논문 2006-43CI-3-1

임베디드 시스템들을 위한 SNMP 기반의 저부하 시스템 모니터링

(Low Overhead System Monitoring Based on SNMP for Embedded Systems)

김 태 수*, 김 동 억*, 김 용 석**

(Tae Su Kim, Dong Uk Kim, and Yong-seok Kim)

요 약

SNMP는 네트워크 장비들의 관리를 위한 표준 프로토콜이지만 요즘의 대부분의 컴퓨터 시스템들은 SNMP 에이전트로서의 기능을 보유하고 있다. 본 논문에서는 임베디드 시스템을 위한 저부하의 SNMP 라이브러리를 개발하고, 이를 활용하여 많은 수의 시스템들을 일괄적으로 모니터링하기 위한 소프트웨어를 구현하였다. 모니터링 시스템은 모니터링 서버, SNMP 에이전트들, 및 클라이언트 프로그램으로 구성된다. 모니터링 서버는 모니터링 대상 시스템에서 실행되는 SNMP 에이전트들로부터 상태정보들을 수집하고 요약 정보를 클라이언트 프로그램에게 전달한다. 클라이언트 프로그램은 Java를 활용한 독자적인 프로그램과 웹 기반의 프로그램의 두 가지 버전을 구현하였다.

Abstract

SNMP is a standard protocol for management of networking devices. Nowadays, most computer systems have capability to act as SNMP agents. In this paper, we developed a low overhead version of SNMP library for embedded systems, and implemented a monitoring software based on the library for a large number of target systems. The monitoring system consists of a monitoring server, SNMP agents, and client programs. The monitoring server collects status information from SNMP agents running on the monitoring targets, and sends summary information to client programs. We implemented two versions of clients, Java based standalone program and Web based program.

Keywords : SNMP, Management, System Monitoring, Java, 웹기반 프로그램

I. 서 론

네트워크에 접속되어 사용되는 서버의 수가 늘어나면서 이들을 효율적으로 감시하고 관리하는 것이 중요한 업무가 되고 있다. 이동통신 업체들의 경우 수백 대의 서버 컴퓨터들이 사용되고 있고 제조업체에서도 수십 대의 컴퓨터들이 활용되고 있다. 수많은 서버 컴퓨터들을 감시하고 관리하는 방법으로서 서버별로 별도로 처리하는 대신에 통합적으로 처리하는 것이 비용 및 효율 면에서 필수적인 상황이 되고 있다. 이러한 목적으로 개발된 것으로서 Tivoli[1], OpenView[2], EasyMon[3] 등이 있다. 이들은 상업적 목적으로 개발된 것으로서 독자적인 프로토콜을 적용하고 있어서 다양한 시스템들을 통합적으로 관리하는 데는 한계가 있다.

본 논문에서는 일반적으로 널리 보급되어 있는 SNMP를 기반으로 하여 컴퓨터 시스템들을 통합적으로 감시하는 소프트웨어 개발에 적용하기 위한 라이브러리인 ES-SNMP (Embedded System SNMP)를 개발하고 이를 활용한 간단한 시스템 모니터링 소프트웨어를 제시하였다. 시스템 모니터링에 SNMP를 활용하면 감시 대상 컴퓨터에는 별도의 프로그램 개발없이 제조업체에

* 학생회원, ** 정회원, 강원대학교 컴퓨터정보통신공학과
(Dept. of Computer and Communications Engineering, Kangwon National University)
※ 본 논문은 부분적으로 강원대학교 정보통신연구소의 지원을 받았음
접수일자: 2005년10월5일, 수정완료일: 2006년5월9일

서 제공하는 SNMP 에이전트를 활성화 하는 것만으로 충분하므로 간편하게 모니터링 시스템을 구축할 수가 있게 된다.

네트워크가 보급되기 시작한 초기에 네트워크 관리를 위해 관리자들이 주로 사용하였던 것은 ping 서비스에 사용되는 ICMP (Internet Control Message Protocol)이다. 하지만 이는 단순한 기능만을 제공하는 프로토콜로서 해당 시스템이 현재 가동 중인지 아닌지를 판별하고 그 시스템에 네트워크를 통하여 통신을 하는데 걸리는 시간이 어느 정도인지를 알려주는 서비스를 제공해주는 제한적인 프로토콜이다. 이 것으로는 구성이 복잡한 네트워크를 관리하기에는 무리가 있으며, 각 네트워크에 대한 세밀한 정보를 제공할 수 없어서 보다 안정적인 네트워크 관리를 위해서는 해당 프로토콜을 사용할 수가 없었다. 이러한 이유로 더 많은 정보를 수집하고 네트워크를 관리할 수 있는 새로운 네트워크 관리 프로토콜들이 제안되었고, IAB (Internet Architecture Board)에서 여러 프로토콜의 기능을 복합적으로 가지고 있는 SNMP[10-13] (Simple Network Management Protocol)를 국제 표준으로 선택하였다.

SNMP를 활용하여 모니터링에 적용하려는 시도로서는 Remos[4], MAPI[5], PerfMC[6] 등이 있다. Remos는 GRID 환경에서 네트워크와 컴퓨팅 노드들에 대한 성능을 모니터링하는 데 있어서 부분적으로 SNMP를 활용하였고, MAPI는 모바일 에이전트들을 모니터링하기위해 JAVA를 기반으로 하여 모니터링 API를 구현하는데 있어서 부분적으로 SNMP를 사용하였다. PerfMC는 대규모 컴퓨팅 클러스터에 대한 성능을 모니터링하기 위해 개발한 것이다.

Net-SNMP는 SNMP 표준에 맞게 개발된 대표적인 패키지로서 공개 소프트웨어로 제공되고 있다. 이는 원래 UC Davis에서 개발한 UCD-SNMP를 공개소프트웨어로 전환한 것으로서 대중적으로 가장 많이 사용되는 SNMP 패키지이다[15,16]. 하지만 이 패키지에서 제공하는 라이브러리는 범용 시스템에 맞추어져 설계되었으며, SNMP의 각 버전과 다양한 상황 등에 적용할 수 있도록 구성되어 있어서, 결과적으로 매우 방대하고 복잡하게 구성 되어 있다. 응용 프로그래머가 네트워크나 시스템 관리를 위해 단지 몇몇의 SNMP 메시지만을 사용할 때에는 이 패키지는 많은 오버헤드를 유발한다. 또한 연결 설정 및 메시지 구성에 Net-SNMP의 여러 가지 함수들을 조합하여 사용해야 하고, 내부적으로 많은 데이터를 유지하므로 실행에 있어서 많은 지연이 생긴다. Net-SNMP 라이브러리를 사용하기 위해서는 복잡한 라이브러리의 구조를 이해해야 하며, 함수들을 조합하여 내부적인 데이터를 구성하고 적재해야 하는 오버헤드가 존재한다. 따라서 Net-SNMP는 성능이 제한적인 임베디드 시스템이나 소규모 시스템에 사용하기에는 적당하지 못하다.

본 논문에서는 SNMP를 이용하는 작고 쉬운 라이브러리 ES-SNMP (Embedded System SNMP)를 구현하고 이를 기반으로 하여 SNMP 에이전트 및 시스템 모니터링 소프트웨어를 구현하였다. 본 라이브러리의 목적은 국제표준 프로토콜인 SNMP의 형식을 그대로 따르지만 불필요한 데이터 적재나 연결설정의 회피로 그 성능을 높이고, 편리한 API (Application Programming Interface)를 제공하여 프로그래머가 보다 편리하게 SNMP 응용 프로그램들을 개발할 수 있게 하는 것이다. 본 논문은 Ⅱ장에서 라이브러리의 동작원리를 기술하고 및 구현한 결과물의 성능을 비교하여 평가한다. Ⅲ장에서는 해당 라이브러리를 이용한 간단한 모니터링 시스템 구축에 대한 설명을 하며, Ⅳ장에서 결론으로 끝맺는다.

Ⅱ. SNMP 라이브러리의 구현 및 성능 평가

1. SNMP의 동작 원리

SNMP는 관리 대상이 되는 에이전트(컴퓨터 또는 네트워크 장비)와 관리를 하는 주체인 매니저로 구성이 되어있다. SNMP의 동작 원리는 매니저의 요청과 에이전트의 응답이라는 방식을 기본으로 동작하며 UDP 프로토콜의 161번 포트를 이용하여 통신한다[7]. 요청과 응답을 사용하지 않고 에이전트가 능동적으로 매니저에게 메시지를 보내는 방안으로는 Trap 메시지를 UDP 162번 포트를 이용하여 전송한다.

SNMP 메시지는 그림 1과 같이 버전, 커뮤니티, 및 SNMP PDU로 구성되어 있다[8]. 버전은 해당 SNMP 메시지의 버전으로 현재 3까지 존재한다. 커뮤니티는 에이전트 객체에 대한 권리를 부여하기위해 사용 한다. 해당 메시지의 종류를 나타내는 PDU Type에는 에이전트의 정보를 읽기위한 GetRequest 와 GetNextRequest, 정보를 설정하기위한 SetRequest, 및 에이전트가 일방적으로 매니저에게 전송하는 Trap이 있다. 모든 응답은 GetResponse로 처리한다. 특정 응답이 어느 요청에 대한 것인지를 구별하기위해서 Request id가 사용된다.

Variable-binding은 여러개의 <OID, value> 쌍들로

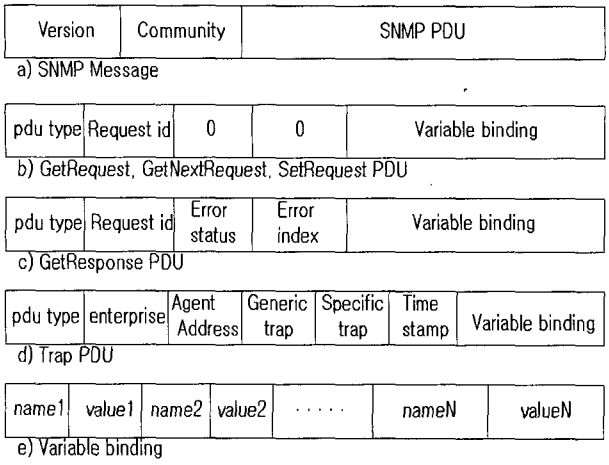


그림 1. SNMP PDU의 구조
Fig. 1. Structure of SNMP PDU.

이루어지는데 OID (Object IDentifier)는 특정 데이터 항목을 구별하는 식별자이다.

GetRequest와 GetNextRequest에서는 매니저가 value 부분을 비운 상태에서 전송하면 에이전트가 각 OID에 해당하는 값을 이 부분에 채워서 매니저에게 응답한다. SetRequest의 경우에는 수정하고자하는 값을 value 부분에 지정하여 전송한다. SNMP 프로토콜 형식에 맞지 않는 메시지들에 대해서는 에이전트는 아무런 메시지도 회신하지 않는다. 따라서 매니저는 요청 메시지를 전송할 때 최대 대기시간을 설정해 두어야 한다. Trap 메시지에서는 에이전트가 매니저에게 보내고자 하는 정보를 <OID, value>의 쌍으로 기록하여 전달한다.

에이전트가 제공 가능한 모든 데이터 항목들은 MIB (Managemet Information Base)에 객체로 정의 되어 있는데 이들은 트리 형태로 계층화되어 있다[10]. 그림 2는 SNMP 객체 트리의 상위부분을 보여주고 있다. 예를 들어서 IP와 관련된 객체의 OID는 "1.3.6.1.2.1.3"으로 시작하며, 수신한 총 IP 패킷 수를 나타내는 ipInReceives는 "1.3.6.1.2.1.3.3"이 된다.

SNMP 객체들 중에서 일반적으로 많이 사용 되는 것은 일반적인 트래픽 관리를 위한 mgmt와 시스템 제조업체나 운영체제 제공업체에서 제공하는 private 객체들이다. Private 객체들은 해당 시스템의 특수한 정보를 담고 있는 것으로서 에이전트의 제조업체들이 설정한 여러 가지 시스템 관련 정보를 담고 있다. SNMP의 일차 목적은 네트워크 관리이지만 본 논문에서와 같이 시스템 관리를 위한 부분으로는 mgmt의 system 객체들과 private 객체들이다.

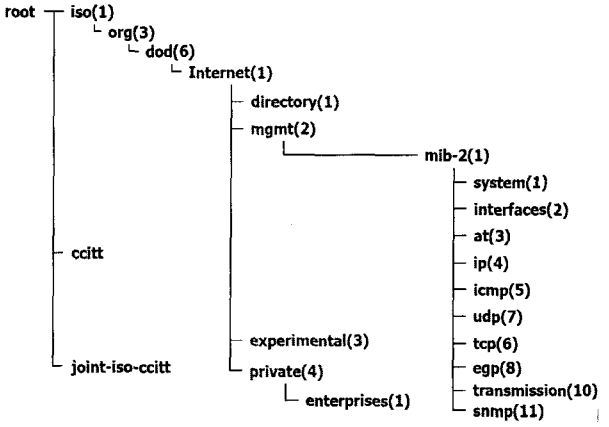


그림 2. SNMP 객체들의 계층 구조
Fig. 2. Hierarchy SNMP Objects.

System 객체들로는 시스템의 설명에 해당하는 sysDescr(1), 시스템을 가동시킨 시점을 나타내는 sysUpTime(3), 시스템의 이름 sysName(5), 시스템의 위치 정보 sysLocation(6) 등이 있다. Private 객체들은 시스템 제조업체별로 독자적인 정보들을 제공하는데 이들을 이용하여 시스템 관리를 위한 정보들을 수집할 수 있다. MIB의 각 객체들에 대한 상세한 내용은 RFC1155의 규칙에 따르는 SMI 문서에 기술된다[12]. Private 객체들에 대한 정보는 각 업체에서 제공하는 SMI 문서를 통하여 얻을 수 있다.

2. SNMP 라이브러리의 구현

본 논문에서 구현한 ES-SNMP 라이브러리는 SNMP에서 정의하고 있는 4가지의 GetRequest, GetNext-Request, SetRequest, 및 Trap 메시지 전부를 처리하기 위한 함수들을 제공한다. 소규모의 시스템에서 시스템에 부하를 줄이는 방향으로 설계하기 위해서 Net-SNMP에서 사용하는 초기화 및 세션 설정과정을 생략하고 각 함수는 독립적으로 동작할 수 있도록 하였다. 또한 PDU 인코딩 및 디코딩 과정에서 객체들에 대한 정보를 중간 단계의 범용적인 형식으로 재구성하는 과정도 생략할 수 있도록 하여 실행 속도를 높이도록 하였다.

매니저가 에이전트에게 요청 메시지를 전송하는 함수에서는 타임아웃 시간과 재시도 횟수를 설정하여 UDP 망의 특성상 메시지가 분실되는 경우를 대비하였다. 또한 내부적으로 매 요청시마다 에이전트의 가동시간을 같이 요청하여 각 OID 데이터에 대한 수집 시점을 알수 있도록 하였다. 이는 에이전트의 특정 객체 값에 대하여 시간에 따른 변화량을 쉽게 계산할 수 있게

해준다.

에이전트와 매니저 교환되는 모든 메시지는 플랫폼에 독립적인 표현 형식인 ASN.1 (Abstract Syntax Notation One)의 형식에 따라 정의되어 있다[17]. 메시지에 포함될 각 데이터들은 ASN.1의 정의에 의한 데이터 타입을 가지고 있으며, 이들은 해당 데이터 타입을 가지고 BER (Basic Encoding Rule)에 의해 인코딩하여 네트워크 상에서 송수신 될 수 있는 메시지를 구성하게 된다[18].

GetRequest 메시지를 활용하여 에이전트의 특정 값을 읽어오기 위한 함수로는 다음과 같이 snmpGetTimeout를 제공한다. 여기서 uptime은 대상 에이전트의 가동 시작 시간 (sysUpTime)으로서 SNMP 표준에 의해 정의된 대로 10msec 단위이다. Type은 읽어온 객체의 값이 어떤 형인지를 반환하도록 한다. Timeout 값은 1msec 단위로 지정하며, 이 시간동안 에이전트로부터 응답이 없으면 동일한 PDU를 다시 전송한다. Retry 값은 재전송하는 최대 회수를 지정한다. 최종적으로 실패하면 -1을 반환하고, 에이전트로부터 성공적으로 응답메시지를 받으면 0을 반환한다. Timeout과 retry 값을 지정하지 않고 사전에 지정된 값을 사용할 경우에는 다음과 같이 단순화된 형태인 snmpGet 함수를 사용할 수 있다.

SetRequest 메시지를 활용하여 에이전트의 특정 값을 설정하는 함수로는 snmpSetTimeout을 사용하며, 인수들로는 type 인수를 정수로 지정하는 것 외에는 snmpGetTimeout 함수와 동일하다. SetRequest에서도 단순한 형태인 snmpSet 함수를 제공하며, 한번에 여러 개의 객체들을 처리하기 위하여 snmpGetMulti와 snmpSetMulti를 제공한다. GetNextRequest 메시지를 이용하여 지정된 객체 바로 다음 객체의 인스턴스 OID와 값을 읽어오는 함수로는 snmpGetNext를, 지정된 객체 하부의 모든 객체 인스턴스들의 OID와 값들을 읽어오는 함수로는 snmpGetInstanceList를 제공한다. SNMP 매니저가 에이전트로부터 수신한 Trap 메시지를 검사하고 읽어내기 위한 함수로는 snmpTrapPoll을 제공한다.

SNMP 표준에서 규정하고 있는 기능을 위한 함수 이외에도 응용 프로그래머의 편리를 위한 몇 몇 함수가 추가로 제공되는데, 일정한 IP 주소 영역 내의 모든 SNMP 에이전트들의 목록을 검색하는 snmpAgentSearch, 타임아웃과 재시도 회수를 사전에 설정하기위한 snmpConfigTimeout 및 snmpConfig-

표 1. ES-SNMP의 주요 라이브러리 함수
Table 1. Library of ES-SNMP.

```
int snmpGet(unsigned long ipaddr, unsigned long
  *sysuptime, char *oid, int *type, char *value)
int snmpGetTimeout(unsigned long ipaddr,
  unsigned long *sysuptime, char *oid, int *type,
  char *value, int timeout, int retry)
int snmpGetNext(unsigned long ipaddr, char *oid,
  char *next, int *type, char *value)
int snmpSet(unsigned long ipaddr, char *oid, int
  type, char *value)
int snmpSetTimeout(unsigned long ipaddr, char
  *oid, int type, char *value, int timeout, int retry)
int snmpTrapPoll(unsigned long *sender, char
  *enterprise, unsigned long *ipaddr, int *traptype,
  int *specific, int *timestamp, int *count, char
  *oid[], int type[], char *value[])
int snmpGetInstanceList(unsigned long ipaddr, char
  *node, int maxcount,
      char *instance[], int type[], char *value[])
int snmpAgentSearch(unsigned long from, unsigned
  long to, int maxcount, unsigned long agentlist[],
  char *name[])
int snmpOid2Name(char *mibfile, char *oid, char
  *name, int *type, int *mode)
int snmpName2Oid(char *mibfile, char *name, char
  *oid, int *type, int *mode)
```

Retry, 숫자로 구성된 OID를 문자열로 반환하는 snmpOid2Name, 문자열 이름으로부터 숫자로 구성된 OID를 반환해주는 snmpName2Oid를 제공한다.

3. SNMP 라이브러리의 성능평가

본 논문에서 구현한 ES-SNMP 라이브러리의 성능을 평가하기 위하여 일반적으로 널리 사용하는 Net-SNMP 라이브러리와 실행에 소요되는 시간을 측정하여 비교하였다. 성능 측정을 위해서 본 논문에서 구현한 ES-SNMP와 Net-SNMP 라이브러리를 각각 사용하는 SNMP 매니저를 구현하고 실행 시간을 측정하였다. 사용한 시스템은 펜티엄 기반의 리눅스(커널 버전 2.2) 시스템이며, Net-SNMP는 libsnmp-0.4.2 버전을 사용하였다.

표 2. 실행시간 비교
Table 2. Comparison of Execution Time.

	평균 응답시간 (msec)	라이브러리 시간 (msec)
Net-SNMP	232	177
ES-SNMP	54	9

표 2는 ES-SNMP와 비교대상인 Net-SNMP의 실행 시간을 측정하여 비교한 결과이다. 시간측정은 동일한 GetRequest 메시지를 100번 요청하고 응답하는 응용프로그램을 작성하여 해당 프로그램이 수행하는 시간을 측정하였다. 응용프로그램은 순수하게 라이브러리가 제공하는 함수만을 호출하여 자신이 원하는 객체의 값을 얻어오기만 한다. 이 프로그램을 100회 반복하여 실행하고 그 평균 응답시간을 측정하였다. 에이전트는 동일하게 Net-SNMP 기반의 에이전트를 적용하였다. 따라서 순수하게 매니저를 위해 사용한 SNMP 라이브러리의 차이에 의한 시간 차이를 측정한 것이다.

표 2에서 평균응답시간은 GetRequest 메시지를 생성하고, 전송하고, 이에 대한 응답 메시지를 받고, 디코딩하여 그 내용을 확인할 때 까지의 평균적인 시간을 의미한다. 비교대상인 Net-SNMP가 232msec 인데 반해서 ES-SNMP는 54msec로 대폭 개선된 것을 확인할 수 있다.

평균응답시간에는 네트워크를 통해 송수신하는 시간지연과 에이전트가 요청 메시지를 읽고 응답 메시지를 생성할 때까지의 시간지연 및 운영체제에서 소모하는 시간까지가 모두 포함된 것이다. 표 1에서 라이브러리 시간은 매니저 프로세스가 사용자 모드에서 실행한 시간만을 측정한 것으로서 부가적인 지연 요소들을 배제하고, 순수하게 SNMP 라이브러리에 의해 실행된 시간 및 약간의 부가적인 오버헤드를 포함한 결과를 의미한다. 하나의 GetRequest를 처리하는데 소요되는 평균적인 라이브러리 실행 시간은 Net-SNMP가 177msec 소요된 것에 비하여 ES-SNMP에서는 9msec로 대폭 개선된 것을 확인할 수 있다.

ES-SNMP가 Net-SNMP에 비해서 실행시간을 대폭 단축할 수 있는 근거는 Net-SNMP에서 사용하는 초기화 및 세션 설정과정을 생략하고 각 함수는 독립적으로 동작할 수 있도록 하였으며, PDU 인코딩 및 디코딩 과정에서 객체들에 대한 정보를 중간 단계의 범용적인 형식으로 재구성하는 과정을 생략할 수 있도록 하는데 기안한다. Net-SNMP 라이브러리를 분석해 보면 하나의 GetRequest 메시지를 전송하기 위해서 매니저에서 총 10회 가량의 여러 가지 라이브러리 함수들이 호출되지만, ES-SNMP에서는 하나의 함수만 호출하면 된다. 또한 ES-SNMP에서는 부가적인 세션 처리와 복잡한 데이터 분석 및 복사과정이 없이 각 요청을 별개의 것으로 간편하게 처리하였다.

Net-SNMP의 경우 라이브러리 내부적으로 GetRequest나 SetRequest를 위한 간결한 함수가 존재하지 않으며, GetRequest 요청을 1회 하기 위해 함수 초기화, 세션 초기화 및 설정, PDU 생성, 메시지 전달, 데이터 분석 등의 여러 단계의 함수를 사용해야 한다. 그 결과 각 함수들을 사용하는데 있어서 많은 오버헤드가 존재한다. 또한 사용되는 정보를 특정 구조체로 저장하여 이것을 여러 개의 함수에 전달해 가며 사용하는 것이 많은 시간을 소모하는 원인중 하나이다. 이러한 오버헤드는 Net-SNMP를 다양한 분야에서 모두 사용할 수 있도록 일반화하는 과정에서 초래된 것이다.

IV. 모니터링 시스템의 구축

1. 시스템 모니터링

많은 수의 시스템들을 체계적으로 모니터링하기 위해서는 모니터링 대상 시스템들에서 필요한 데이터들을 수집하여 사용자에게 일목요연하게 보여주어야 한다. 본 논문에서는 간단하면서도 오버헤드가 적게 구현한 ES-SNMP를 활용하여 간단한 모니터링 시스템을 구현하였다. 모니터링 시스템은 기본적으로 그림 3에서 보는바와 같이 3가지 요소로 구성되어 있다. 모니터링 대상 시스템마다 SNMP 에이전트들을 설치하고, 모니터링 서버는 SNMP 매니저 역할을 통해 에이전트들로부터 GetRequest 요청을 통해 필요한 데이터들을 수집한다. 사용자는 클라이언트 프로그램을 통하여 모니터링 서버에 수집된 데이터를 그래픽 화면으로 볼 수 있도록 한다. 모니터링 서버는 수집된 모든 데이터들을 DB에 저장하고 적절히 가공하여 클라이언트 프로그램에게 제공한다.

사용자가 직접 보게 되는 클라이언트 프로그램은 Java를 사용한 독립된 프로그램과 웹 브라우저를 통해 어디서나 볼 수 있도록 하는 웹 버전으로 2가지를 구현하였다. 클라이언트 프로그램은 일목요연하게 모니터링 대상 시스템들의 상태를 사용자에게 보여주는 역할을 하며, 긴급 상황이 발생하면 경보 및 통신 수단을 통해 관련자들에게 정보를 제공할 수도 있다.

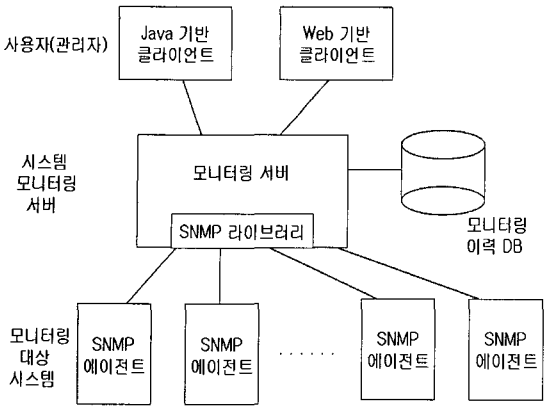


그림 3. 모니터링 시스템 구성도
Fig. 3. Organization of the Monitoring System.

모니터링 서버는 리눅스 환경에서 C언어로 제작하였으며, DBMS로는 MySQL을 사용하였다. Java 기반 클라이언트 프로그램은 JAVA2의 Swing을 이용하여 작성하였고, 웹 기반 클라이언트 프로그램은 JSP를 이용하여 작성되었으며, 웹 서버로는 httpd-2.0.50 Tomcat 5.0을 설치하여 사용하였다.

모니터링 서버 프로그램은 ES-SNMP 라이브러리를 이용하여 등록된 에이전트에게 주기적으로 SNMP GetRequest 메시지를 전송하여 원하는 정보를 수집한다. 그리고 수집된 정보를 적절히 가공하여 각 클라이언트에게 전달한다. 에이전트는 모니터링 대상마다 설치되어서 필요한 정보를 수집하고 모니터링 서버가 요청하면 넘겨준다. 에이전트로부터 수집된 정보는 적절히 가공되어 클라이언트 프로그램에게 실시간으로 전송하게 된다. 만일 이러한 정보가 정상상태를 벗어나는 경우에는 모니터링 이력 데이터베이스에도 저장하여 사후에 긴급 상황들에 대한 원인 분석과 통계자료로서 활용할 수 있게 한다.

모니터링을 위해 수집할 정보는 목적에 따라 아주 다양할 것이다. 본 논문에서 구현한 서버 프로그램에서는 일반적으로 관심의 대상이 될 수 있 것들로서 네트워크 사용율, 트래픽 에러율, CPU 사용율, 메모리 사용율을 대상으로 하였다. 이 외에도 에이전트의 기본적인 정보(IP 주소, 시스템 이름, 연락처, 시스템 위치) 들을 에이전트로부터 읽어온다.

네트워크 트래픽의 사용율과 에러율은 일반 MIB의 객체가 아니므로 모니터링 서버에 의해 다음과 같이 계산한다. 여기서 DISCARDS는 ipInDiscards와 ipOutDiscards의 합이고 DELIVERS는 ipInDelivers와 ipOutDelivers의 합이다.

$$\text{네트워크 사용율} = \frac{ifInOctets + ifOutOctets}{ifSpeed} \quad (1)$$

$$\text{트래픽 에러율} = \frac{DISCARDS}{DELIVERS + DISCARDS} \quad (2)$$

모니터링 대상 시스템은 관리의 대상이 되는 시스템으로서 컴퓨터나 네트워크 장비 등이 포함된다. 각 시스템에는 SNMP 에이전트가 필요한데 시스템의 제조회사에서 이미 설치한 상태로 출하하거나, 범용컴퓨터의 경우 사용하는 운영체제에 따라 적절하게 설치되어 있다. 필요에 따라서는 SNMP 에이전트를 직접 개발하여 탑재할 수도 있다. 현재 구현된 모니터링 시스템은 Windows와 Linux 컴퓨터를 모니터링 대상으로 하고 있다. Windows의 경우 이미 포함되어 있는 SNMP 서비스를 활성화함으로써 에이전트 프로세스가 실행되고, Linux의 경우 Net-SNMP을 가동함으로서 에이전트 시스템의 역할을 수행하게 된다. 대부분의 네트워크 장비들은 제조업체에 의해 기본적으로 SNMP 에이전트 수행 기능을 갖추고 있다.

기본적인 정보 이외에 제조회사나 운영체제의 종류에 따라 특수한 (private) MIB가 제공되는데 이 정보를 이용하려면 해당 MIB 별로 제공정보들을 파악할 필요가 있다. 하지만 공통적으로 제공되는 MIB의 객체를 요청할 경우에는 에이전트의 플랫폼에 상관없이 정보를 요구할 수 있다. 단지 에이전트에서는 SNMP 서비스를 제공해주기만 하면 된다. 결국 공통적으로 제공되는 MIB의 정보를 요청하는 서버 프로그램의 경우는 특별한 플랫폼에 영향을 받지 않고 네트워크에 연결된 컴퓨터나 네트워크 장비들에게 정보를 요청할 수 가 있으며, 그들을 관리할 수가 있는 것이다. SNMP는 기본적으로 네트워크 장비들을 목표로 작성되었으므로 통신에 관련되는 네트워크 사용율과 트래픽 에러율은 표준 MIB 정보로부터 얻을 수 있지만 CPU 사용율이나 메모리 사용율 등은 운영체제에 따라 독자적으로 제공되는 private MIB 정보를 활용해야 한다.

3. 모니터링 클라이언트

클라이언트 프로그램은 모니터링 서버와 독자적인 방법으로 정보를 교환하고 모니터링 상태에 대한 정보를 사용자에게 적절한 방법으로 보여준다. 각 모니터링 항목이 미리 설정된 수치를 넘어서게 되면 이상 상태로 표시해 주고 필요에 따라 경보음을 울리거나 단문

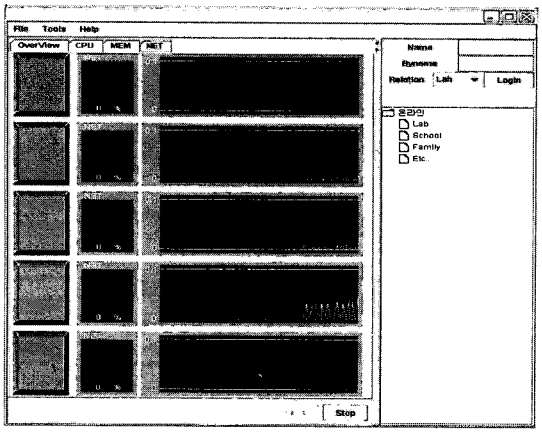


그림 4. Java 기반 클라이언트의 실행화면
Fig. 4. Java-Based Client.

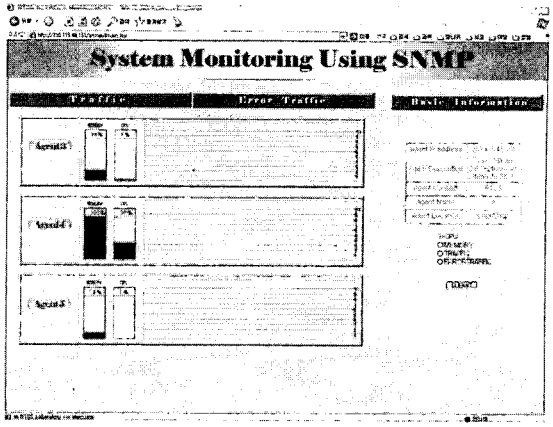


그림 5. 웹 기반 클라이언트의 실행 화면
Fig. 5. Web-Based Client.

메시지, 전자우편 등을 통하여 관련된 사람들에게 긴급 정보를 전달할 수도 있다. 클라이언트 프로그램은 별도로 설치된 컴퓨터에서만 실행되는 독립 프로그램으로 작성될 수도 있고 웹 브라우저를 통하여 어디서나 접속하여 상태를 파악하도록 할 수도 있다. 본 논문에서는 Java2의 Swing을 활용하여 독립된 클라이언트 프로그램을 구현하였고, JSP를 활용한 웹 프로그램으로도 구현하였다.

그림 4는 Java 언어로 작성된 클라이언트 프로그램을 실행한 화면의 예를 보여 준다. 모든 에이전트의 CPU 사용율, 메모리 사용율, 및 네트워크 사용율의 상황을 제공하는 오버뷰 기능을 게이지로 제공 한다. 각 에이전트마다 CPU, 메모리 및 네트워크의 3개의 게이지 표시가 제공된다. 세부 메뉴 (네트워크, CPU, 메모리)로 들어가면, 그림 4와 같이 현재의 수치를 수직의 눈금으로 보여주고 시간에 따른 변화는 꺽은 선 그래프로 표현하여 준다. 그래프의 눈금은 백분율로 제공하는데 네트워크 사용율의 경우 해당 백분율의 격차가 커 네트워크 상황에 따라 최대값을 조절하고 네트워크의 사용율은 여기에 맞춰서 적절한 크기로 그래프를 그린다.

맨 앞의 사각형에는 에이전트의 상태를 색으로 표현해준다. 정상상태는 녹색, 위험상태는 노란색, 비상상태는 빨간색으로 표시한다. 또한 이 사각형은 각각의 에이전트를 나타내며 클릭 시 에이전트의 상세 정보를 화면에 보여준다. 이 상세 정보에는 에이전트의 이름, 위치, 설명, 관리자 연락처 등이 적혀있으며, 이러한 정보를 이용해 에이전트 소유자에게 시스템 관리자가 e-mail 및 간단한 메시지를 전송할 수 있다. 에이전트 블록의 윗부분에는 오버뷰, CPU, 메모리 등 다른 화면으로의 전환 탭이 있다. 각 탭을 이용하여 쉽게 다른 상태를 나타내는 창으로 전환이 가능하며 각 에이전트의 상태를 알 수 있다.

웹 기반 클라이언트 프로그램은 시스템 관리자가 어느 곳에 있어도 웹 브라우저를 통해 시스템의 상황을 확인할 수 있게 한다. 독립적인 클라이언트 프로그램처럼 많은 기능을 제공할 수는 없지만 간단하게 CPU 사용율, 메모리 사용율, 네트워크 사용율 등을 감시할 수 있다. 그림 5는 웹 기반의 클라이언트 프로그램을 실행한 화면을 보여주고 있다. 앞의 두 개의 게이지는 CPU 사용율과 메모리 사용율을 각각 보여준다. 꺽은 선 그래프는 네트워크 사용율의 시간에 따른 변화를 보여주고 있으며 상단 부분의 버튼으로 네트워크 에러율도 볼 수 있다. 만일 특정 데이터가 정해놓은 한계치를 넘어설 경우 브라우저 창 위로 에이전트의 번호와 어떠한 데이터 값이 넘었는지에 대한 경고창이 뜬다.

좌측 프레임에 있는 각 에이전트별 버튼을 선택하면 해당 에이전트의 상세 정보를 오른쪽 프레임에서 볼 수 있다. 이는 Java 기반 프로그램과 같은 내용을 담고 있다. 이러한 상세 정보 밑에는 각 에이전트마다의 에러 리포트를 볼 수 있는 버튼이 있다. 이 버튼은 관리하는 4가지 항목 (CPU, 메모리, 네트워크, 에러) 각각에 대한 보고서를 볼 수 있는 페이지를 보여준다. 해당 리포트는 서버 프로그램이 저장해놓은 이력 데이터베이스에서 추출해 오는 것으로서 각각의 항목에 대해 일별, 월별로 해당 항목이 언제 기준치대비 얼마를 초과했는지 볼 수 있는 페이지이다. 이러한 데이터베이스를 적절히 이용하여 통계자료로 사용하거나, 관리자가 자리를 비운 시간에 각 에이전트에 대한 상태 조사 등을 위해 이용된다.

Ⅳ. 결 론

SNMP를 이용하여 시스템 모니터링을 구현할 때 일반적으로 널리 사용되는 일반적인 SNMP 라이브러리들을 사용하면 크기와 속도 면에서 부담이 크다. 또한 이것을 활용하여 시스템을 구축하기 위해서는 학습해야 하는 내용도 방대하다. 이들은 많은 기능을 가지고 있어서 고성능의 하드웨어를 사용하는 관리 시스템에서는 그 활용 가치가 높겠지만, 임베디드 시스템과 같은 소규모 시스템이나 소규모 네트워크에 적용하기에는 무리가 따른다.

본 논문에서 구현한 ES-SNMP 라이브러리는 기존의 Net-SNMP에 비해서 속도를 대폭 개선하고 필요한 메모리의 양도 줄였다. 소규모의 시스템에서 시스템에 부하를 줄이는 방향으로 설계하기 위해서 Net-SNMP에서 사용하는 초기화 및 세션 설정과정을 생략하고 각 함수는 독립적으로 동작할 수 있도록 하였다. 또한 PDU 인코딩 및 디코딩 과정에서 객체들에 대한 정보를 중간 단계의 범용적인 형식으로 재구성하는 과정도 생략할 수 있도록 하여 실행 속도를 높이도록 하였다.

ES-SNMP 라이브러리를 활용하여 네트워크로 접속된 많은 수의 시스템들을 일괄적으로 감시할 수 있는 모니터링 시스템의 구현 방안을 제시하고 구현하였다. 이를 위한 클라이언트 프로그램으로서 Java 기반의 프로그램과 웹 기반의 프로그램을 각각 구현하였다. 시스템 모니터링에 SNMP를 적용하면 감시대상 시스템들에 별도의 프로그램을 설치할 필요가 없이 이미 가지고 있는 SNMP 에이전트를 활성화 시키는 것만으로 가능하다. 서버에서는 ES-SNMP 라이브러리를 활용하여 감시대상 시스템들의 SNMP 에이전트들을 통하여 필요한 정보들을 수집하고 DB에 저장하며 적절히 가공하여 클라이언트 프로그램으로 전달해 준다.

본 논문에서 구현한 ES-SNMP 라이브러리 및 이를 활용한 시스템 모니터링 방안은 소형 임베디드 시스템이나 소규모 시스템들을 간편하게 관리하기에 적합하다. 시스템 별로 각 자원의 사용율과 네트워크의 상태 감시를 통해 관리대상이 되는 에이전트 시스템의 문제를 즉시 인식하고 그에 대한 대처를 할 수 있으며 사후에는 이력 정보를 검토하여 문제의 원인을 분석할 수도 있다. ES-SNMP 라이브러리를 활용하면 시스템 자원에 대한 부하가 적은 매니저 프로그램으로서 가벼운 모니터링 시스템을 구축 할 수 있다.

참 고 문 헌

[1] S. Hochstetler, *IBM Tivoli Monitoring Version 5.1: Advanced Resource Monitoring,* IBM Redbooks, 2002.

[2] J. Blommers, *OpenView Network Node Manager,* Prentice Hall, 2000.

[3] EasyMon, *"http://wasymon.net/easymon_01.htm"*

[4] P. Dinda et al., *"The Architecture of the Remos System,"* Proc. 10th Int. Symp. on High Performance Distributed Computing (HPDC'01), pp. 1-14, 2001.

[5] P. Bellavista et al., *"How to Monitor and Control Resource Usage in Mobile Agent Systems,"* Proc. 3rd Int. Symp. on Distributed-Objects and Aplications (DOA'01), 2001.

[6] M. Marzolla, *"A Performance Monitoring System for Large Conputing Clusters,"* Proc. 11th Euromicro Conf. on Parallel, Distributed and Network-Based Processing (Euro-PDP'03), 2003.

[7] D. Mauro and K. Schmidt, *Essential SNMP,* O'Reilly, 2001.

[8] W. Staling, *SNMP, SNMPv2, SNMPv3, and RMON1 and 2, 3rd ed.,* Addison-Wesley, 1999.

[9] M. A. Miller, *Managing Internet works with SNMP,* 1997.

[10] RFC 1757, "Remote Network Monitoring Management Information Base", 1995.

[11] RFC 1157, "Simple Network Management Protocol", 1990.

[12] RFC 1155, "Structure of Management Information", 1990.

[13] RFC 1905, "Protocol Operations for SNMPv2", 1996.

[14] RFC 1155, "Structure and Identification of Management Information for TCP/IP-Based Internets", 1990..

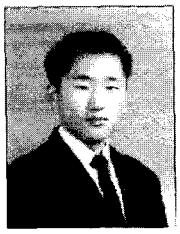
[15] Net-SNMP, *"http://www.net-snmp.org"*

[16] SourceForge, *"http://sourceforge.net/projects /net-snmp"*

[17] ITU-T X.680-683, Abstract Syntax Notation One (ASN.1), 2002.

[18] ITU-T X.690, ASN.1 Encoding Rules - Specification of Basic Encoding Rules (BER), Cannonical Encoding Rules (CER), and Distinguished Encoding Rules (DER), 2002.

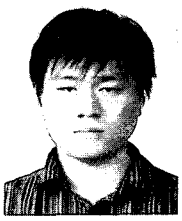
저 자 소 개



김 태 수(학생회원)
2005년 강원대학교
컴퓨터공학과 학사 졸업.
2006년 현재 강원대학교 대학원
컴퓨터정보통신공학과
석사과정 재학

<주관심분야 : 컴퓨터, 통신, 임베디드 컴퓨팅 >

김 용 석(정회원)
1984년 서울대학교 해양학과
학사 졸업.
1986년 KAIST 전기및전자공학과
석사졸업.
1989년 KAIST 전기및전자공학과
박사 졸업.
1990년～1995년 한국생산기술연구원 및
전자부품연구원 선임연구원
1995년～현재 강원대학교 컴퓨터학부
컴퓨터정보통신전공 교수
<주관심분야 : 운영체제, 실시간시스템, 임베디드 시스템>



김 동 억(학생회원)
2005년 강원대학교
컴퓨터공학과 학사 졸업.
2006년 현재 강원대학교 대학원
컴퓨터정보통신공학과
석사과정 재학

<주관심분야 : 컴퓨터, 통신, 운영체제, 임베디드 컴퓨팅>