액티브 네트워크 기술을 이용한 새로운 망관리 기법

(A Novel Network Management Approach using Active Network Technology)

이 병 기 † 조 국 현 ††

(ByeongKi Lee) (KukHyun Cho)

요 약 전통적인 망 관리 접근 방법은 중앙의 관리 시스템이 예외적인 상황을 찾고자 관리되는 노드들을 반복적으로 폴링 하면서 달성된다. 이러한 중앙집중적인 관리 패러다임은 관리되는 노드들의 수와 복잡도가 증가함에 따라 한계에 부딪히고 있다. 관리 시스템은 관리대상 정보들이 이전과 똑같은 상태를 유지함에 따라 많은 양의 중복된 정보로 폭주하게 된다. 이러한 수동적인 망 관리 방법은 확장이 불가능하고, 트래픽 증가로 인해 많은 비용이 들기 때문에 더욱 신속한 접근 및 확장 능력을 가진 기술을 적용하는 것이 필수적이다.

액티브 네트워크는 사용자에 의해 커스터마이즈된 응용이나 프로토콜들을 망 노드들에 동적으로 삽입할 수 있도록 함으로서 망 노드들이 새로운 사용자 요구나 서비스에 신속하게 대응할 수 있도록 한다. 본 논문에서는 이러한 액티브 네트워크 기술을 망 관리에 적용함으로서, 관리 노드들이 관리 시스템으로부터의 수동적인 영역에서 벗어나 능동적으로 자신의 문제에 대처할 수 있고, 또한 새로운 관리 요구사항들에 신속하게 대응할 수 있도록 하는 새로운 망 관리 접근 방법에 대해 기술한다.

Abstract Traditional network management is achieved by having management system repeatedly poll the managed devices, looking for anomalies. This centralized management paradigm has been stretched to its limits by the number and complexity of managed nodes. Management system inundated with large amounts of redundant information when management components are in the same state they were in previously. This passive network management solution does not scale and is not cost effective due to traffic increase. Therefore it is essential that network management employ techniques with more immediate access and more ability to scale.

Active networks provides a mechanism to rapidly adapt the network to a new user requirements and services by allowing their users to injected customized applications and protocols into the nodes of the network. In this paper, we introduce a novel network management approach based on the active network technology in which managed node itself actively cope with its problem break away from passive area of management system and rapidly adapt to a new requirements for network management.

1. 서 론

현재의 망에서 중간의 노드들은 종단의 노드들에 비해 폐쇄적이며, 수직적으로 통합된 시스템이다. 그들의 기능은 표준화 과정을 통해 결정되며, 장비 판매자에 의해 이미 구성되어 내장된 소프트웨어에 의해 구현된다. 새로운 망 프로토콜이나 서비스의 개발은 표준화 위원회에 의해 만들어지지만, 이들 위원회의 모든 구성원이 만족하는 솔루션에 도달하는 데에는 상당히 오랜 시간이 걸린다. 또한 새로운 프로토콜이나 서비스가 만들어졌다 할지라도, 장비 판매자들이 이러한 새로운 기술을 자사 장비에 적용시키는 것은 별개의 부수 작업이며, 또 하나의 문제는 기존의 장비들이 새로운 프로토콜을 지원하지 않기 때문에 발생되는 호환성 문제이다.

액티브 네트워크(Active Network)는 이러한 문제점

· 이 논문은 1998년도 광운대학교 교내학술연구비 지원에 의해 연구되었음.

† 학생회원 : 광운대학교 컴퓨터과학과
lbk@cs.kwangwoon.ac.kr
†† 정 회 원 : 광운대학교 컴퓨터과학과 교수
khcho@cs.kwangwoon.ac.kr

논문접수 : 2000년 5월 26일
심사완료 : 2000년 12월 26일

을 해결하고자 시도하고 있다. 액티브 네트워크란 기존 망에서의 라우터나 스위치와 같은 중간 노드들이 단순히 패킷의 헤더만을 처리하는 것에 한 걸음 더 나아가 사용자가 패킷에 프로그램 코드와 데이타를 함께 넣어 전송하고 중간 노드에서는 이를 처리할 수 있는 환경을 말한다. 또한 중간 노드에서는 단순히 패킷의 경로를 설정하고 전달하는 기능을 담당하고, 에러처리 및 흐름제어와 같은 패킷의 복잡한 처리는 종단의 단말장치에서만 처리하던 것과는 달리 중간 노드에서 여러 가지 처리를 가능하게 함으로서 기존의 망에서 제공하지 못했던 유연성과 다양한 장점을 제공할 수 있다[1, 2].

망 관리는 새로운 도전에 직면해 있는 새로운 기능들의 효율적인 사용을 통해 효과적으로 관리 기능을 수행할 수 있을 것이다. 따라서 본 논문에서는 기본적으로 오늘날 가장 널리 사용되고 있는 SNMP(Simple Network Management Protocol) 기반의 망 관리 구조에 새로운 액티브 네트워크 기술을 통합함으로서 망 관리 작업을 효율적으로 수행할 수 있도록 하는 데에 초점을 둔다. 이를 통해 얻어질 수 있는 장점은 다음과 같다. 첫째, 관리 결정 위치를 관리중인 노드에 가깝게 이동시킴으로서 관리 시스템의 현재의 관심 사항에 알맞게 정보를 구성하기 때문에 반환되는 트래픽의 양이 감소될 수 있다. 둘째, 관리 시스템에서 적용되는 많은 관리 규칙들을 프로그램 내에 삽입하여 관리 노드들에 전송함으로서 관리 시스템으로부터의 간섭 없이 자동적으로 문제를 식별하고 교정할 수 있다. 셋째, 관리 시스템이 수많은 관리 노드들에 일련의 폴링을 수행하지 않고, 단일 패킷의 관리 노드 순회를 통해 측정 및 제어 동작을 단순하게 함으로서 모니터링 및 제어 루프를 단축할 수 있다.

액티브 네트워크에서의 주요 성분은 패킷 내에 새로운 응용이나 프로토콜에 대한 코드와 데이타를 캡슐화시켜 전송하는 액티브 패킷(Active Packet)과 이러한 액티브 패킷을 수신하여 해석하고 처리하는 액티브 노드(Active Node)로 이루어진다. 본 논문에서 제안하는 망 관리를 위한 액티브 관리 패킷(Active Management Packet : AMP) 구조는 액티브 네트워크 위원회에서 제안한 액티브 네트워크 캡슐화 프로토콜(Active Network Encapsulation Protocol : ANEP)을 기반으로 한다[3, 5].

본 논문에서는 관리자의 서비스 요구에 대응하여 액티브 관리 패킷을 손쉽게 생성할 수 있도록 하는 액티브 관리 패킷 생성기(AMP Generator : AMPG)와 이러한 액티브 네트워크 기술을 바탕으로 망 환경 변화에 신속하게 대응할 수 있도록 하는 액티브 관리 시스템(AMan)에 대해서 기술한다. 또한 액티브 관리자에 의해 구성된 액티브 관리 패킷이 자신의 기능을 효율적으로 수행할 수 있도록 하기 위해 이러한 액티브 관리 패킷의 수신, 스케쥴링, 실행, 모니터링 및 전송 등의 다양한 기능을 처리하는 액티브 노드 관리 엔진을 설계하고, 구현한다.

본 논문에서의 액티브 관리 에이전트의 프로토타입 시스템은 오늘날의 IP 망에서 쉽게 적용될 수 있는 표준(Java, SNMP, ANEP over UDP)들을 적용한다[5, 24, 25, 26, 27]. 액티브 관리 패킷은 사용자의 요구에 따라 망 관리 및 모니터링 프로그램과 다양한 파라메타들이 자바 클래스 형태로 ANEP 내에 캡슐화되며, 또한 액티브 관리 에이전트의 각 구성 요소들은 모두 자바 언어를 이용한 클래스들의 집합으로 구성된다. 자바를 이용하여 액티브 관리 시스템의 기반 구조를 구현하는 이유는 자바 가상 머쉰 환경이 시스템에 의존하지 않는 프로그램들을 지원하며, 또한 안전한 코드 분산을 위해 필요한 호환성 및 보안과 같은 기본적인 기능들을 제공하기 때문이다. 더욱이 자바는 멀티 스레딩, 쓰레기 수집 및 네트워킹 등과 같은 많은 특징들을 가지고 있으며, 자바의 객체 직렬화 메카니즘은 액티브 관리 패킷의 상태 유지와 전송을 위해 유용하기 때문이다.

2. 관련 연구

급속한 망의 성장에 따라 기존의 중앙집중적인 관리 시스템은 관리자 시스템의 복잡도 및 관리 트래픽의 증가 그리고 관리 기능의 유연성 부족이라는 문제점에 직면하고 있다. 따라서 망을 유연하고 효과적으로 관리하기 위한 분산 관리 시스템의 요구가 증가하고 있다. 이 장에서는 중앙집중적인 관리 시스템의 문제점들을 해결하고, 관리 시스템의 유연성 및 확장성 그리고 상호동작성을 제공하기 위해 연구되고 있는 여러 가지 분산 관리 시스템들에 대해 기술하고, 각기 접근 방법에 따라 분류하고자 한다.

2.1 이동 코드 기반

이동 코드 기술을 이용한 망 관리 기법은 여러 가지 매우 다양한 기술들이 연구되고 있지만 모두가 지향하는 공통적인 개념은 망 관리에서의 유연성을 제공하고자 하는데 있다. 즉 동적으로 관리 프로그램들을 에이전트에 전송하고, 그 프로그램이 해당 에이전트에서 수행될 수 있도록 하는 것이다. 이동 코드 기법은 기술, 설계 기법 및 응용간의 경계를 분명히 정의해야 할 필요가 있으며, 이동 코드 시스템(Mobile Code System :

MCS)의 능력에 따라 강 이동(strong mobility)과 약 이동(weak mobility)으로 분류 할 수 있다[36, 37]. 강 이동에서 수행 단위(프로세스나 쓰레드)는 자신의 코드와 수행 상태를 다른 호스트로 이주시킬 수 있으며, 해당 호스트에서 수행이 재개된다. 한편 약 이동에서는 한 호스트에서의 수행 단위가 또 다른 호스트로부터 동적으로 유입되는 코드와 연결되며 해당 코드 역시 이주 가능하지만, 수행 상태가 자동적으로 MCS에 의해 보존되지는 않는다. 기존의 MCS들을 분석하면, 다음과 같은 세 가지 형태로 분류된다.

2.1.1 원격 평가 (Remote Evaluation : REV)

클라이언트가 서버상에 있는 서비스를 호출할 때, 서버스의 이름과 입력 파라메타를 전송하는 것이 아니라, 코드를 전송한다. 따라서 클라이언트는 서비스를 수행하는데 필요한 코드를 소유하고 있으며, 반면에 서버는 자원을 소유하고, 클라이언트에 의해 전송된 코드를 수행하기 위한 환경을 제공한다.

2.1.2 요구시 코드 (Code on Demand : COD)

클라이언트는 자신의 작업을 수행하기 위해 먼저 코드 서버에 접속하고, 그 서버로부터 필요한 코드를 다운로드하고, 수행 중에 해당 코드에 연결한다. 따라서 클라이언트는 자원을 소유하고 있으며, 서버는 코드를 제공한다.

2.1.3 이동 에이전트 (Mobile Agent : MA)

이동 에이전트는 다른 호스트로 자동 이주할 수 있는 수행 단위로서, 각 노드들을 순회하면서 자신의 작업을 수행한다. 개념적으로 MA는 자신의 전체 가상 머쉰을 호스트간에 이주시킬 수 있다. 따라서 클라이언트는 코드를 소유하고 있으며, 자원을 제공하지는 않는다.

망 관리를 위해 이동 코드 기법을 적용한 연구는 Goldszmidt와 Yemini에 의해 관리자-에이전트 위임(MAD) 모델에서 처음 시작되었으며, 네트워크 및 시스템 관리에서 사용되었다. 이러한 관리 기법은 나중에 위임 관리(MbD)로 재명명 되었으며, 기본 개념은 관리 기능을 위해 데이타를 이동하는 것이 아니라 관리 기능 자체를 위임시켜 전송한다는 것이다[19]. 이를 통해 원격 응용에 의해 서비스를 동적으로 확장할 수 있는 능력을 제공함으로서 망 관리 기능의 유연성을 제공한다.

2.2 분산 객체

이동 코드 기술과는 무관하게 분산 객체 기술을 토대로 분산 망 관리를 위한 기술들이 또한 연구되어지고 있다. 분산 객체 기술을 이용한 분산 망 관리 프레임웍 중의 대표적인 것으로는 Sun의 자바 기술을 이용한 JMAPI(Java Management Application Programming Interface)와 OMG(Object Management Group)에 의해 표준화되고 있는 CORBA(Common Object Request Broker Architecture)가 있다[32, 34].

2.2.1 JMAPI

대부분의 기업들은 망 관리에 따른 비용을 절약하기 위해 다중의 플랫폼을 각기 지원하는 추가적인 도구보다는 플랫폼에 무관한 관리 애플릿들을 해당 장치에 제공함으로서 모든 플랫폼에서 동일하게 웹 브라우저를 이용하여 망 장비들을 관리할 수 있도록 하고자 한다. 자바로 작성되는 이러한 관리 애플릿들은 망 관리자로 하여금 SNMP 보다는 자바 RMI(Remote Method Invocation) 기술을 가지고 망 장비들을 관리할 수 있도록 한다. Sun은 이러한 애플릿들의 구성 및 활용할 수 있는 지침과 도구들을 만들었고, 이것이 바로 JMAPI이다.

2.2.2 CORBA

OSI는 객체 지향이며, SNMP 엔티티들은 객체들에 쉽게 매핑될 수 있기 때문에, CORBA를 기존의 망 관리 환경에 통합하기 위한 노력들이 많이 연구되고 있다. X/Open과 망 관리 포럼이 주도하는 JIDM(Joint Inter-Domain Management) 그룹은 CMIP, SNMP 그리고 CORBA를 기반으로 하는 관리 시스템들이 상호 동작할 수 있도록 하기 위한 도구를 구성하였다[33]. SNMP/CMIP 상호동작성은 이전에 NM 포럼의 IIMC(ISO-Internet Management Coexistence) 그룹에 의해 다루어졌다. JIDM은 이에 추가로 CMIP/CORBA 및 SNMP/CORBA 간의 명세 및 상호동작 변환에 관한 내용을 다루고 있으며, GDMO/ASN.1과 CORBA IDL 그리고 SNMP MIB와 CORBA IDL 간의 매핑을 위한 알고리즘을 정의하였다[38, 39]. JIDM 매핑은 CORBA 프로그래머가 GDMO, ASN.1 그리고 CMIP에 대한 지식 없이도 OSI나 SNMP 관리자와 에이전트들을 작성할 수 있도록 하며, 또한 역으로 CMIP, SNMP 프로그래머가 IDL에 대해 알지 않고도 IDL 기반의 자원, 서비스 및 응용들을 엑세스 할 수 있도록 한다.

2.3 통합 솔루션

분산 관리 기능을 기존 관리 프레임웍에 통합하기 위한 방법이 ISO/ITU 및 IETF 등의 표준화 단체에 의해 개발되어졌다. ISO/ITU는 이를 위해 CMIP 명령어 분류자(command sequencer)를 설계하였고, IETF는 SNMP Script MIB를 표준화하였다[31, 35]. 양쪽은 각기 CMIP과 SNMP를 기반으로 하는 관리 프레임웍에 완전하게 통합된다. 관리 기능을 분산시키기 위한 일반적인 접근 방법은 양쪽이 유사하다.

양쪽 표준은 관리 기능이 스크립트에 위임되고, 이러

한 스크립트들은 자신이 수행되어야 할 위치로 전송된다. 이들 스크립트들은 원격에서 시작될 수 있고, 변수들이 이들에 전달될 수 있으며 또한 결과는 발생자에게 반환될 수 있다. 여기에서 이러한 행위들이 어떻게 실현되는 가에 따라 두 표준이 구별된다.

2.3.1 CMIP Command Sequencer

CMIP 명령어 분류자는 스크립트 수준에서 상호동작성을 용이하게 하기 위해 표준 부록에 시스템 관리 스크립트 언어(SMSL)라는 특별한 스크립트 언어를 정의하였다. 명령어 분류자의 SMSL 언어는 관리 시스템의 모든 클래스들을 충분히 지원하며, 로컬 및 원격의 관리 객체들은 CMIP 기능을 이용하여 직접 접근 될 수 있다. 이것은 관리자들간의 통신을 용이하게 한다.

2.3.2 IETF Script MIB

IETF Script MIB는 관리 기능들이 망 노드들에 설치된 수행가능 코드(스크립트) 형태로 정의된다. 여기에서는 이러한 관리 기능의 위임을 위해 SNMP MIB를 정의한다. 이 MIB는 임의의 프로그래밍 언어들을 지원하며, 코드 형식에 관한 어떠한 가정도 없다. 또한 어떠한 구현이 Script MIB의 제어 하에서 원시 코드를 수행할 수 있다면, 컴파일된 원시 코드를 위임할 수도 있다. 특히, Script MIB는 분산된 위치에 관리 스크립트를 전송하고, 그 위치에서 관리 스크립트의 초기화, 중지, 재개 및 종료를 가능하게 한다. 또한 관리 스크립트에 대한 변수 전송이나 수행중인 관리 스크립트를 감시 및 조정하고, 결과를 전송할 수 있다.

2.4 액티브 네트워크

액티브 네트워크에서는 사용자에 의해 커스터마이즈된 관리 응용이나 서비스들을 망 노드들에 동적으로 삽입함으로서 새로운 사용자 요구나 서비스에 신속하게 대응할 수 있도록 한다. 삽입된 관리 소프트웨어는 이 때 망 내의 프로그램가능한 액티브 네트워크 노드에 의해 수행된다. 이러한 액티브 네트워크 기술은 기존의 패킷 형식을 유지하면서 동적으로 프로그램가능한 노드로 프로그램들을 다운로드 할 수 있도록 하는 분리 접근(discrete approach) 방법과 전송 패킷 자체가 하나의 프로그램으로 구성되어 노드에 삽입되는 통합 접근(integrated approach) 방법의 두 가지로 분류할 수 있다[1, 2]. 액티브 네트워크 기술을 이용한 관리 프레임웍은 이동 에이전트 기술을 이용하거나 전용의 하드웨어 및 소프트웨어 기술을 이용하여 구성될 수 있다. 본 논문에서는 망 관리 수행을 위한 액티브 패킷인 AMP를 설계하였고, 이 패킷 내에 관리 프로그램을 삽입하여 전송하는 통합 접근 방법을 이용하였다. 또한 삽입된 패킷의 처리를 위한 수행환경을 설계 및 구현하였고, 해당 패킷의 이주 및 수행을 위해 이동 에이전트 기술과 자바 기술을 이용하였다.

3. 액티브 네트웍 구조

액티브 네트워크에 관한 연구는 현재 여러 곳에서 개별적으로 진행되고 있으며, 현재 연구 진행중인 주제로는 액티브 네트웍 구조, 노드 OS, 구현 기술, 명세, 종단 시스템 문제 등과 보안, 이동성, 혼잡 제어를 포함하는 여러 응용 등이 있다[3, 4, 6, 9, 12, 15, 16, 20].

3.1 액티브 네트웍 구성 요소

액티브 네트워크는 망 구조를 하드웨어와 소프트웨어를 함께 묶는 메인프레임 마인드에서, 하드웨어와 소프트웨어의 기술 혁신을 분리시키는 가상화 된 접근으로 변화시키는 기회를 제공한다.

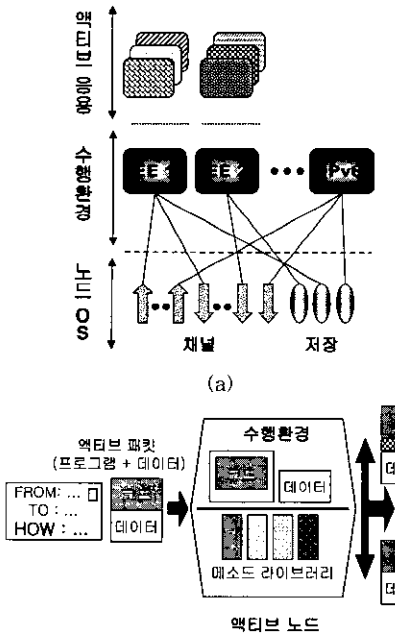


그림 1 액티브 네트워크의 구조

액티브 네트워크는 그림 1의 (a)에서 보는 바와 같이, 기존 망에서의 패킷에 해당하는 액티브 패킷과 이를 수행할 수 있는 중간 노드의 수행 환경(Execution Environment : EE)으로 구성된다. 기존의 전통적인 패킷과 달리 액티브 패킷은 실제 수행될 수 있는 프로그램 코드와 데이타로 구성되어 있다. 액티브 네트워크에서 라우터나 스위치 등의 중간 노드, 즉 액티브 노드는

액티브 패킷에 있는 코드가 추출되고, 실행되는 수행 환경을 제공한다는 점에서 프로그램 가능하다고 말할 수 있다. 액티브 노드의 기능은 노드 OS, 수행 환경, 액티브 응용 등의 다양한 구성 요소들에 의해 분리되어 처리된다. 이러한 구성 요소들의 일반적인 구조는 그림 1의 (b)와 같다. 수행 환경은 종단간 망 서비스들이 접근될 수 있도록 인터페이스를 제공하는 범용 컴퓨팅 시스템의 '쉘' 프로그램과 같이 동작한다. 이러한 구조는 복수의 수행 환경이 하나의 액티브 노드 상에 존재 할 수 있도록 한다. 노드 OS는 수행 환경이 액티브 응용들에 존재하는 추상들을 구성하도록 하는 기본 기능을 제공한다. 이를 위해 노드 OS는 액티브 노드의 자원을 관리하고, 수행 환경으로부터의 전송, 연산, 저장을 포함한 다양한 노드 자원들에 대한 요구를 중재한다[3].

3.2 수행 환경 (EE)

수행 환경은 액티브 네트웍의 사용자들이 활용할 수 있는 가상 머쉰과 프로그래밍 인터페이스를 정의한다. 사용자는 패킷 내에서 적절히 코드화된 명령들을 수행 환경에 전송함으로서 가상 머쉰을 제어한다. 이러한 명령들의 수행은 일반적으로 수행 환경과 액티브 노드의 상태를 변경할 수 있으며, 즉시, 또는 임의의 시간 후에 수행 환경이 패킷을 전송할 수 있도록 한다[3, 4].

액티브 노드를 통한 패킷의 일반적인 흐름은 그림 2와 같다. 패킷이 수신되면, 먼저 패킷 헤더내의 정보를 토대로 패킷을 분류한다. 이러한 분류 작업을 통해 패킷이 전달되어야 하는 입력 채널을 결정하며, 입력 패킷의 분류는 해당 EE가 가르키는 양식에 의해 조정된다. 일반적인 경우, EE는 이더넷 타입이나 IP 프로토콜의 조합, TCP 포트 번호와 같은 속성을 가진 패킷들에 대한 채널 생성을 요청한다. 이러한 요구는 EE 자체에 의해 또는 액티브 응용에 의해 발생된다. 입력 채널 처리 후, 패킷은 채널의 생성을 요청한 EE에 전달된다. 그림에서의 EE1은 UDP 데이타그램에 캡슐화된 ANEP 패킷을 수신한다. 또한 EE2는 ANEP 패킷을 포함하는 UDP 데이타그램, 일반 UDP 데이타그램, IP 데이터그램내의 ANEP 패킷, 그리고 여러 양식(특정 프로토콜 번호나 특정 소오스 및 목적지 쌍)에 맞는 IP 데이타 그램을 수신한다. 출력 부분에서, EE는 출력 스케줄링 뿐 만 아니라 프로토콜 처리를 포함해 출력 채널에 패킷을 제공함으로서 패킷들을 전송한다. 따라서 일반적인 패킷 처리 방법은 링크 입력, 분류, 입력 프로토콜 처리, EE/AA 처리, 출력 프로토콜 처리, 스케줄링 그리고 링크 전송이다.

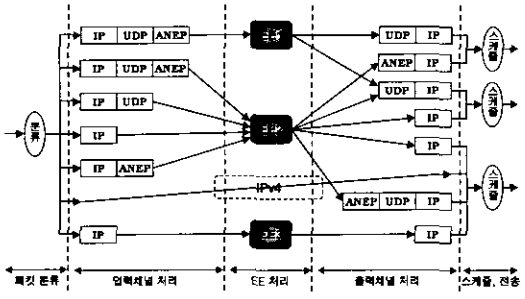


그림 2 액티브 노드에서의 패킷 흐름

수행 환경에 의해 제공된 가상 머쉰의 기능은 아키텍처에 의해 정의되지 않는다. 노드 운영체제는 가상 머쉰을 구현하기 위해 수행 환경들에 의해 사용될 수 있는 기본 기능들을 제공한다. 따라서 대부분의 네트웍 아키텍처 측면들은 수행 환경에 의해 정의된다. 이러한 수행 환경을 정의하는 데는 여러 가지 사항을 고려해야만 한다. 이러한 고려사항 중 중요한 것으로는 프로그램 언어, 프로그램의 인코딩 방식, 프리미티브 제공문제, 보안 문제 그리고 자원의 할당 및 공유 문제 등이 있다.

액티브 네트워크는 망 기반구조에 손쉽게 프로그램들을 추가할 수 있도록 하는데 목적을 둔다. 따라서 수행 환경의 프로그래밍 언어와 런타임 환경의 선택은 중요한 요소가 되며, 이러한 연구중에 대표적인 사례로는 ANTS, Smart Packets, 스위치웨어(SwitchWare), 그리고 넷스크립트(NetScript) 등이 있으며, 현재 다양한 연구가 진행되고 있다[6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21].

3.3 액티브 네트워크 캡슐화 프로토콜

사용자가 특정 EE에 패킷들을 전달할 수 있도록 하기 위해서는 여러 가지 수단이 필요하다. 액티브 네트워크 캡슐화 프로토콜(ANEP)은 이러한 기능을 수행한다[5]. ANEP 헤더는 특정 수행 환경에 할당되는 "타입 식별자" 필드를 포함한다. 만약 하나의 EE가 어느 노드에 존재할 경우, 해당 EE와 같은 타입 식별자를 가진 유효한 ANEP 헤더를 포함한 패킷은 표시된 EE에 연결된 채널들로 라우팅 될 것이다. 이러한 기본 채널들은 EE가 시작할 때 생성된다. 프로토콜 헤더들이 어떻게 구성되어 있는지가 중요하며, 패킷내 일련의 프로토콜 헤더들의 다양한 위치에 ANEP 헤더가 존재할 수 있다.

어떠한 EE에 의해 처리되는 패킷이 ANEP 헤더를 꼭 포함할 필요는 없다. EE들은 액티브 노드를 인식하지 못하는 기존 종단 시스템들에 의해 발생된 패킷을 적절한 채널을 설정하여 처리할 수도 있다. 이에 대한 예로서 그림 2에서는 IPv4 포워딩 동작을 제공하는

"pseudo-EE"와 EE3 에서와 같이 투명하게 헤더를 변경하여 향상된 TCP 서비스를 제공하는 EE를 표현하고 있다.

또한 인증, 기밀성, 무결성 등과 같은 다양한 옵션들이 ANEP 헤더 내에 명시될 수도 있으며, 타입-길이-값(TLV)의 형태로 표현된다. 이러한 옵션들을 위한 필드는 플래그, 옵션 타입, 옵션 길이, 페이로드로 구성된다.

3.4 노드 운영체제

노드 운영체제는 수행 환경들과 하위의 물리 자원들(전송 대역폭, 프로세서 사이클, 스토리지 등) 사이에서 동작하는 계층이다. 이러한 노드 운영체제는 복수의 수행 환경들을 동시에 지원하고, 모든 액티브 노드에 공통인 기본 수준의 기능을 제공하기 위함이다. 이러한 공통 기능은 각 수행 환경이 패킷을 수신 및 전송하기 위한 하위 채널을 구현하고, 복수 환경들에 의한 전송 및 대역폭 계산과 같은 노드 자원들에 대한 엑세스를 제어하며, 라우팅과 같은 공통의 서비스를 지원하는 것을 포함한다[4].

노드 운영체제 인터페이스는 노드 자원들에 대한 4가지의 기본 추상을 정의한다. 스레드, 메모리, 채널의 세 가지 추상은 각각 계산, 스토리지, 통신을 위한 시스템 자원들에 대한 추상이다. 플로우는 계정, 수락 제어를 위한 기본 추상이며, 시스템 내 자원의 스케쥴링을 담당한다. CPU 사이클, 메모리, 망 대역폭과 같은 시스템 자원들은 특정 플로우에 할당된다. 플로우는 기본적으로 입력 및 출력 채널, 메모리 풀 그리고 스레드 풀을 포함한다. 입력 채널에 도착한 액티브 패킷은 해당 플로우에 할당된 스레드와 메모리를 이용하는 수행 환경에 의해 처리되고 나서 출력 채널로 전송된다.

4. 액티브 관리 시스템(AMan)

4.1 설계 요구 사항

본 논문은 액티브 네트워크 기술을 이용하는 망 관리 기술에 초점을 두기 때문에 다음의 요구 사항들이 본 논문의 설계 지표로 사용된다.

4.1.1 기존 시스템의 지원

현재 운용중인 망 관리 시스템에 투자된 비용과 노력이 상당할 것으로 예상된다. 따라서 기존 관리 솔루션에 대한 지원이 필요할 것이다.

4.1.2 호환성

호환성은 코드 이동성을 위한 전제조건이다. 오늘날의 망은 상당히 이질적이며, 판매자에 특정한 플랫폼을 토대로 하는 다양한 망 요소들로 더욱 복잡해지고 있다. 자바는 이질적인 환경에서 동작하는 많은 플랫폼에서 동일하게 수행되는 응용들을 구현하는데 적합하다. 이러한 이유로 인해 본 논문에서 구성되는 기반 구조는 자바 기술을 이용한다.

4.1.3 지속적인 상태 유지

대부분의 망 관리 응용에서 응용이 오랜 시간동안 어느 한 노드에 거주하는 것은 자연스러운 요구이며, 또한 이동 프로그램은 이동 중에 자신의 상태를 지속적으로 유지하길 원한다. 이동 프로그램은 한 노드에서 수행된 다음, 일정 기간동안 중지되고 또 다른 노드로 계속 이동한다. 따라서 이동 프로그램은 수행이 중지된 동안에 자신의 상태를 저장해야만 하며, 수행을 계속 진행하고자 할 때 이전 상태를 복원해야만 한다. 자바는 클래스들의 상태가 파일 시스템 내에 저장하거나 또는 출력 시스템을 통해 전송될 수 있도록 하는 객체 직렬화 메카니즘을 제공한다. 따라서 클래스는 자신이 직렬화되기 이전의 상태를 가지고 재구성될 수 있다.

4.1.4 보안

보안은 분산 컴퓨팅 분야에서 항상 중요한 문제이며, 특히 액티브 노드는 프로그램가능한 실체이며, 새로운 서비스나 프로토콜들이 전송되어 실행되기 때문에 노드의 안전과 보안에 관한 중요한 문제를 야기한다. 신뢰성 없는 프로그램들은 그들이 노드에서 수행되기 전에 인증되고 유효성을 검증 받아야만 한다.

4.1.5 노드 자원들에 대한 인터페이스

관리되는 노드의 자원들에 접근하기 위한 액티브 관리 패킷의 기능은 망 관리를 위해 필수적이다. 이질적인 시스템상의 데이타를 접근하는 것은 비표준 네이밍과 노드 데이타를 관리하기 위한 절차로 인해 복잡해 질 수 있다. 따라서 하위 시스템의 사전 지식없이 자신의 작업을 수행하기 위해 액티브 관리 패킷이 접근할 수 있는 표준화된 노드 자원들에 대한 인터페이스가 필요하다.

4.1.6 이동 코드간 통신

액티브 관리 패킷은 할당된 작업을 수행하기 위하여 다른 노드와 상호동작할 필요가 있다. 이동중인 액티브 관리 패킷간의 통신을 다루기 위해서는 여러 가지 문제가 존재한다. 한가지는 각 패킷의 현재 위치와 관련되며, 다른 하나는 전송중인 데이타의 순서와 관련된다. 액티브 관리 패킷은 이동하기 때문에 자신의 수행과 관련해 통신해야만 하는 다른 관리 패킷의 현재 위치가 어디인지를 알지 못한다. 따라서 수신자에게 동기적으로 또는 비동기적으로 해당 메시지가 전송되도록 해야 한다.

4.2 액티브 관리 시스템 구조

본 논문에서 기술하는 액티브 네트워크 기반의 새로운 망 관리 구조인 AMan(Active Management)은 그림 3에서 보는 바와 같이 액티브 관리자, 액티브 에이전트 및 SNMP 에이전트의 3 부분으로 이루어지며, 다음과 같은 주요 성분들로 구성된다.

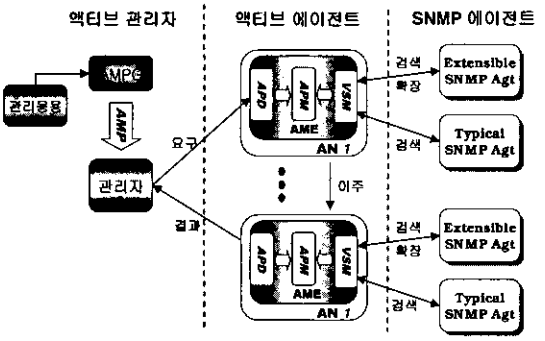


그림 3 AMan 망 관리 모델

4.2.1 액티브 관리자

액티브 관리자는 액티브 패킷 내에 망 관리를 위해 여러 가지 파라메타를 포함하는 AMP를 생성 및 전송하고 해당 결과를 출력하는 책임을 진다. 또한 액티브 관리자에 포함되는 AMPG는 다양한 파라메타를 가진 관리 응용인 서비스 지향의 AMP 객체를 손쉽게 생성할 수 있도록 하며, 이를 액티브 에이전트에 전송하는 역할을 수행한다. AMP는 미리 정의된 정책을 기반으로 정보를 수집하기 위해 관리 노드들 사이에서 이동한다.

4.2.2 액티브 에이전트

액티브 에이전트는 액티브 관리자에 의해 구성된 AMP가 자신의 기능을 효율적으로 수행할 수 있도록 하기 위해 액티브 관리 패킷의 수신, 스케쥴링, 실행, 모니터링 및 전송 등의 다양한 기능을 처리하는 액티브 관리 엔진(AME)을 포함한다. AME는 액티브 관리자로부터 전송된 AMP의 원활한 수행을 위해 로컬의 다양한 물리 자원 및 응용들에 대한 인터페이스를 제공하며, 또한 AMP의 인증 및 보안, 상태 관리, 이주 관리를 위한 기능을 제공한다. 액티브 에이전트의 액티브 노드 관리 엔진 내에 존재하는 액티브 패킷 데몬(APD)은 액티브 관리자로부터 전송된 AMP를 수신하고, 적절한 수행 환경에서 처리될 수 있도록 수신 AMP를 해석하고, 분류하는 역할을 수행하며, APM은 AMP가 자신의 코드를 원활히 수행할 수 있도록 관리하는 책임을 진다.

4.2.3 SNMP 에이전트

SNMP 에이전트는 기존 라우터나 스위치 내에 내장된 고유의 망 관리 에이전트이다. 따라서 액티브 노드는 이러한 기존 응용들에 대한 인터페이스를 제공하기 위한 기능이 필요하며, SNMP 에이전트에 대한 인터페이스는 액티브 에이전트내의 SNMP 관리 모듈인 가상 SNMP 관리자(VSM)가 수행한다. VSM은 망 관리와 관련된 AMP의 동작을 수행하고, SNMP 에이전트로부터 해당 결과를 수신하여 이를 액티브 관리자에 전송하는 역할을 한다.

4.3 액티브 관리 패킷(AMP) 형식

액티브 네트워크에서 액티브 패킷(또는 스마트패킷)이라 부르는 데이타 패킷은 정보의 실체이다. 이러한 패킷들은 목적지 주소, 사용자 데이타와 액티브 노드에서 지역적으로 수행되는 메소드를 포함한다. 액티브 패킷에 대한 일반적인 구조는 액티브 네트워킹 위원회에서 제안한 액티브 네트워크 캡슐화 프로토콜(ANEP)을 따르며, 이를 포함하여 본 논문에서 제안한 액티브 관리 패킷(AMP)의 형식은 그림 4와 같다.

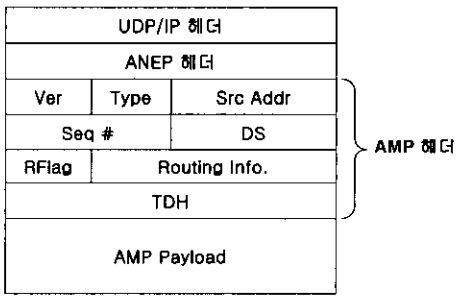


그림 4 액티브 관리 패킷 형식

AMP에 의해 전달되는 코드와 데이타는 바이트 코드 형식으로 존재하며, 실제로 사용자 데이타 상에서 동작하는 메소드들을 기술하는 자바 클래스이다. 사용자 데이타를 포함하는 이러한 자바 클래스의 인스턴스는 클래스 정의를 포함한 바이트 코드들과 함께 직렬화된 객체로서 AMP 내에서 전송된다.

AMP는 공통의 AMP 헤더 내에 싸여진 프로그램, 결과 데이타 또는 메시지들을 포함할 수 있으며, ANEP 내에 캡슐화된다. 버전 번호(Ver)는 현재 AMP의 버전이나 패킷 형식 변경을 식별하기 위해 사용된다. 타입(Type) 필드는 프로그램 패킷, 데이타 패킷, 에러 패킷 또는 메시지 패킷의 4가지 형태 중 하나를 가르킨다. 프로그램 패킷은 해당 호스트에서 수행하고자 하는 코드를 운반하며, 데이타 패킷은 수행 결과를 소오스 노드에 되돌려 주는 데이타를 포함하며, 메시지 패킷은 수행 가

능 코드 이외의 정보 메시지를 전달한다. 마지막으로 에러 패킷은 프로그램 패킷의 전송이나 프로그램 수행시 예외 상황이 발생할 경우의 에러 조건을 반환한다. 소오스 주소(Src Addr)는 AMP를 전송한 노드의 주소를 포함한다. 이 값은 각 클라이언트에 대한 ANEP 데몬에 의해 발생된다. 프로그램 패킷이 네트웍을 순회하고, 하나 이상의 응답을 발생시키기 때문에 이 값은 해당 응답이 전송되어야만 하는 클라이언트를 식별하기 위해 사용된다. 순서 번호(Seq #) 필드는 같은 소오스 주소로부터의 메시지들을 구별하기 위한 값을 포함한다. 이 값은 클라이언트로 하여금 응답 패킷과 삽입된 프로그램을 일치시키기 위함이다. 소오스 주소 값과 같이 응답 패킷은 프로그램 패킷의 순서 번호를 되돌려 준다. 디지탈 서명(DS) 필드는 해당 AMP를 전송하는 사용자의 인증을 위한 필드이다. 다음으로 라우팅 정책 플래그 및 라우팅 정보는 해당 AMP의 노드 순회 라우팅을 관리하기 위한 필드로서 hop-by-hop이나 엔드투 엔드 방식으로 노드들을 이주하거나 또는 사용자가 지정한 임의의 순서대로 라우팅 할 수 있도록 한다. TDH 필드는 앞서 기술된 타입 필드에 따라 AMP 페이로드를 구별하기 위한 타입 헤더 필드이다. 마지막으로 Payload는 실제 전송되는 코드와 데이타를 포함한다.

5. 액티브 에이전트 구조 설계

제안되는 액티브 에이전트 구조는 액티브 노드의 운영체제와 자바 수행 환경인 자바 가상 머쉰 (JVM), AMP의 수신, 스케쥴링, 실행, 모니터링 및 전송 등의 기능을 수행하는 AME 그리고 실제 망 관리 정보를 수집하는 SNMP 에이전트로 이루어져 있으며 다음의 그림 5와 같다.

노드 운영 체제는 현재의 시스템에서 보편적으로 사용되는 운영 체제(Windows, Linux, Solaris)를 이용하며, AME의 구현을 위해 분산된 이질적인 환경에서 플랫폼에 독립적인 자바를 사용하였다. 자바를 이용하는 또 다른 이유는 액티브 네트워크 기반의 응용을 개발하는데 있어서 강력한 네트워킹 기능과 보안 및 자바 가상 기계(JVM)와 같은 가상 환경을 제공하기 때문이다.

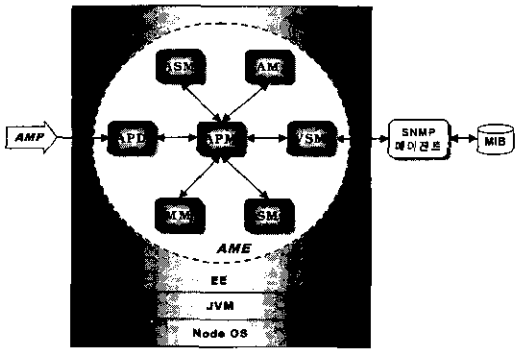


그림 5 액티브 에이전트 구조

5.1 액티브 에이전트 구성 요소

5.1.1 액티브 관리 패킷 데몬 (APD)

APD(AMP Dameon)는 액티브 관리자로부터 전송된 AMP를 해당 TCP, UDP 포트를 통해 수신하고, 이를 해석하는 역할을 수행한다. 먼저 AMP가 액티브 에이전트에 도착하면, 역다중화기에 대기행렬화되며, APD는 해당 AMP가 ANEP에 따르는 가를 검사하며 헤더의 타입 ID 필드를 토대로 AMP의 수행을 관리하는 APM으로 역다중화하는 역할을 수행한다.

5.1.2 액티브 관리 패킷 관리자 (APM)

APM(AMP Manager)은 액티브 에이전트로 수신된 AMP가 자신이 작업을 원할히 수행할 수 있도록 AMP를 관리하는 역할을 수행한다. APM은 AMP가 해당 노드에 도착하면, AMP의 수행을 지원하는 ASM, SM, AM, VSM, MM을 포함한 모든 관리 객체들을 초기화한다. AMP에 있는 코드는 하나의 스레드 내에서 수행되며, 자원들에 대한 모든 요구는 해당 스레드에 의해 이루어진다. AMP당 하나의 스레드를 가지는 것은 해당 스레드에 할당된 자원들을 관리하기 쉽기 때문에 유용하다. 수신된 AMP가 자신의 관리 작업을 완료하면, AM은 MM을 통해 다음에 방문할 노드를 결정한다. 그리고 나서 수행 결과를 다음 목적지로 이동하는 AMP에 추가하여 전송하고, 해당 AMP와 관련된 모든 자원을 해제한다.

5.1.3 인증 및 보안 관리자 (ASM)

액티브 노드는 프로그램 가능한 실체이며, 새로운 서비스나 프로토콜들이 전송되어 실행되기 때문에, 노드의 안전과 보안에 관한 문제를 야기한다. 따라서 ASM (Authentication & Security Manager)는 AMP 내의 응용이 액티브 노드에 있는 내부 데이타 구조나 그 노드에서 수행하는 다른 AMP의 내부 데이타 구조의 무결성을 손상시키는 것으로부터 보호하기 위한 메카니즘을 제공하며, 자바의 보안 클래스 패키지를 이용하여 구현된다.

자바의 보안 API는 더욱 강력한 규칙을 구현하기 위해 쉽게 확장될 수 있다. 자바 보안 API는 자바 응용에서 저수준 또는 고수준의 보안 기능을 구현하기 위해 사용될 수 있는 풍부한 메소드들을 제공한다. 본 논문에

서는 AMP의 인증을 위해 자바에서 제공하는 공중 키와 사설 키의 쌍으로 이루어진 DSA를 이용하였으며, AMP의 무결성 보장을 위해 MD5 알고리즘을 이용하였다.

5.1.4 AMP 중재자 (AM)

AMP는 보통 자신의 활동을 조정할 필요가 있으며, 이러한 조정은 자신들간에 메시지를 전달함으로서 수행될 수 있다. AMP간 통신은 그들의 위치에 투명한 방법으로 이루어지며, 이러한 작업은 AM(AMP Mediator)을 통해 관리되어진다. AM은 이러한 AMP간 통신을 위한 메시지 저장소(Message Pool : MP)를 이용하며, Serializable 인터페이스를 이용하여 구현된다. MP는 같은 액티브 노드에서 수행하는 스레드들간의 동기화 객체로서 두 AMP간의 동기적인 메시지 교환을 위해 내용에 무간한 메시지 저장소를 제공하며, 메시지의 내용은 TLV(Tag-Length-Value) 형식으로 표현된다.

5.1.5 저장 관리자 (SM)

SM(Storage Manager)은 AMP의 수행을 위해 APM과 상호동작하며, AMPContainer 패키지 내에 저장되는 AMP를 관리하는 책임을 진다. 즉, AMP Container 클래스는 AMP내에 포함된 바이트코드(클래스)와 관련 파라메타를 저장한다. 따라서 해당 클래스들은 로컬 디스크에 저장되지 않으며, JAR 포맷으로 저장되고, 적절한 표시자에 의해 구별된다. 한편 해당 AMP의 이주시 AMPContainer는 직렬화되고, 다음 노드의 APD로 전달된다. 결과적으로 파라메타는 하나의 객체로서 존재할 수 있으며, Serializable 인터페이스를 이용하여 구현된다.

5.1.6 이주 관리자 (MM)

MM(Migration Manager)은 AMP의 다음 목적지를 결정하기 위한 기능을 수행한다. AMP의 이주 전략은 액티브 관리자 시스템에서의 AMPG로터 설정되며, 액티브 노드에 수신된 AMP는 이렇게 설정된 이주 전략에 따라 migrate() 메소드를 통해 다음 액티브 노드로 전송되거나, 스케줄된 액티브 노드 순회가 끝난 경우 동작 결과를 액티브 관리자에게 반환할 수 있도록 한다.

5.1.7 가상 SNMP 관리자 (VSM)

VSM(Virtual SNMP Manager)은 액티브 노드로 삽입된 AMP가 SNMP 에이전트와 상호작용할 수 있도록 하는 인터페이스를 제공한다. VSM은 AMP가 고유의 액티브 네트워크 관련 기능을 수행한 후, AMP내의 SNMP 관련 요청 서비스 타입과 OID 그리고 접근 제어 등의 정보를 추출하여 SNMP 에이전트에 전달하고, AMP가 이주를 계속할 경우 해당 요구의 결과를 다음 노드로 진행하는 AMP 내에 추가하거나 또는 해당 AMP가 지속 상태를 가질 경우 이를 SM에 전달하는 역할을 수행한다.

5.2 액티브 에이전트 프로토타입 구현

망 관리를 위한 액티브 에이전트 프로토타입 시스템의 주요 클래스 및 메소드는 그림 6과 같이 구성된다. 현재는 액티브 네트워크 환경을 기반으로 하여 망 관리 정보 수집을 위한 부분 위주로 구현되어 있으며, 각 관리자의 성능이나 유연성을 향상시키기 위한 계속적인 연구가 진행중이다.

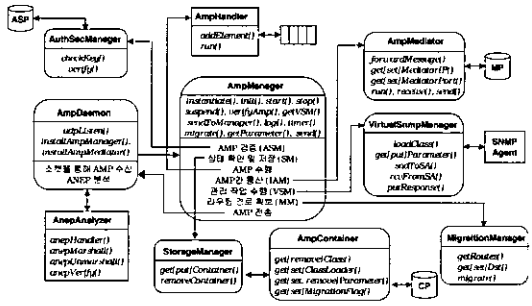


그림 6 액티브 에이전트의 주요 클래스 및 메소드

그림에서와 같이 AmpDaemon(APD) 클래스는 해당 TCP, UDP 포트로 입력되는 AMP를 수신하는 데몬 프로세스이다. udpListen() 메소드를 통해 UDP 포트 3322에서 AMP를 수신하여, ANEP 헤더를 분석(class AnepAnalyzer)하고, ANEP 타입 식별자를 토대로 AMP 패킷을 역다중화하여, APM에 전달한다. APD로 수신된 AMP의 수행을 관리하고, 제어하는 역할을 APM이 담당한다. AM을 구현하는 AmpManager 클래스는 액티브 에이전트의 핵심 부분이며, AMP와 AME의 여러 다른 서비스와의 인터페이스를 제공한다.

APM은 그림 7에서와 같이, APD로부터 전달된 AMP를 큐에 저장(class AmpHandler)하고, 인스턴스

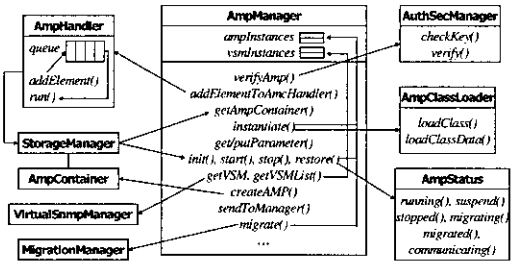


그림 7 APM 기능 모듈

화 시킨다. 그리고 나서 AMP의 AuthSecManager 클래스를 통해 사용자와 코드에 대한 인증을 수행하고, AMP의 현재 상태를 저장(class Storage Manager, AmpStatus)한다. APM은 인스턴스화된 AMP의 망 관리 작업 수행을 위해 해당 객체와 파라메타를 VSM (class VirtualSnmpManager)에 전달한다. VSM은 해당 객체와 파라메타(요구 타입, 커뮤니티, OID)를 분석하여 SNMP 에이전트에 대해 sndToSA() 메소드를 통해 관리동작을 수행하고, rcvToSA() 메소드를 통해 결과를 받아 이를 다시 APM에 전달한다. APM은 VSM으로부터 결과를 받아 AMP를 구성하여 이를 액티브 관리자에 전송하거나 만약 해당 AMP가 다른 노드로의 라우팅이 예정되어 있을 경우, MM(class Migration Manager)를 통해 다음 목적지를 결정하여, AMP를 전송한다.

5.3 액티브 관리 패킷 동작 흐름도

액티브 관리 패킷(AMP)이 액티브 관리자로부터 생성되어, 액티브 에이전트에 전달, 수행되는 동안의 흐름도는 다음 그림 8과 같다.

AMPG 로부터 생성된 AMP는 부호화되고 압축된다. 그리고 이동 코드의 인스턴스를 직렬화함으로서 확보된 지속 상태와 순서화된 순회 노드 등의 파라메타와 함께 망 관리 응용에 의해 액티브 에이전트에 전송된다. 액티브 에이전트는 APD를 통해 AMP를 수신하고, SM을 통해 해당 패킷의 코드 및 데이타와 상태를 저장소에 저장한다. 전송이 완료되면, 액티브 에이전트는 인증 및 데이타 무결성을 위해 ASM을 통해 AMP를 검증한다. 만약 AMP가 검사를 통과하면, 해당 패킷은 인스턴스화

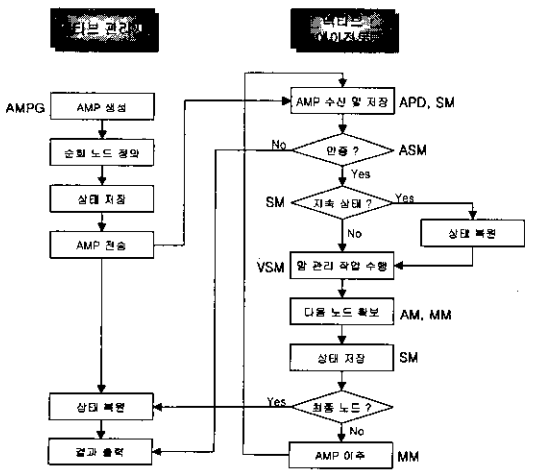


그림 8 액티브 관리 패킷 동작 흐름도

되며, 문제 발생시 AMP는 폐기되고, 액티브 관리자에 해당 결과를 전송한다. AMP는 지속 상태를 가질 수도, 갖지 않을 수도 있다. 만약 지속 상태를 가지고 있다면, 역직렬화를 통해 상태를 복원한다.

그외의 AMP는 인스턴스화 되고, init(), start() 메소드가 호출된다. 이 지점에서 AMP는 하나의 스레드로서 수행되며, AMP에 포함된 수행 코드와 파라메타를 토대로 관리작업을 수행한다. 액티브 관리자에 의해 또는 AMP 자체에 의해 다른 액티브 에이전트로 AMP의 이주가 요청되었다면, MM은 관련된 AMP에 대해 migrate() 메소드를 호출한다. 이때 AM은 필요한 경우, SM과 상호작용하여 해당 AMP와 관련된 지속 상태를 AMP에 적재한다. 이주가 완료되면, APM은 AMP가 점유하고 있는 메모리 공간이나 확보한 자원 등을 해제하고, AMP의 작업을 종료한다.

6. 액티브 관리 시스템 구현

6.1 액티브 관리자 시스템 구현

액티브 관리자는 액티브 에이전트의 기능을 수행하는 액티브 노드와의 상호작용을 통해 관리 및 제어 동작을 수행한다. 관리대상 노드들인 액티브 에이전트들은 그림 9에서와 같이 관리자에 의해 브로드캐스트 폴링을 통해 탐색된다. 자동 노드 탐색에 의해 발견된 액티브 에이전트는 액티브 관리자의 탐색 노드 리스트에 해당 노드 이름이 추가된다. 발견된 액티브 노드 에이전트들의 상태는 실시간으로 ICMP, SNMP, ANEP의 동작 상태에 따라 액티브 관리자 시스템의 GUI 상에 디스플레이 된다. 또한 액티브 관리자는 AMPG를 통해 생성된 AMP를 해당하는 액티브 노드에 전송하고, 그 결과를 수신한다.

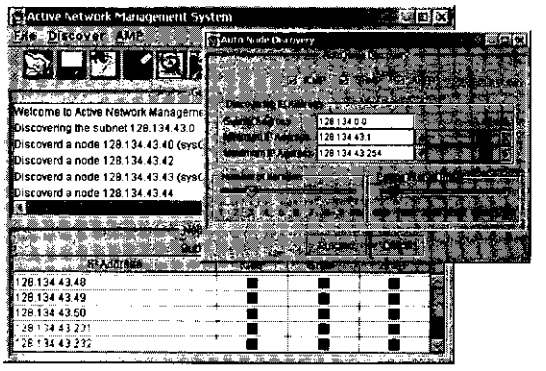


그림 9 액티브 관리자 시스템 사용자 인터페이스

6.2 액티브 관리 패킷 생성기(AMPG) 구현

AMPG는 기본적으로 사용자에 의해 커스터마이즈된

AMP를 생성 및 구성하기 위한 도구이다. AMPG는 먼저 관리 응용에 의한 노드 탐색 과정을 통해 발견된 액티브 에이전트들에 대한 관리 요구를 발생시키기 위해 순회되어야 할 노드들을 선택한다.

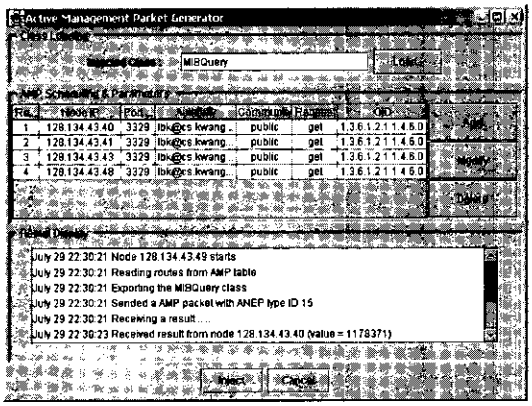


그림 10 AMP 생성기 사용자 인터페이스

그림 10에서와 같이 선택된 노드들이 순회노드 리스트에 추가되면, 관리자는 액티브 에이전트에서 수행해야 할 수행가능 파일의 클래스(MIBRequest)를 할당하고, 해당 노드들에 대한 요청 MIB 값과 서비스를 그림 11의 AMP MIB 선택기를 통해 선택한다.

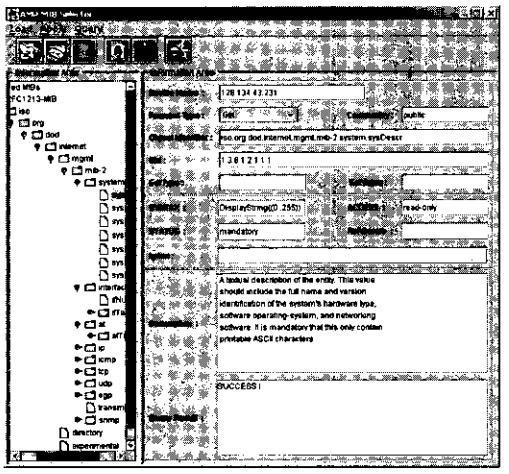
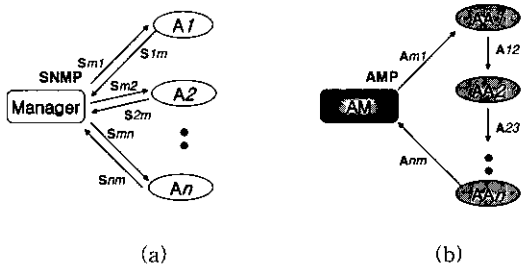


그림 11 AMP MIB 브라우저 사용자 인터페이스

마지막으로 관리자는 순회노드 리스트에 있는 노드들에 대한 라우팅 순서를 결정하여 AMP를 구성한다. AMPG는 액티브 관리 패킷에 명시된 특성을 포함하는 스켈레톤 자바 소오스 코드를 사용한다. 생성된 자바 코드가 컴파일되고, 생성된 자바 바이트코드가 압축되며, 해당 액티브 노드 에이전트들에 UDP 연결을 통해 전송된다.

6.3 시험 환경

일반적인 SNMP 기반 망 관리 방법은 그림 12의 (a)에서와 같이 중앙의 관리자가 관리 대상인 되는 여러 에이전트들을 하나씩 폴링하는 방법으로 이루어진다.



(a) (b)

그림 12 전통적인 SNMP와 AMan 기반의 망 관리 방법

액티브 네트워크에서는 중간 노드(라우터나 스위치)들이 해당 노드로 삽입된 액티브 패킷을 해석할 수 있는 수행환경이 존재해야 한다. 그러나 기존 망 노드들은 이러한 액티브 패킷을 해석할 수 있는 수행환경이 존재하지 않는다. 따라서 본 논문에서는 이러한 실행가능성(feasibility) 문제를 해결하고, 제안한 망 관리 구조의 실험을 위해 그림 11의 (b)에서와 같이 기존의 유닉스나 리눅스 기반의 호스트에 액티브 네트워크 수행환경을 구성하여 이를 액티브 노드로 가정한 중간 단계의 솔루션을 구성하였다.

전통적인 SNMP 기반의 관리자 및 액티브 관리자는 PC 기반의 윈도우즈 운영체제에서 동작하며, 관리 대상 에이전트 및 액티브 에이전트들은 솔라리스 OS 기반의 썬 워크스테이션과 리눅스 기반의 PC로 구성되었다. 전통적인 SNMP 망 관리 방법은 그림에서와 같이 일대일 폴링을 통해 해당 결과를 수신한다. 그러나 액티브 관리자는 액티브 노드 탐색 응용을 통해 동작중인 액티브 노드를 탐색하며, 액티브 관리 패킷 생성기를 통해 순회 노드, 라우팅 순서, 관리 서비스 및 요청 MIB 등의 파라메타를 포함한 AMP를 생성하여 첫번째 액티브 에이전트에 전송한다. 전송된 AMP는 라우팅 순서에 따라 각 노드들을 방문하고, 해당 노드에서는 AMP 수행 결과를 다시 AMP에 추가하여 다음 노드로 전송하며, 이러한 과정이 반복된 후 최종 노드에서는 이렇게 수집된 결과로 구성된 AMP를 다시 액티브 관리자에 전송한다.

6.4 적용 및 분석

본 논문에서의 프로토타입을 구성하는데 있어서 우리는 성능에 목적을 두지 않고, 액티브 기술을 이용한 망 관리 시스템 구조의 설계와 응용들을 시험할 수 있도록 하는 개념 시스템을 구성하는데 목적을 두었다. 액티브 망 관리 시스템 구조의 응용은 폴링과 같은 데이타 집중적인 관리 동작 위에서 시험되었으며, SNMP 기반의 폴링과 비교하여 요청된 관리 정보를 수집 및 반환하기 위한 AMP의 사용이 기존의 중앙집중적인 망 관리 방법과 비교하여 확장성 및 유연성을 향상시키는 것으로 보여주었다.

6.4.1 관리 기능의 유연성

망 관리자는 보통 관리되는 망 노드들의 특정 제어를 수행한다. 이러한 망 요소들은 정적인 장치들의 유형에 따라 또는 동적인 여러 특성에 따라 제어가 수행된다. 예를 들어, 함수 *fA*가 선택된 각 노드들에 따라 계산되어지는 MIB 변수들의 수 *nM*에 의존하고, 만약 이 함수가 특정 값을 초과하거나 증가율이 어떠한 임계값을 초과한다면 관리자는 또 다른 제어 동작을 수행하여야 한다. 이러한 작업은 다음과 같은 알고리즘으로 표현될 수 있다.

```
for each Network Element NEi satisfying a property pA do
        get the values of MIB variables needed for fA
        calculate fA
        if error (fA(i), fA(i-1), t) > b then
        execute fB
        sleep(t)
```

여기에서 *NEi*는 관리되는 망 노드이며, *pA*는 해당 *NE*가 만족해야 하는 일반적인 특성이다. 또한 *fA*는 *nM* MIB 변수에 따른 건강 기능이며, *b*는 임계값 그리고 *fB*는 관리 동작이다. 사실 이러한 기능을 전통적인 중앙집중 관리 시스템에서 수행하고자 한다면, 포함되는 MIB 변수의 값이 연속적인 폴링 동작을 통하여 검색되어야만 하고, 검색 결과에 대응하는 패킷을 다시 전송해야만 하기 때문에 많은 양의 네트웍 트래픽을 유발한다. 이러한 경우 *b* 임계값과 *error* 함수가 표준 MIB내에 존재할 수 없기 때문에 SNMP에서의 트랩 메카니즘이 사용될 수도 없다. 반면에 액티브 네트워크 기술을 이용하면, 해당 패킷내에 각 함수와 MIB 변수 및 관련 파라메타들을 함께 전송함으로서 이러한 기능을 손쉽게 달성할 수 있다.

6.4.2 응답시간

본 논문에서 제안된 구조는 그림 13에서와 같이 6개의 관리 대상 노드로 이루어진 소규모의 망에서 시험되었다.

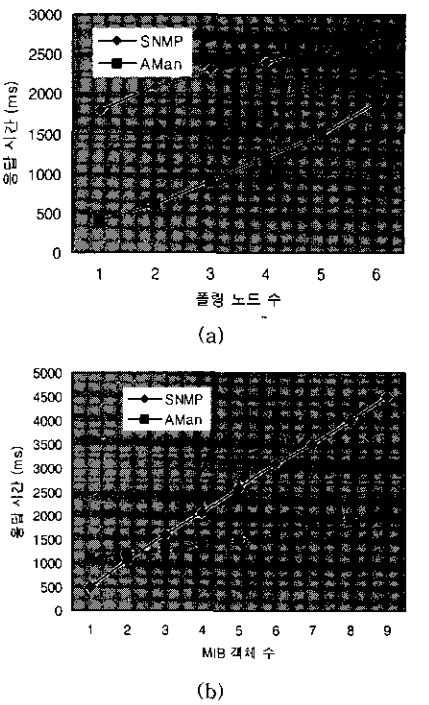


그림 13 MIB 객체 수와 폴링 노드 수에 따른 응답시간 비교

그림 13에서와 같이, 고정된 수의 관리 노드에 대한 MIB 변수(a)와 일정한 수의 MIB 객체들에 대한 관리 노드 수(b)를 토대로 응답시간을 측정한 결과 액티브 네트워크 기반의 관리가 많은 수의 MIB 객체들을 적용할 때 제한된 크기의 망에서 더 나은 성능을 가진 것으로 증명되었으나, 관리되는 노드의 수가 증가함에 따라 변동하지 않는 SNMP 기반의 폴링 응답 시간과 반대로 선형적으로 증가함을 보여주었다. 이것은 SNMP 기반 폴링 기법에 비해 더 낮은 응답 시간을 유지하면서 관리될 수 있는 노드들의 수에 제한이 있음을 보여준다. 이러한 액티브 기반 망 관리 구조에서의 노드 수 증가에 따른 문제는 관리되는 망을 여러 도메인으로 분리하고, 각기 위임된 액티브 에이전트와 분할 NMS 구조를 적용함으로서 향상될 수 있을 것이다.

6.4.3 트래픽 부하

관리되는 각 노드들에 대해 관리 동작 수행에 따른 패킷 크기를 측정함으로서 전체 관리 트래픽을 계산할 수 있다. 관리 트래픽을 측정하기 위해 6.4.1에 기술된 알고리즘에 따라 관리 노드들에 대한 건강 기능을 검사한다고 하자. 앞의 그림 12에서와 같이 전통적인

SNMP 폴링 기법을 이용한다면 폴링 대상의 정적 에이전트들로부터 관련 객체 값들이 반환되고 이 값들은 모두 관리자에 의해 계산된다. 이러한 경우 Sp가 평균 요구/응답 크기이고, k개의 동작 변수들에 대해 n개의 장치들에 폴링이 적용된다고 한다면, i 번의 폴링 간격 동안에 소요되는 전체 트래픽은 다음과 같다.

$$T_{centralized} = 2 * Sp * n * k * i$$

이와 비교하여 Aman 기반의 관리 기법을 이용할 경우, 이러한 알고리즘이 구현된 코드와 관련 동작 변수들이 하나로 압축되어 구성된 AMP가 전송된다. 따라서 건강 기능의 계산은 관리 대상의 로컬의 액티브 에이전트에서 수행되고, 마지막 전달된 노드로부터 결과가 반환된다. 따라서 AMan 기반 관리 기법에서 소요되는 전체 트래픽은 다음과 같다.

$$T_{AMan} = (n + 1) * Ap * i$$

여기에서 압축된 AMP의 크기(Ap)는 Sp와 비교하여 대략 50% 정도 증가하지만, k개의 동작 변수가 Ap 내에 하나로 구성되어지기 때문에 노드 수가 증가함에 따라 관리 데이타로 인한 트래픽 부하는 대략 k 인자만큼 감소하게 된다.

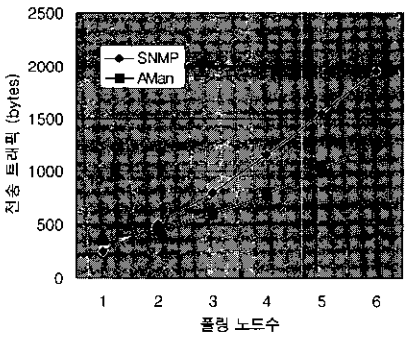


그림 14 중앙집중적인 SNMP와 AMan 기반 관리 트래픽의 비교 측정

그림 14는 중앙집중적인 SNMP 기반의 관리 트래픽과 AMan 기반 관리 트래픽 비교 측정 결과를 보여준다. AMan 기반 관리 기법에서 나타난 문제점은 기존의 SNMP 에이전트에서는 수행되지 않는 AMP내 코드와 데이타의 처리 과정으로 인해 액티브 에이전트에서의 처리 오버헤드가 발생한다는 점이다. 그러나 이러한 처리 오버헤드는 감당할 수준이며, 관리 트래픽이 액티브 에이전트들에 공평하게 분산되고, 관리자 호스트에 데이타가 집중되는 단점이 보완됨으로서 이러한 문제점을 상쇄할 수 있었다.

7. 결 론

기존의 중앙집중적인 망 관리 시스템에서는 물리적으로 분산된 관리 대상 에이전트들로부터 데이타를 수집하고, 분석하는 작업이 중앙의 관리자 시스템에 집중됨으로서 관리자 시스템에서는 병목 현상이 발생하고, 중복된 많은 관리 데이타로 인해 망 트래픽이 증가하게 된다. 관리되는 에이전트 또한 수동적이며, 다양한 판매자에 의해 제공됨으로서 관리 기능을 수정하거나 확장하기가 상당히 복잡하다. 따라서 본 논문에서는 액티브 네트워크 기술을 이용한 새로운 망 관리 기법을 도입함으로서 이러한 문제점들을 해결하고자 시도하고 있다.

액티브 네트워크 기술은 전통적인 패킷 교환 망에서의 수동적인 패킷 처리를 넘어서 사용자에 의한 계산 및 수행이 가능하게 함으로서 현재의 망 환경에서 다룰 수 없는 다양하고도 혁신적인 기술들을 연구할 수 있는 기반을 제공한다. 또한 새로운 프로토콜이나 서비스가 출현할 때마다 새로운 장비에 새로운 프로토콜과 서비스를 내장하지 않고도 사용자가 직접 삽입할 수 있도록 함으로서 하드웨어 플랫폼의 변경이 필요 없이 신속한 서비스를 가능하게 한다.

본 논문의 목적은 액티브 네트워크 기반 위에서 새로운 응용을 시험할 수 있도록 하는 프로토타입을 구성하는데 있으며, 이러한 경험을 바탕으로 액티브 노드에서의 더욱 용이한 망 관리 프로그래밍과 신속한 관리 작업을 위해 액티브 노드 관리 엔진을 설계 및 구현하였다. 또한 액티브 관리 패킷이 지능적인 노드 진단 및 자동 구성 그리고 장애 복구와 같이 다양한 응용을 수행할 수 있도록 하기 위한 AMPG를 구현하였다.

본 논문에서는 액티브 네트워크 기술을 이용하여 망 관리자 및 관리 노드 에이전트의 프로토타입 구현에 대해 기술하였으며, 이러한 새로운 망 관리 기법은 다가올 미래의 네트웍에서 새롭고 효과적인 방법으로 망 관리를 수행할 수 있을 것이다.

참 고 문 헌

[1] D. Tennenhouse and D. Wetherall, "Towards an Active Network Architecture," Computer Communication Review 26(2), April 1996.

[2] D. Tennenhouse et al, "A Survey of Active Network Research," IEEE Communications Magazine, January 1997.

[3] DARPA AN Architecture Working Group, "Architectural Framework for Active Networks," http://www.cc.gatech.edu/projects/canes/papers/arc

h-1-0.pdf, July 1999.
[4] DARPA AN Node OS Working Group, "NodeOS Interface Specification," http://www.cs.princeton.edu/nsg/papers/nodeos99.ps, January, 2000.
[5] D. Alexander, et al, "Active Network Encapsulation Protocol (ANEP)," RFC Draft, http://www.cis.upenn.edu/~switchware/ANEP/docs/ANEP.txt, July 1997,
[6] D. Wetherall, J. Guttag, and D. Tennenhouse, "ANTS: A Toolkit for Buildig and Dynamically Deploying Network Protocols," IEEE OPENARCH '98, April 1998.
[7] D. Wetherall, U. Legedza and J. Gutttag, "Introducing New Internet Services: Why and How," IEEE Network Magazine, July 1998.
[8] D. Wetherall, J. Guttag and D. Tennenhouse, "ANTS: Network Services Without the Red Tape," IEEE Computer, April 1999.
[9] D. Alexander et al, "The SwitchWare Active Network Architecture," IEEE Network Magazine, July 1998.
[10] D. Alexander et al, "A secure active network environment architecture: Realization in Switch Ware," IEEE Network Magazine, July 1998.
[11] M. Hicks et al, "PLAN; A Programming Language for Active Networks," International Conference on Functional Programming (ICFP'98), September 1998.
[12] Livio Ricciulli, "An Adaptable Network Control and Reporting System," Sixth IFIP/IEEE International Symposium on Integrated Network Management, May 1999.
[13] M. Molteni, L. Ricciulli and S. Tsui, "User Guide to Anetd 1.4," http://www.csl.sri.com/ancors/anetd, April 2000.
[14] K. Calvert, S. Bhattacharjee, S. Zegura, J. Sterbenz, "Directions in Active Networks," IEEE Communications Magazine, October, 1998.
[15] S. Bhattacharjee, K. Calvert, E. Zegura, "An Architecture for Active Networking," Proceedings of High Performance Networking (HPN'97), April, 1997.
[16] S. Bhattacharjee, K. Calvert, E. Zegura, "Active Networking and the End-to-End Argument," International Conference on Network Protocols (ICNP'97), October 1997.
[17] Y. Yemini and S. da Silva, "Towards programmable networks," Interational Workshop on Distributed Systems: Operations and Management (DSOM'96), October 1996.
[18] S. Silva, D. Florissi, Y. Yemini, "Composing Active Services in NetScript," DARPA Active Networks Workshop, March, 1998.
[19] Y. Yemini, G. Goldszmidt, and S. Yemini, "Network Management by Delegation," Proceedings of ISINM'91, April 1991.
[20] B. Schwartz, A. Jackson and W. Strayer, "Smart Packets for Active Networks," OPENARCH'99, March 1999.
[21] A.B.Kulkarni et al, "Implementation of a Prototype Active Network," OPENARCH'98, April 1998.
[22] Danny Raz and Yuval Shavitt, "An Active Network Approach to Efficient Network Management," Proceedings of the 1st International Working Conference on Active Networks (IWAN'99), July 1999.
[23] A. Bieszczad, B. Pagurek and T. White, "Mobile Agents for Network Management," IEEE Communications Surveys, Fourth Quarter 1998.
[24] James Gosling, Bill joy, and Guy Steele, The Java Language Specification, Addison Wesley, 1996.
[25] J. Case, M. Fedor, M. Shoffstall, J. Davin, "A Simple Network Management Protocol(SNMP)," RFC1157, May 1990.
[26] M. Rose, K. McCloghrie, "Structure and Identification of Management Information for TCP/IP-based Internets," RFC1155, May 1990.
[27] K. McCloghrie and M. Rose, "Management Information Base for Network Management (MIB-II)," RFC1213, March 1991.
[28] M. Daniele, B. Wijnen and D. Francisco, "AgentX Extensibility Protocol Version 1," RFC2257, January 1998.
[29] Agent Extensibility Issue, The Simple Times, Volume 4, Number 2, April 1996.
[30] "Agent Extensibility (AgentX) Protocol Version 1," RFC2741, January, 2000.
[31] J. Schonwalder, J. Quittek and C. Kappler, "Building Distributed Management Applications with the IETF Script MIB," IEEE Journal on Selected Areas in Communications, Vol. 18, No. 5, May 2000.
[32] Sunsoft, Java Management API Architecture. Revision A. September 1996.
[33] S. Mazumdar, "Inter-Domain Management between CORBA and SNMP," In proc. 7th IFIP/IEEE int. Workshop on Distributed Systems: Operations & Management(DSOM'96), October 1996.
[34] OMG, "The Common Object Request Broker: Architecture and Specification," Revision 2.0, July 1995.
[35] OSI-Systems Management-Command Sequencer, International Standard ISO 10164-21. 1998.
[36] A. Fuggetta, G.P. Picco and G. Vigna, "Understanding Code Mobility," IEEE Trans. on Software Engineering, July 1997.

[37] T. Magedanz and T. Eckardt, "Mobile Software Agents: a new Paradigm for Telecommunications Management," In Proc. 1996 IEEE Network Operations and Management Symposium(NOMS '96), April 1996.
[38] S. Mazumdar and T. Roberts, "Translation of GDMO Specification into CORBA-IDL," Report of the XoJIDM task force, August 1995.
[39] S. Mazumdar, "Translation of SNMPv2 Specification into CORBA-IDL," Report of the XoJIDM task force, September 1996.
[40] AdventNet, AdventNet SNMP API 3.0, URL:http://www.adventnet.com/

이 병 기

1991년 원광대학교 물리학과(학사). 1994년 광운대학교 컴퓨터과학과(석사). 1994년 ~ 1996년 (주)코오롱정보통신 기술연구소. 1996년 ~ 현재 광운대학교 컴퓨터과학과 박사과정. 관심분야는 컴퓨터 네트워크, 액티브네트워크, 망관리, 분산 시스템 등

조 국 현

한양대 전자과 졸업. 일본 동북대 대학원(공학박사). 1984년 ~ 현재 광운대학교 컴퓨터과학과 교수(신기술 연구소 연구원). 광운대학교 이과대학 학장, 전자계산소장, 전산대학원장 역임. 한국정보과학회 학회지 편집위원, 이사 역임. 현재 OSIA 회장. 관심분야는 정보통신망 관리, 분산처리, 멀티캐스팅 및 정보통신분야의 표준화