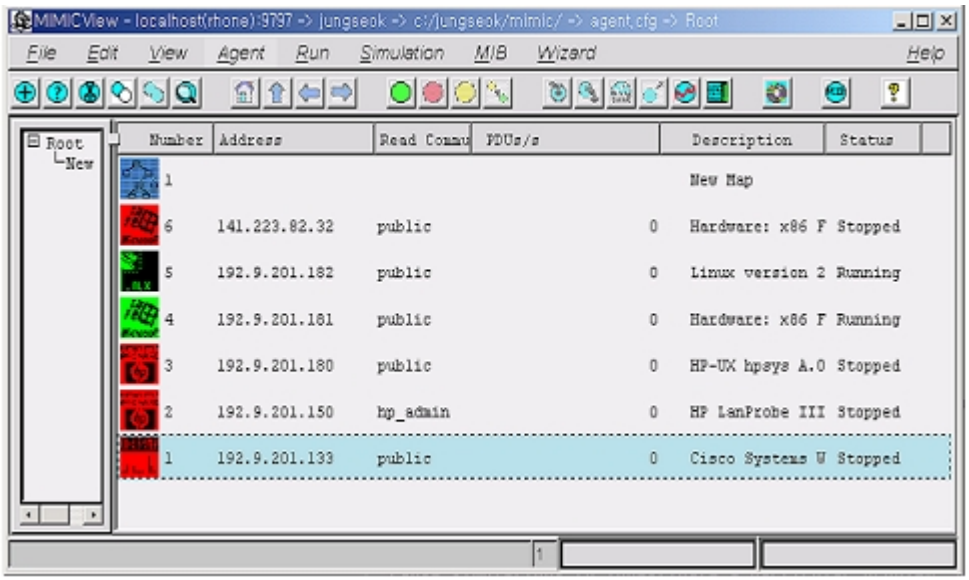

그림 19 MIMIC 의 동작 화면

Tcl/Tk[20]를 사용한 시뮬레이션을 지원하고, 현재 많이 사용되는 네트워크 장비 내부에 탑재된 MIB을 바이너리 파일 형태로 제공하며, 새롭게 출시 되는 장비에 대한 MIB 데이터를 인터넷을 통해 자동으로 업데이트 할 수 있다. SNMP 트랩(trap)을 생성시키는 기능도 포함이 되어 있으며, 발생 주기, 타입 등을 지정해 줄 수 있다.

2.5.2 Simple Agent

Simple Agent [3]는 SimpleSoft사에서 개발한 것으로 동시에 하나의 SNMP 에이전트 만을 시뮬레이션 할 수 있으며, SNMPv1 과 SNMPv2C 를 지원한다. ASCII 파일 형태의 MIB파일을 로딩하는 방법(MIB compiler)과 기존에 존재하는 네트워크 장비에 탑재된 SNMP 에이전트 내부의 MIB 구조를 SNMP WALK등의 방법을 사용해서 얻어낸 뒤 그 구조를 시뮬레이션에 사용하는 방법(MIB learner) 두 가지를 지원한다. 그리고 내장된 MIB Instance creator는 MIB에 기술된 MO들의 값을 사용자가 편집, 지정할 수 있도록 하는 도구이다. 그림 20 은 Simple Agent의 동작 화면이다.

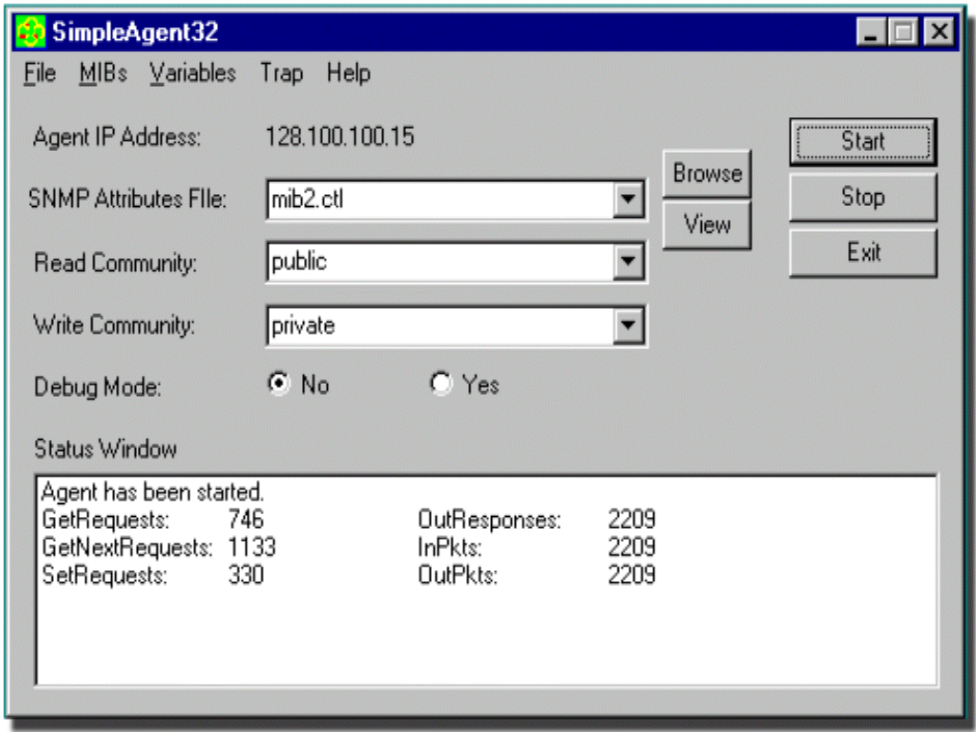

그림 20 Simple Agent 동작 화면

각 MO의 데이터 값에 fixed, sequential, random, randomUp, clock, lastset의 특성을 부여할 수 있는 특징이 있고 SNMP 트랩 발생, MIB 데이터 테이블의 행(row) 을 동적으로 생성하는 기능, 디버깅 모드에서 SNMP 패킷 교환 내용에 대한 기록 보전 기능이 있는 것이 특징이다. 동작 환경은 Microsoft Windows 95/98/NT 4.0 이고 32MB의 램과 5MB의 디스크 공간이 필요하다.

2.5.3 Simple Agent Pro

이 제품 또한 Simple Agent사의 제품이며, 기본적인 기능들은 동사의 Simple Agent 제품과 같으나 하나의 시뮬레이터 안에서 다수의 SNMP 에이전트들을 동시에 시뮬레이션 할 수 있다는 것이 특징이다. 그림 21 에서 Simple Agent Pro 안에서 서로 다른 IP 어드레스를 사용하는 여러 개의 SNMP 에이전트가 동시에 시뮬레이션 될 수 있다는 것을 볼 수 있다.

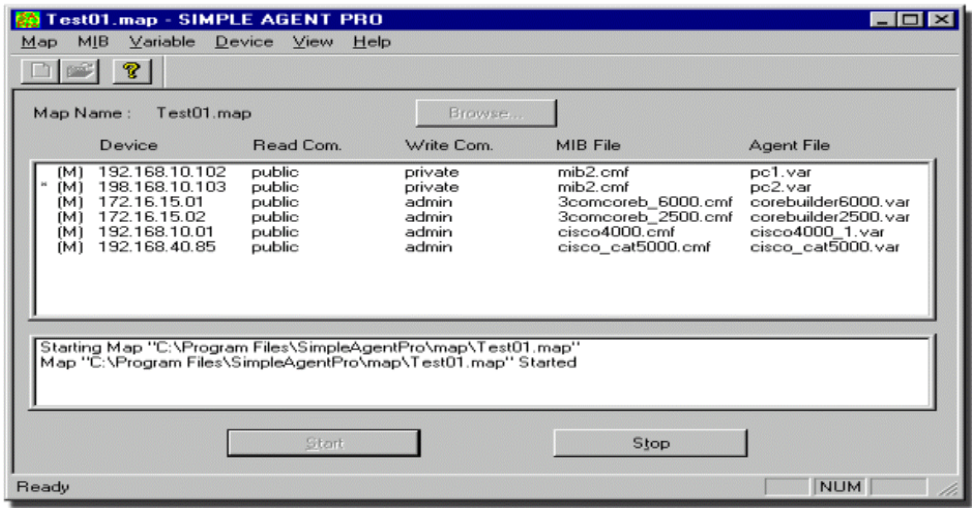

그림 21 Simple Agent Pro 동작 화면

Simple Agent 제품이 Microsoft Windows 계열 OS에서만 동작하는 것 과는 달리 이 제품은 SUN Solaris, Windows 2000, Windows NT, Windows 95/98/ME 등 다양한 OS위에서 동작한다. 시뮬레이션시에 각 MO의 데이터를 fixed, sequential, random, randomUp, clock based, lastset 등의 방법에 의해 자동적으로 값이 변하도록 설정할 수 있다. 이러한 방법을 통하여 NMS를 테스트 하는 과정에서 MO의 데이터값을 계속 수작업으로 바꾸는데 필요한 수고를 줄일 수 있다. 예를 들면 Counter형 MO의 데이터에 sequential 타입을 지정해서 NMS에서 읽어 낼때마다 1씩 증가하게 하는 방법 과 ClockTick타입의 변수에 Clock based 타입을 지정해서 현재 시간을 돌려주는 방법 등이 있다. 또한 Simple Agent Pro 내부의 API를 외부로 export시켜서 다른 프로그램을 통해서 Simple Agent Pro의 동작을 제어 할 수 있다. 그리고 Tcl 기반의 스크립트를 통하여 SNMP 에이전트의 동작 특성과 MIB variables간의 연관성을 모델링 할 수 있다. 동시에 다수의 SNMP 에이전트 시뮬레이션이 가능하기 때문에 (4000 devices on SUN Solaris 2.6, 2000+ on Windows NT and WIndows 2000, and upto a 1000 on Windows 95) 이 제품을 Simple Agent 사에서는 Network 시뮬레이터라고 부른다.

2.5.4 GeNMSim

GeNMSim[4] 은 MileStone사의 제품으로 기본적인 SNMP 에이전트 시뮬레이터로서의 기능 이외에 Tcl 언어를 사용한 callback function을 지원해서 시뮬레이션에 활용할 수 있다. MIB 파일들로부터 자동적으로 시뮬레이션될 SNMP 에이전트를 생성하는 기능을 가지고 있고, 동시에 시뮬레이션 되는 SNMP 에이전트의 개수의 제한은 없으며, 내장 SNMP 트랩 제네레이터 모듈이 내장되어 있다. 또한 GeNMSim ëProbeí란 제품을 사용하여 온라인 네트워크 트래픽 분석을 제공한다. 그리고 SNMP프로토콜의 버전은 SNMPv1, v2 를 지원한다. 동작 환경은 Windows

NT/95/98/2000 등 Microsoft계열의 OS, Sun Solaris 2.6, IBM AIX 4.1 이다. 그림 22 은 GeNMSim의 동작화면이다.

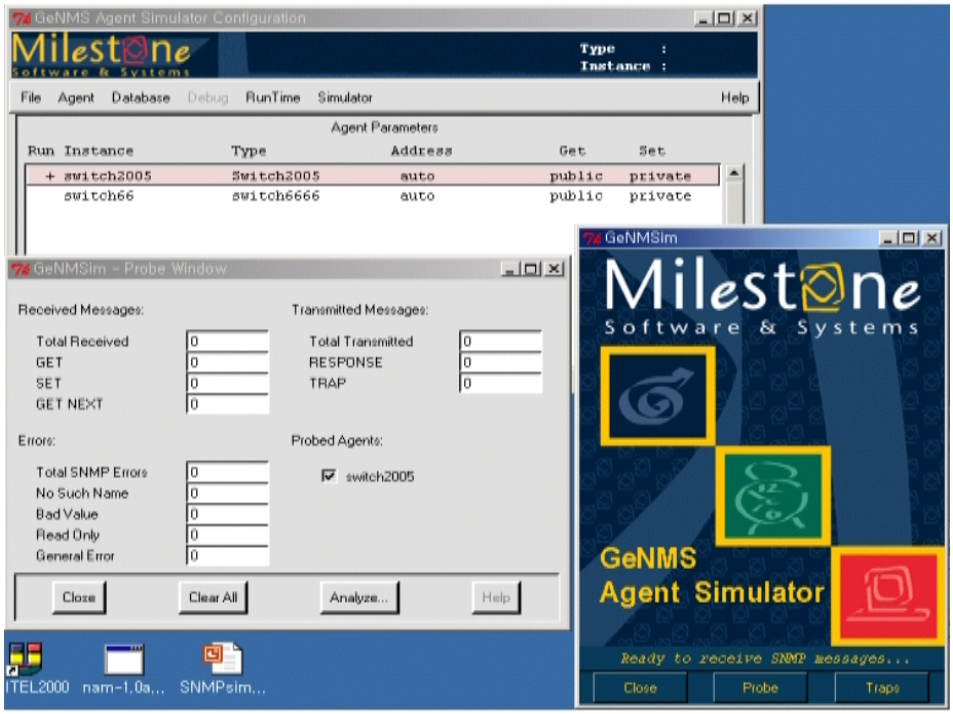

그림 22 GeNMSim 의 동작 화면

3. 고려사항

기존 SNMP 에이전트 시뮬레이터의 구조는 일반적으로 그림 23 와 같이 일반적으로 SNMP 프로토콜 처리기, MIB 파일 로더(loader), MIB 데이터 핸들러, 이벤트 로그 모듈, SNMP 트랩 발생기, 사용자 인터페이스 등으로 이루어져 있다.

그림 23 기본적인 SNMP 에이전트 시뮬레이터의 구조

또한 동시에 여러 개의 서로 다른 SNMP 에이전트를 하나의 SNMP 시뮬레이터에서 시뮬레이션 할 때는 위의 구조에 SNMP 에이전트 관리자가 추가가 되어야 한다. 이 관리자는 SNMP 에이전트들을 관리하고, SNMP 에이전트 시뮬레이션에 사용되는 시스템 리소스들을 관리하는 역할을 한다.

3.1 시스템 인터페이스

시스템 인터페이스는 SNMP 에이전트 시뮬레이터에서 운영체제에서 지원하는 시스템 API를 사용하기 위한 구성 요소이다. SNMP 에이전트 구현을 위한 Socket API[21]와 기타 파일 처리 관련 API, 사용자 인터페이스를 제공하기 위한 시스템 API등을 SNMP 에이전트 시뮬레이터에게 제공하는 역할을 한다. SNMP 에이전트는 일반적으로 UDP 포트번호 161 을 사용한다[17]. 그러나 시뮬레이션의 목적이나 시뮬레이션이 진행되는 호스트 시스템의 사정으로 다른 TCP/IP포트나 다른 이더넷(Ethernet) 어드레스를 사용할 경우도 있다. 따라서 이러한 설정이 사용자 인터페이스를 통하여 가능하도록 SNMP 에이전트 시뮬레이터를 설계 하여야 한다. 또한 SNMP 에이전트에서 사용되는 시스템 리소스를 관리하고, SNMP 에이전트 시뮬레이터의 환경 설정 정보를 시스템 정보 영역에 기록하여 매 기동시 마다 참조할 수 있도록 하는 역할을 한다.

3.2 SNMP 프로토콜 처리기

SNMP 프로토콜 처리기는 SNMP 프로토콜에 의한 NMS와 SNMP 에이전트간의 통신을 하기 위한 것으로 SNMP 에이전트 시뮬레이터의 핵심 요소이다. 시스템 인터페이스를 통해 전달된 SNMP message 내용을 해석해서 NMS나 SNMP 브라우저에서 요구하는 MO를 파악하고, 해당 MO의 정보를 SNMP message의 형태로 바꾸어서 시스템 인터페이스를 이용하여 다시 전송하는 역할을 한다. 이 프로토콜 처리기에서 지원하는 SNMP 버전은 SNMPv1, SNMPv2, SNMPv3 이 있다. SNMP 프로토콜 처리기는 ASN.1 encoding/Decoding 루틴, BER encoding/Decoding 루틴, SNMP message 포맷 처리 루틴등을 가지고 있다.

3.3 MIB 파일 로더

MIB 파일 로더(loader)는 SNMP MIB을 시뮬레이션에 사용될 수 있도록 메모리로 읽어 들이는 역할을 한다. ASCII 포맷의 MIB파일을 로드해서 사용하는 방법, 미리 text형태의 MIB파일을 바이너리 파일 형태의 데이터로 변환하여 보관한 후 시뮬레이션 시에 변환된 파일을 읽어들이는 방법, 그리고 SNMP " get next" PDU를 동작중인 다른 SNMP 에이전트에 보내 돌아오는 response를 이용해 MIB 데이터 파일을 구성하는 읽어 들이는 방법 이렇게 구현 방식에 따라 크게 3 가지로 분류 할 수 있다.

ASCII 포맷의 MIB파일을 SNMP 에이전트 시뮬레이터에서 직접 읽어 들이는 방법은 가장 흔히 사용되고 사용자 입장에서 볼 때 가장 손쉬운 방법이다. 그러나 이러한 방법은 SNMP 에이전트 시뮬레이터 자체에 MIB 컴파일러를 내장해야 하기 때문에 SNMP 에이전트 시뮬레이터 자체의 메모리 사용량이 커져서 동시에 시뮬레이션 할 수 있는 SNMP 에이전트의 수가 줄어들게 된다. 그리고 사용하는 MIB이 복잡해질수록 ASCII형태의 MIB파일을 읽어들여서 해석하는데 MIB 트리를 구성하고 MIB에서 MO정보를 추출하는데 시간이 많이 걸리기 때문에 SNMP 에이전트 시뮬레이터 에서 SNMP 에이전트를 실행시키는 것이 느려지게 된다.

이러한 문제를 해결하기 위하여 별도의 프로그램(흔히 MIB 컴파일러라고 부른다)을 통하여 ASCII 파일 형태의 MIB파일을 바이너리 파일 형태로 변환하는 방법이 사용된다. 이러한 방법은 별도의 MIB 컴파일 변환과정이 필요하다는 불편한 점이 있으나, 한번 MIB파일을 컴파일 해놓으면 시뮬레이션때마다 바로 사용할 수 있기 때문에 SNMP 에이전트 시뮬레이터의 실행 시간이 빨라지게 된다. 그리고 하나의 SNMP 에이전트안에 탑재된 MIB 모듈이 여러 개 일 경우 이러한 변환과정을 거치면서 파일 하나로 통합하여 관리할 수 있다는 장점이 있다. SNMP

에이전트 시뮬레이터에서 MIB 컴파일과 관련된 루틴들이 제거됨에 따라 메모리 사용량이 줄어들어 보다 더 많은 SNMP 에이전트를 동시에 시뮬레이션 할 수 있다.

마지막 방법은 SNMP 에이전트에 SNMP get Request PDU를 보내 돌아오는 응답을 분석해서 에이전트 자체에 내장된 MIB을 추정, 구성하고 더불어 실제 데이터를 얻어내는 방법이다. 이 방법은 SNMP 에이전트에 내장된 MIB파일을 구할 수 없을 때나 시뮬레이션 시에 장비에 설정된 실제 데이터를 이용하고자 할 때 많이 사용된다.

3.4 MIB 데이터 핸들러

MIB 데이터 핸들러는 시뮬레이션 도중에 특정 MO에 대한 SNMP request를 받았을 때 해당 MO의 데이터를 생성하여 SNMP response를 처리하는 루틴으로 전달하는 역할을 한다. 실제 SNMP 에이전트에서는 특정 MO의 데이터가 OS나 어플리케이션 내부 정보, 또는 하드웨어에 존재하지만, SNMP 에이전트 시뮬레이터는 모두 메모리에 존재한다. 따라서 SNMP 에이전트 시뮬레이터에서의 MIB 데이터 핸들러는 결과적으로 MO에 소속된 메모리상의 데이터 영역을 읽고, 쓰는 루틴이다. 이 MIB 데이터 핸들러의 동작 방법에 따라 분류를 해보면 MO의 데이터 타입에 따라 미리 정의된 값을 사용하는 방법, 사용자가 특정 MO의 데이터 값을 직접 입력하여 사용하는 방법, 실제로 존재하는 네트워크에서 수집한 데이터 테이블 을 사용하는 방법, 미리 정의되어진 내부 지원 수학 함수에 의해 생성된 데이터를 사용하는 방법, SNMP 에이전트 시뮬레이터상에서 지원되는 스크립트 언어를 사용하여 에이전트의 동작을 프로그래밍하여 그 결과를 이용하는 방법 등이 있다.

MO의 데이터 타입에 따라 미리 정의된 데이터를 사용하는 경우는 MIB데이터 타입에 따라 정해진 SNMP 에이전트 시뮬레이터 내부의

기본 값을 시뮬레이션에 사용하는 것으로 구현이 간단하기 때문에 거의 모든 SNMP 에이전트 시뮬레이터에서 기본적으로 제공된다.

사용자가 특정 MO의 데이터 값을 직접 입력하여 사용하는 방법은 특정 MO의 데이터를 사용자가 사용자 인터페이스나 설정 파일을 통해 변경하도록 하는 방법이다.

실제로 존재하는 네트워크에서 수집한 데이터 테이블을 사용하는 방법은 실제 가동중인 네트워크 장비에 탑재된 SNMP 에이전트의 내부에 탑재된 SNMP 에이전트에서 사용되는 MIB과 MIB값을 읽어서 시뮬레이션에 사용하는 것으로 NMS개발이나 디버깅 시에 아주 유용하게 사용할 수 있는 기능이다.

시뮬레이터 내부에 구현되어 기본적으로 제공되는 함수에 의한 MO의 데이터 생성 방법은 간단한 수학 함수를 이용해서 OID에 대응하는 SNMP 에이전트내부의 데이터 값을 규칙적으로 변화시키는데 유용하다. 보통 Increment, Decrement, Random, TimeStamp 등의 내장 함수를 제공하며, 이 함수들을 SNMP GetRequest/ GetNextRequest/ GetBulkRequest 요청을 받아서 처리하기 전과 처리한 후에 실행되도록 설정할 수 있다. 이러한 방법을 통하여 규칙성 있게 변하는 MO값을 사용하여 NMS 동작을 시험할 수 있다.

SNMP 에이전트 시뮬레이터상에서 지원되는 스크립트 언어를 사용하여 SNMP 에이전트의 동작을 프로그래밍하여 그 결과를 이용하는 방법은 좀더 복잡하고 사실적인 동작을 시뮬레이션 하기 위한 것이다. 사용자의 프로그램에 의하여 시뮬레이션 되어지는 SNMP 에이전트 내부의 동작을 좀 더 현실성 있게 시뮬레이션 하기 위한 것으로 실제 장비 내에서 발생하는 특정 상황을 쉽게 재현할 수 있다. 예를 들어 GeNMSim[6] 제품은 Tcl/Tk 언어를 사용하는 callback function 방법을 사용해서 시뮬레이션을 할 수 있게 되어 있으며, PreInit, PostInit,PreGet, PostGet, PreSet, PostSet등 NMS에서 SNMP 에이전트에게 데이터를 요구

하거나 (PreGet, Post Get), 데이터 변경을 지시할 때 (PreSet, PostSet) 실행되는 수동적인 callback 함수를 사용한다. MIMIC 은 Tcl/Tk 을 사용해서 사용자 정의 시뮬레이션을 지원하지만 일종의 batch job성격이 강하다. Tcl/Tk 에서 MIMIC 자체의 shell API를 불러 쓸수 있어서 MIMIC 프로그램 자체를 제어할 수 있다. 아래는 MIMIC에서 사용되는 시뮬레이션 스크립트 중의 하나를 예시한 것이다.

```
proc add_agent s {start max} {
    set mibs "linux-2.0.x.random,RFC1213-MIB,
1 linux-2.0.x.random,IF-MIB,
1 linux-2.0.x.random,RFC1414-MIB,
1 linux-2.0.x.random,HOST-RESOURCES-MIB,1"
    for {set i $start} {$i <= $max} {incr i} {
            mimic agent assign $i
            mimic agent add 192.9.209.$i $mibs
    }
}
```

위의 시뮬레이션 스크립트는 일련번호 start 부터 max까지 루프를 돌면서 mibs라는 문자열에 포함된 MIB들을 탑재한 SNMP 에이전트 시뮬레이션을 수행하라는 것이다.

3.5 SNMP 에이전트의 메모리 사용

다수의 SNMP 에이전트를 동시에 시뮬레이션 하기 위해서는 SNMP 에이전트가 사용하는 메모리를 최소화 할 필요가 있다. 보통의 PC에서 구현된 SNMP 에이전트의 경우 평균 메모리 사용량이 1MB(RedHat Linux 7.0 기준)에 육박한다. 이러한 메모리를 사용하는 SNMP 에이전트를 그대로 시뮬레이션에 사용하면 시스템 메모리의 한계로 인하여 동시에 시뮬레이션 할 수 있는 SNMP 에이전트의 개수가 줄어들게 된다. 따라서 SNMP 에이전트 시뮬레이터에 사용될 SNMP 에이전트 모듈은 아주 작은 메모리를 사용해야 하며, 최소한의 CPU 사용률을 가져야 한다.

본 논문에서는 이러한 분석내용을 바탕으로 SNMP 에이전트 시뮬레이터를 설계하고 구현하는 것에 대하여 다루고자 한다

4. SNMP 에이전트 시뮬레이터의 설계

우리의 SNMP 에이전트 시뮬레이터는 크게 SNMP 에이전트(SNMP Agent) 모듈과 SNMP 에이전트 매니저(SNMP Agent manager) 모듈로 이루어진 SNMP 에이전트 시뮬레이터 본체와 그리고 외장 MIB 컴파일러 프로그램으로 이루어져 있다.

4.1 SNMP 에이전트 시뮬레이터

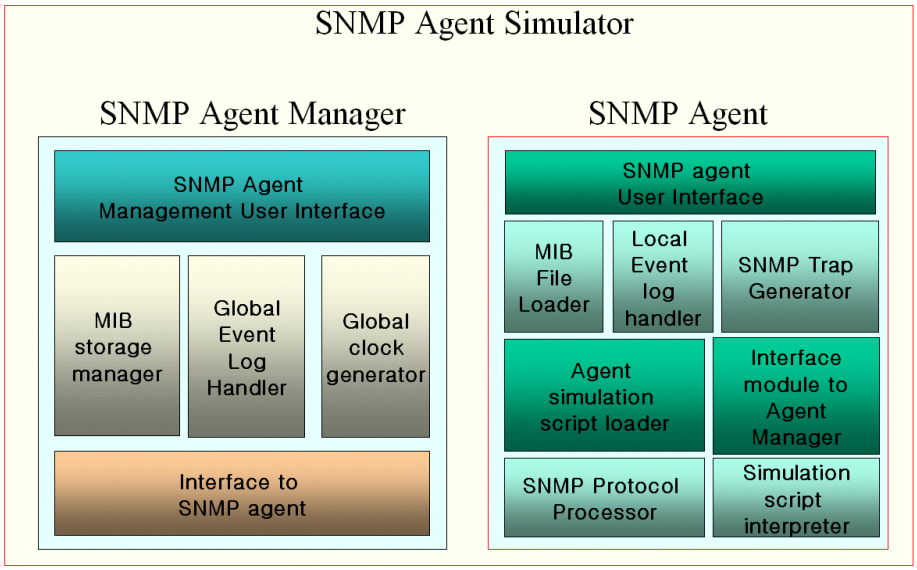

그림 24 구현한 SNMP 에이전트 시뮬레이터의 구조도

본 논문에서 구현한 SNMP 에이전트 시뮬레이터의 구조는 그림 24 와 같다. 각 SNMP 에이전트 객체를 시뮬레이션 하는 SNMP 에이전트 모듈과 다수의 SNMP 에이전트들을 관리하는 SNMP 에이전트 매니저로 구성이 되어 있다.

4.1.1 SNMP 에이전트 모듈

SNMP 에이전트 모듈은 SNMP 프로토콜 처리기, MIB 파일 로더(MIB file loader), SNMP 트랩 발생기(SNMP trap generator), 로컬 SNMP 에이전트 이벤트 로그 핸들러, SNMP 에이전트 시뮬레이션 스크립트 처리 프로세스, SNMP 에이전트 오브젝트 사용자 인터페이스, 그리고 SNMP 에이전트 매니저와 통신하는 인터페이스 모듈로 구성이 되어 있다.

SNMP 프로토콜 처리기는 SNMP를 사용한 SNMP 에이전트 오브젝트와 SNMP 매니저 또는 NMS간의 통신을 처리한다. Net-SNMP 라이브러리를 사용하여 SNMP 에이전트 동작에 필요한 함수들만을 모아 하나의 C++ class object(SNMP agent class)를 구성하였으며, 이로 인해 최소한의 메모리를 사용하는 SNMP 에이전트 구현이 가능하였다. C++ class object를 구성한 이유는 하나의 SNMP 에이전트 시뮬레이터에서 다수의 가상 SNMP 에이전트를 구동함에 있어서 OOP(Object Oriented Programming)의 개념을 도입함으로써 프로그램의 효율성과 개발의 용이성을 높이기 위한 것이다. Net-SNMP에서 사용되는 *.conf 파일에 의한 동작 파라메터 설정 방식을 다수의 SNMP 에이전트 시뮬레이션을 동시에 실행시킬 때 각각의 SNMP 에이전트마다의 서로 다른 동작환경을 보장하고 쉽게 환경 설정을 하기 위해서 SNMP 에이전트 클래스 안의 private 멤버 변수를 이용하도록 수정하였다.

MIB 파일 로더(loader)는 SNMP 에이전트 시뮬레이션에 사용할 MIB 데이터를 파일로부터 읽어서 메모리에 적재하는 역할을 한다. MIB 파일 로더는 SNMP MIB 컴파일러(compiler)를 통해 변환과정을 거친 바이너리 형태의 MIB 데이터나 또는 SNMP 프로토콜의 “get_next” PDU를 실제 장비의 SNMP 에이전트에게 보내서 응답으로 받은 실제 장비로부터 획득해서 구성한 바이너리 파일 형태의 MIB 데이터를 읽어들일 수 있다.

SNMP 트랩 발생기(trap generator)는 사용자로부터의 입력 또는

SNMP 에이전트 시뮬레이션 스크립트의 동작의 결과에 따라 SNMP 트랩을 발생시키는 역할을 한다.

로컬 에이전트 이벤트 로그 핸들러(Local 에이전트 Event Log Handler)는 SNMP 에이전트에서 발생하는 이벤트를 시간과 함께 기록하는 역할을 하여 개발중의 디버깅 용도로 사용할 수 있게 한다. 기록되는 이벤트의 종류에는 SNMP request와 response내용, 사용자의 입력 등이 있다. 그리고 로컬 에이전트 이벤트 로그 핸들러는 발생한 이벤트를 SNMP 에이전트 매니저로 전달하여 시뮬레이터상에서 일어나는 전체적인 동작 상황에 대한 기록을 남길 수 있도록 한다.

SNMP 에이전트 사용자 인터페이스는 현재 동작중인 SNMP 에이전트의 상태를 표시해 주고 사용자로부터의 제어 명령을 SNMP 에이전트에 전달해 주는 역할을 한다.

에이전트 시뮬레이션 스크립트 처리 프로세스는 SNMP 에이전트에서 사용되는 MIB 데이터 값이 실제 운영되는 장비와 유사한 패턴을 따라 변경되게 하기 위해서 시뮬레이터 사용자가 간단한 스크립트 언어를 사용하여 실제의 SNMP 에이전트 내부의 MIB 데이터 값 변경과 트랩 발생기의 동작을 제어할 수 있도록 하기 위한 것이다. 이러한 시뮬레이션 동작의 예를 들면 다음과 같다.

if (1.3.6.1.4.1.2021.14.1.1.1 >= 3000) and (1.3.6.1.4.1.2021.14.1.1.2 >= 100000)) then

Generate Trap and Set 1.3.6.1.4.1.2021.14.1.1.3 to 30

이 시뮬레이션 스크립트는 동시에 OID 1.3.6.1.4.1.2021.14.1.1.1 값이 3000 이상이 되고 OID 1.3.6.1.4.1.2021.14.1.1.2 데이터 값이 100000 이상이면 트랩을 발생시키고 OID 1.3.6.1.4.1.2021.14.1.1.3 데이터를 30 으로

설정하라는 것이다. 이러한 복잡한 시뮬레이션 과정을 가능하게 함으로써 강력한 시뮬레이션을 할 수 있다.

4.1.2 SNMP 에이전트 매니저 (SNMP agent Manager)

SNMP 에이전트 관리자는 각 SNMP 에이전트의 동작을 통제하는 역할과 시뮬레이션에 사용되는 MIB을 관리하는 역할, 그리고 복수의 SNMP 에이전트들에서 발생한 이벤트를 종합하여 시간 순서대로 기록하는 역할을 한다. 그리고 동시에 시뮬레이션 되는 SNMP 에이전트끼리 이벤트 동기화가 필요할 경우 사용되는 기준 클럭을 발생시키는 Global clock generator가 내장되어 있다. 이 Global clock generator에서 발생된 기준 클럭은 각 SNMP 에이전트들의 시뮬레이션 기준 클럭이 되며, 이 클럭의 주기를 조정해서 시뮬레이션 과정에서 step by step ,trace등의 동작을 시킬 수 있다.

4.1.3 MIB Compiler

SNMP 에이전트에서 사용될 SNMP MIB을 SNMP 에이전트 시뮬레이터에서 읽을 수 있는 형태로 변경하는 역할을 한다. 본 연구에서 개발한 SNMP 시뮬레이터에서는 외부에 별도로 존재하는 SNMP MIB 컴파일러를 사용하였다. 이렇게 외장 MIB 컴파일러를 사용하는 이유는 SNMP 에이전트 시뮬레이터 자체의 메모리 사용량을 줄여서 동시 시뮬레이션 되는 SNMP 에이전트의 수를 늘리기 위한 것이다. 본 연구에서는 SNMP MIB을 메모리로 읽어 들여서 해석한 뒤, ..cmd 파일과 .dmd 파일을 생성한다. “ .cmd” 파일은 SNMP 에이전트 시뮬레이션 자체에 관련된 정보를 가지고 있다. 그리고 “ .dmd”파일은 SNMP 에이전트 user interface에 MIB파일에 포함되어 있는 SNMP MIB파일의 부가적인 정보(SNMP 에이전트 유저 인터페이스에서 사용하는 정보)를 가지고 있다. 이렇게 변환된 MIB파일들은 시스템에 정의된 디렉토리에 복사된다.

이렇게 함으로써 MIB 파일 관리가 쉬워진다.

4.1.4 동작 원리

사용자가 SNMP 에이전트 시뮬레이터를 구동하여 시뮬레이션에 사용될 MIB을 선택하고 SNMP 에이전트에서 사용되는 파라메터(SNMP 에이전트의 사용 IP 어드레스, UDP 포트 번호, SNMP Community)를 입력하고 시작 버튼을 누르면 SNMP 에이전트 클래스 인스턴스를 새로 생성하고 생성된 SNMP 에이전트 클래스를 수행할 쓰레드(Thread)를 생성시켜 구동시킨다. SNMP 에이전트 실행을 중단시킬때는 SNMP 에이전트 Manager UI로부터의 입력에 따라 Windows OS에서 제공하는 시스템 event를 발생시켜서 SNMP 에이전트 쓰레드에 전달한다. 시뮬레이션 되는 SNMP 에이전트의 CPU 사용량을 최소화 하기 위해서 NMS로 부터의 요청이 없을때는 Thread의 상태를 suspend모드로 진입시켰다가 요청이 들어왔을 때 wake-up되게 설계를 하였다. 따라서 다수의 SNMP 에이전트가 동시에 시뮬레이션 되더라도 아주 원할하게 동작하였다.

5. 구현

이 장에서는 본 논문에서 제안한 SNMP 에이전트의 구조를 구현한 내용에 대하여 다루고 있다.

5.1 SNMP 에이전트 시뮬레이터의 구현 환경

구현한 SNMP 에이전트 시뮬레이터는 Microsoft Windows NT 4.0[22]과 Microsoft Windows 2000[23]에서 동작하도록 설계되어 있다. 시스템의 구현 과정은 Microsoft사에서 제공한 표준 Win32 API를 사용하여 구현하였다. 따라서 SNMP 에이전트 시뮬레이터 자체는 Microsoft Win32 API를 지원하는 어떤 OS에서도 동작하지만 Win95, Win98, Windows Me에서는 시스템 자체의 리소스 핸들의 개수 제한으로 인하여 동시에 다수의 SNMP 에이전트 시뮬레이션을 수행 할 경우에는 시스템 리소스 부족으로 인하여 제대로 실행이 되지 않았다. 따라서 Microsoft Windows NT 4.0과 Microsoft Windows 2000 등 MS사의 서버제품군의 OS만을 대상 OS로 선정 하였다.

5.2 개발 툴 및 사용 라이브러리

개발툴은 Microsoft Visual C++ 6.0[24]을 사용하였다. Microsoft Visual C++는 Microsoft사의 OS에서 어플리케이션을 개발하기 위한 필수 도구이다. Microsoft Visual C++ 6.0 에서 제공하는 MFC 클래스 라이브러리와 Win32 표준 API를 같이 사용하였다. MFC(Microsoft Foundation Class) 라이브러리는 응용프로그램 개발에 사용될 수 있는 클래스들의 모음인데, 자주 쓰이는 이미 작성되어 있는 코드들을 제공함으로써 개발 소요 시간을 절약해주고, 응용프로그램을 개발하기 위한 전반적인 틀을 제공한다. SNMP 에이전트 시뮬레이터의 사용자 인터페이스 관련 루틴들은 개발 시간의 단축을 위하여 MFC 라이브러리 루틴을 사용하였으며, 기타

시스템 관련 함수들은 최대한의 성능을 이끌어 내기 위하여 low-level의 Win32 API를 사용하였다. 또한 SNMP 프로토콜 처리를 위해서 Net-SNMP 4.1.2 버전[12]을 사용하였다. 기존에 제공되는 Net-SNMP 라이브러리에서 지원하는 SNMP 에이전트에서의 MIB파일 로딩 방법이 MIB2C 프로그램을 통한 MIB을 C언어 소스 코드로 변경하는 방법이나 프로그래머가 소스코드 상에서 직접 MIB정보를 입력하는 방법이었기 때문에 SNMP 에이전트의 코딩(coding)과 컴파일(Compile) 과정에서 에이전트에서 사용되는 MIB이 결정되었다. 이러한 동작방법은 SNMP 에이전트 시뮬레이터에서처럼 수시로 사용하는 MIB이 바뀌는 경우에는 적합하지 않았다. 따라서 Net-SNMP 라이브러리를 수정하여 바이너리 파일 형태의 MIB을 run-time에 읽어서 동작할 수 있도록 수정을 하여 원하는 용도에 맞도록 최적화 시켰고, MIB 데이터 핸들링 루틴을 수정하였다. 또 Net-SNMP 라이브러리의 API를 사용하여 SNMP 에이전트 클래스를 구현하였다. 또한 동시에 시뮬레이션 되어 지는 SNMP 에이전트의 개수를 최대화 하기 위해 SNMP 에이전트 클래스가 사용하는 메모리를 최대한 줄이는데 중점을 두었다. 따라서 현재는 하나의 SNMP 에이전트를 실행시키는데 69KB의 코드가 더 요구될 뿐이다(MIB 데이터에 소요되는 메모리 크기는 제외). 또한 Microsoft Windows 환경에 최적화 하기 위하여 기존 Net-SNMP 라이브러리에서 사용되는 blocking socket I/O 함수를 WinSock2[9]에서 지원하는 non-blocking socket I/O함수를 사용하도록 변경하였다. 문제가 되었던 부분은 데이터를 주고받는 것에 관련된 소켓 라이브러리의 블록킹 모드의 함수들이다. 소켓 라이브러리에서 사용되는 함수들을 호출하게 되면 그 작업이 끝나기 전까지는 블록킹 상태가 된다. 따라서 Windows OS의 경우는 블로킹 상태에서는 응용프로그램의 동작이 원할하게 이루어 지지 않는다. 따라서 Windows OS에서의 문제점을 해결하기 위하여 윈속에서는 이러한 문제를 해결하기 위해서 다양한 종류의 API함수들을 제공하고 있다. 본 구현에서는 WinSock의 32 비트 버전인 WinSock 2 에서 제공하는

asynchronous notification과 overlapped I/O 메커니즘을 이용하였다. 또한 각각의 SNMP 에이전트 오브젝트 클래스와 멀티 쓰레드(Multi thread)를 사용하여 다수의 SNMP 에이전트 시뮬레이션을 가능하게 하였다. 멀티 쓰레드를 사용함으로써 각각의 SNMP 에이전트 오브젝트를 멀티 프로세스 환경에서 동작시키는 것보다 적은 시스템 리소스를 소모할 수 있었다. 따라서 하나의 SNMP 에이전트 시뮬레이터 안에서 더 많은 SNMP 에이전트를 구동 시킬 수 있었다.

5.3 개발한 SNMP 에이전트 시뮬레이터의 동작 모습

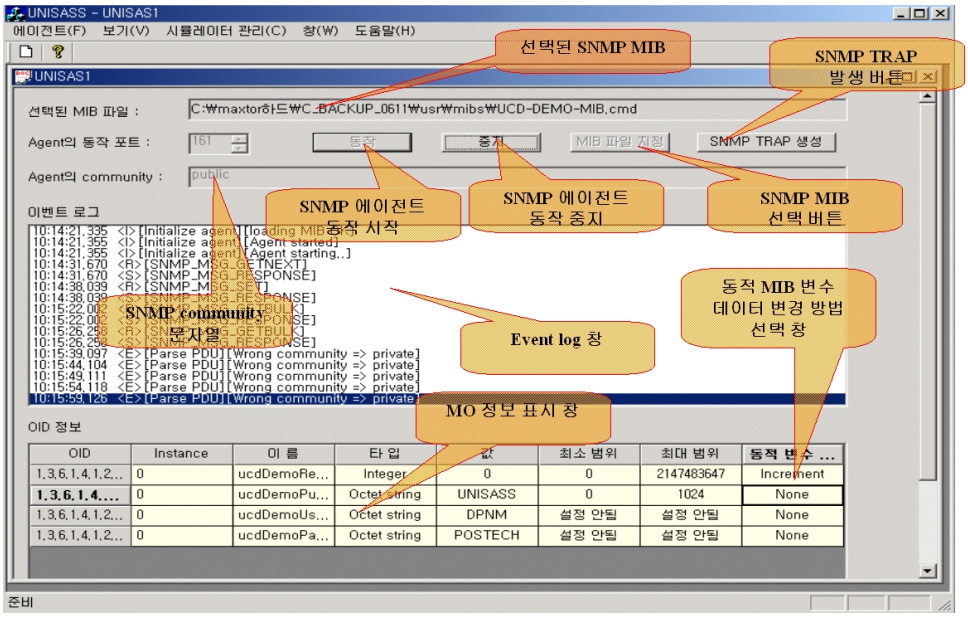

그림 25 개발한 SNMP 에이전트 시뮬레이터의 화면

그림 25 는 본 연구에서 구현한 SNMP 에이전트 시뮬레이터의 화면 구성도이며 시뮬레이션 되는 SNMP 에이전트 UI를 설명하고 있다. 그림 26 은 SNMP 에이전트 시뮬레이터에서 사용될 MIB파일을 선택하는 화면이다. 그림 26 에서는 미리 컴파일된 MIB 데이터 파일(확장

자 : .cmd) 을 선택한 것을 보여 준다.

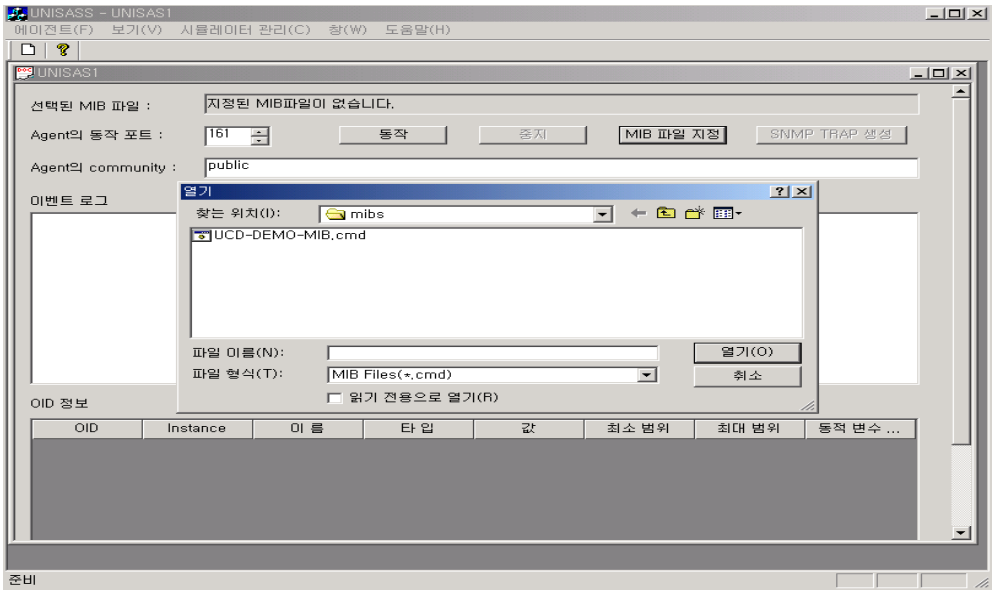

그림 26 시뮬레이션에 이용할 MIB 파일 선택 화면

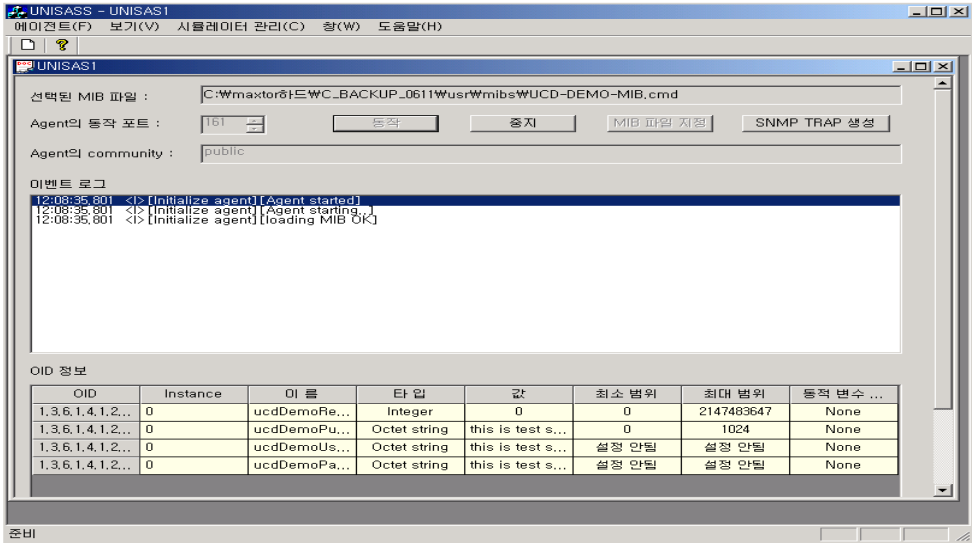

그림 27 선택된 MIB 을 이용한 SNMP 에이전트 시뮬레이션

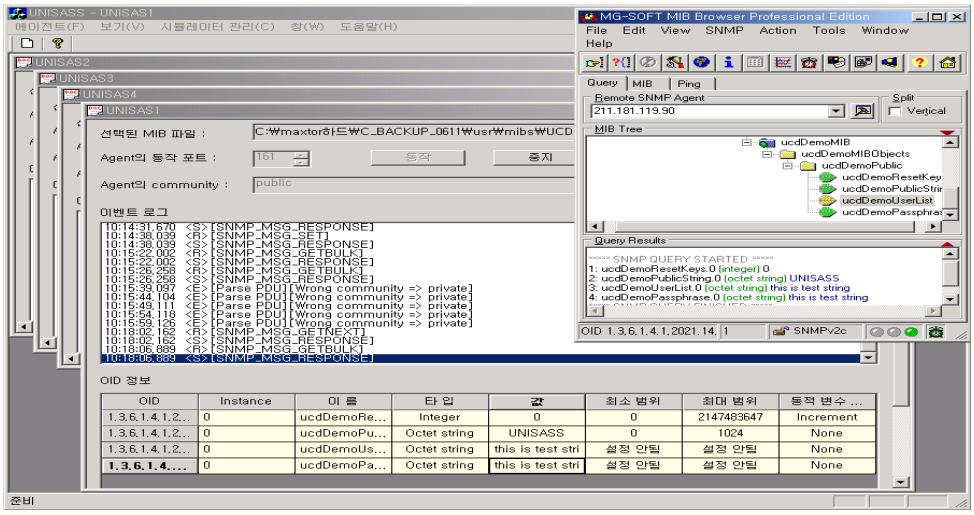

그림 28 다수의 SNMP 에이전트 시뮬레이션을 실행하는 모습

그림 27 에서는 선택된 MIB을 이용해서 하나의 SNMP 에이전트 시뮬레이션이 수행되는 것을 보여주며 그림 28 에서는 동시에 여러 개의 SNMP 에이전트를 시뮬레이션 하면서 MG-SOFT MIB Browser를 사용해서 시뮬레이션 되는 SNMP 에이전트의 동작을 확인하는 것을 보여준다.

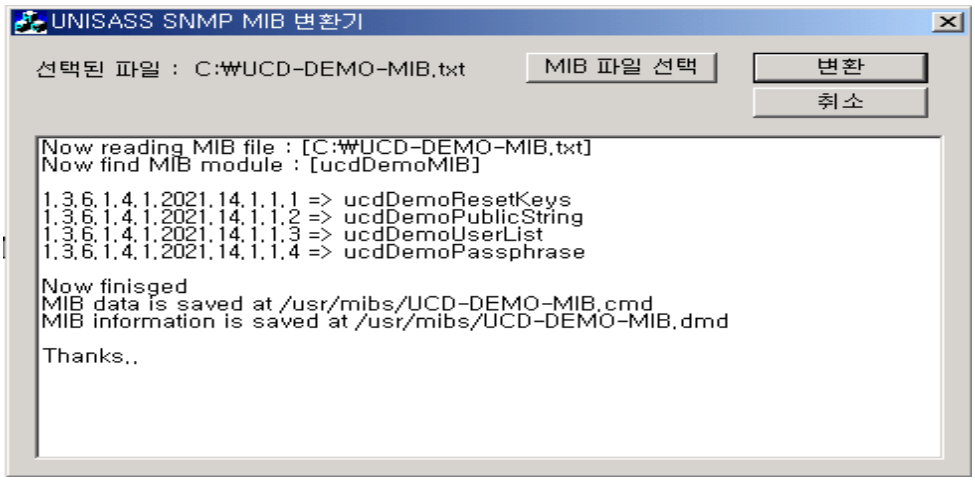

그림 29 SNMP MIB Compiler 의 동작 모습

그림 28 에서 MIB 파일 정보창과 MG-SOFT MIB browser를 사용해서 출력한 데이터가 일치하는 것을 확인할 수 있다. 그림 29 는 SNMP 에이전트 시뮬레이션에 사용될 MIB파일을 읽어 들여서 컴파일 후 구현한 SNMP 에이전트 시뮬레이터에서 사용될 수 있는 포맷으로 바꾸는 SNMP MIB compiler의 동작 모습을 보여준다.

6. 결론 및 향후 과제

상업적으로 판매되는 많은 네트워크 장치들이 SNMP 에이전트를 내장하고 있고 이러한 SNMP 에이전트의 기능을 이용한 NMS의 개발이 늘어나고 있다. SNMP 에이전트 자체, 또는 SNMP 에이전트를 이용한 NMS를 개발하거나 시험 및 개선을 할 때 실제의 네트워크 환경에서 진행하는데는 공간적, 비용적, 시간적 제한이 있다. 따라서 이런 어려움을 해결하기 위하여 SNMP 에이전트 시뮬레이터의 사용이 늘어나는 추세이다. 이 논문에서는 기존 SNMP 에이전트 시뮬레이터의 구조를 분석하고, 개선된 SNMP 에이전트 시뮬레이터의 구조를 제시하였다. C++ 언어를 사용하여 SNMP 에이전트 class를 설계하면서 최소한의 메모리를 사용하도록 net-SNMP library를 수정하였으며, 이 작업으로 인하여 하나의 SNMP 에이전트를 시뮬레이션 할 때 시뮬레이션을 위한 데이터 공간을 제외한 코드부분의 메모리 사용량이 69KB로 매우 작게 개발하였다. 다수의 SNMP 에이전트 시뮬레이션 결과를 종합 분석하기 위하여 global event log handler를 통해 event들을 serialization시켜서 저장하기 때문에 시뮬레이션 결과 분석을 보다 용이하게 하였다. 이 SNMP 시뮬레이터는 NMS개발과정에서 test-bed로 사용할 수 있고, SNMP 에이전트개발 과정에서 MIB 설계를 검증하는 용도로 사용할 수 있다. 또한 아주 작은 메모리를 사용하는 SNMP 에이전트 클래스를 구현하여 Embeded OS에서도 사용이 가능하다. 그리고 외부에서 MIB compile 결과와 back end process definition table 만 지정해주는 것 만으로도 SNMP 에이전트 개발이 가능하므로 Instant SNMP 에이전트 development toolkit으로 사용할 수 있다.

향후 과제는 우리가 개발한 SNMP 에이전트 시뮬레이터가 실제 시뮬레이션 시에 얼마나 쉽게 사용되고 효율적인지를 검증하고 개선하는 것이다. 또 보다 사실적인 SNMP 에이전트의 시뮬레이션을 위한 SNMP 에이전트들과 SNMP 에이전트가 위치한 네트워크 환경까지 시뮬

레이션 할 수 있게 하기 위해 SNMP 에이전트 시뮬레이터와 네트워크 시뮬레이터를 통합하는 방법을 연구하고 구현하는 것도 고려하고 있다.

7. 참고 문헌

[1] Gambit Communications,"MIMIC simulator - problems," http://www.gambitcomm.com/site/products/mimic.htm#problems.

[2] Gambit Communications ,"MIMIC – SNMP Agent simulation tool," http://www.gambitcomm.com.

[3] Simple Soft Inc "Simple Agent pro, Simple Agent – SNMP Agent Simulator," http://www.smplsft.com.

[4] Milestone Software & Systems, "GeNMSim," http://ftp.sage.usenix.org/publications/library/proceedings/tcl97/full_papers/margolin/margolin_html/SimTclWSFinal.html.

[5] J. Case, M. Fedor, M. Schoffstall and C. Davin, "The Simple Network Management Protocol (SNMP)," RFC 1157, May 1990.

[6] ISO, " ISO - International Organization for Standardization ," http://www.iso.ch/iso/en/ISOOnline.frontpage.

[7] Network Sorcery. Inc., "ICMP, Internet Control Message Protocol ," http://www.networksorcery.com/enp/protocol/icmp.htm.

[8] IAB, "Internet Architecture Board Home Page," http://www.iab.org.

[9] C. Partridge , "RFC1021 - THE HIGH-LEVEL ENTITY MANAGEMENT SYSTEM (HEMS)," Internet Working Group, http://aurora.rg.iupui.edu/doc/rfc/rfc-html/rfc1021.html.

[10] Davin, J., J. Case, M. Fedor, and M. Schoffstall, "A Simple Gateway Monitoring Protocol", RFC 1028, Proteon, University of Tennessee at Knoxville, Cornell University, and Rensselaer Polytechnic Institute, November 1987.

[11] Stallings, William. SNMP, SNMPv2, and CMIP. Don Mills: Addison-Wesley, 1993.

[12] NYSERNet, "NYSERNet network,", http://nysernet.org/.

[13] SURAnet, "SURAnet = Southeastern Universities Research Association network," http://www.sul.com.br/~mig/link_s/suranet.htm.

[14] IETF, "The Internet Engineering Task Force," http://www.ietf.org.

[15] ITU-T, "The ITU Telecommunication Standardization Sector (ITU-T)," http://www.itu.int/ITU-T/.

[16] Information processing systems - Open Systems Interconnection, "Specification of Abstract Syntax Notation One(ASN.1)", International Organization for Standardization, International Standard 8824, December 1987.

[17] IANA, “IANA | Protocol/Number Assignments Directory,” http://www.iana.org.

[18] Information porcessing systems - Open Systems Interconnection,"Specification of Basic Encoding Rules for Abstract Syntax Notation One(ASN.1)", International Organization for Standardization, International Standard 8825, December 1987.

[19] www.snmp.com , “SNMPv3 White Paper,” http://www.snmp.com.

[20] Tcl/Tk, “SourceForge: Project Info - Tcl,” http://sourceforge.net/projects/tcl

[21] The WinSock channel at Stardust.com, “Introduction: What is WinSock?,” http://www.winsock.com/winsock/intro.htm.

[22] Microsoft, “The Windows NT 4.0”, http://www.microsoft.com/ntserver/nts/default.asp.

[23] Microsoft , “The Windows 2000 Server Family,” http://www.microsoft.com/windows2000/guide/servers.asp.

[24] Microsoft, “icrosoft Visual Studio - Product Overview,” http://msdn.microsoft.com/vstudio/prodinfo/overview.asp.

[25] Net-SNMP project, “Welcome To The NET-SNMP Home Page,” http://net-snmp.sourceforge.net.

[26] AdventNet Inc “Simulation toolkit,” http://www.adventnet.com.

[27] IETF “Management Information Base Network Management of TCP/IP based internets: MIB-II,” RFC 1213, http://www.ietf.org.

[28] Marshall T. Rose. “The Simple Book: An Introduction to Internet Management (2nd edition),” Prentice Hall, 1993. ISBN 0-13-177254-6.

[29] Marshall T. Rose, Keith McCloghrie. “How to Manage Your Network Using SNMP,” Prentice Hall, 1995. ISBN 0-13-141517-4.

[30] William Stallings. “SNMP, SNMPv2, and CMIP. Practical Guide to Network-Management Standards,” Addison-Wesley, 1993, ISBN 0-201-63331-0.

[31] Cerf, V., "IAB Recommendations for the Development of Internet Network

Management Standards", RFC 1052 , April 1988.

[32] Cerf, V., "Report of the Second Ad Hoc Network Management Review Group", RFC1109, August 1989.

[33] Rose, M., and K. McCloghrie, "Structure and Identification of ManagementInformation for TCP/IP-based internets", RFC 1155, May 1990.

[34] McCloghrie, K. and M. Rose, "Management Information Base for Network Management of TCP/IP-based internets", RFC 1066, TWG, August 1988.

[35] Satz, G., "Connectionless Network Protocol (ISO 8473) and End System to Intermediate System (ISO 9542) Management Information Base", RFC1162, cisco Systems, Inc., June 1990.

[36] Jacobson, V., "Congestion Avoidance and Control", SIGCOMM 1988, Stanford, California.

[37] Rose, M., and K. McCloghrie, Editor, "Concise MIB Definitions", RFC 1212, Performance Systems International, Hughes LAN Systems March 1991.

[38] IETF,“Coexistence between Version 1 and Version 2 of the Internet-standard Network Management Framework,” RFC 1908, http://www.ietf.org.

[39] IETF “A Simple Network Management Protocol,” RFC 1157, http://www.ietf.org.