[UCI]I804:24011-200000266703

2018年 2月
석사학위논문

# 유무선 혼합 AMI시스템을 위한 이중 역할 망 관리 에이전트 구현

조 선 대 학 교
산업기술융합대학원

소프트웨어융합공학과

권 혁 록

# 유무선 혼합 AMI시스템을 위한 이중 역할 망 관리 에이전트 구현

Implementation dual-role network management agent for wired / wireless hybrid AMI system

2018년 2월 23일

조선대학교
산업기술융합대학원
소프트웨어융합공학과
권 혁 록

# 유무선 혼합 AMI시스템을 위한 이중 역할 망 관리 에이전트 구현

지도교수 김 판 구

이 논문을 공학석사학위신청 논문으로 제출함

2017년 10월

조 선 대 학 교
산업기술융합대학원
소프트웨어융합공학과
권 혁 록

# 권혁록의 석사학위논문을 인준함

위원장 조선대학교 교수 반 성 범 (인)

위 원 조선대학교 교수 신 주 현 (인)

위 원 조선대학교 교수 김 판 구 (인)

2017년 11월

조 선 대 학 교

산업기술융합대학원

# 목 차

# 그림 목차

# 표 목차

# ABSTRACT

## Implementation dual-role network management agent for wired / wireless hybrid AMI system

HyukRok Kwon
Advisor : Prof. PanKoo Kim Ph.D.
Department of Software
Convergence Engineering
Graduate School of Industry
Technology Convergence,
Chosun University

In order to construct a AMI system, data transfer from the electronic watt me ter to the meter reading server is transmitted via the modem and the data conc entration unit (DCU). At this time, the network between the modem and the D CU is configured by power line communication (PLC). Remote meter reading on the power line communication base has restrictions on the distance and speed w ith which various kinds of noise and limited transmission capability can commu nicate like the underground section. Also, software that registers a PLC modem, decides an optimal route, and enables communication is a network management agent installed in the DCU. In power line communication, such a network mana gement agent plays an important role.

Where power line communication is impossible in constructing the AMI syste m, it is possible to use a method of building a mixture of various communicati on methods like wireless communication. From the era of IoT (Internet Of Thing s) to the countless end devices, the necessity of management also emerges and t he path to the terminal is not a TCP / IP network but a heterogeneous networ k is connected in a complex manner and a server May be impossible to directly connect to the terminal equipment.

In this paper, we construct a network management agent in a dual structure,

the parent communicates with the server, communicates with the management a gent of another terminal, acquires the information of the terminal device In orde r to provide control so that we can implement JAVA based network manageme nt agent system, we can expect agent effect through experiment and demonstrati on.

# Ⅰ. 서론

## A. 연구 배경 및 목적

AMI(Advanced Metering Infrastructure, 지능형 계량 인프라)는 스마트그리드를 구현하기 위해 필요한 핵심으로서 전자식 전력량계, 통신망, MDMS(Meter Data Management System, 계량데이터관리시스템)와 운영시스템으로 구성되고 전자식 전력량계 내에 모뎀을 설치하여 양방향 통신이 가능한 지능형 전력계량인프라를 말한다[1]. AMI 운영시스템에서는 소비자와 전력회사 간 양방향통신으로 원격검침, 수요관리, 전력소비 절감과 전기품질 향상 등 다양한 융복합 서비스를 제공하게 된다[1].

AMI를 구축하기 위해서는 다른 통신방식에 비해 PLC가 절대적으로 유리하다. 특히 PLC는 새로운 통신망을 구축하지 않고 기존의 전력선을 이용할 수 있기 때문이다. 그래서 현재 국내 AMI는 고속 PLC(Power Line Communication, 전력선 통신) 통신을 이용하여 구축하고 있다. 그러나 고속 PLC 통신을 이용한 AMI를 구축하는 데는 약간의 문제가 발생하고 있다. 지중구간이나 전압 전류가 갑자기 변하는 부분, 예로서 코일(유도성)이나 콘덴서(용량성)의 스위칭시 발생되는 노이즈 때문에 PLC 통신이 잘 안 되는 곳이 발생한다. 이러한 상황에서 AMI를 고속 PLC 통신만으로 구축하기에는 한계가 있다. 그래서 PLC 통신방식과 더불어 무선 등 다양한 통신방식을 병행해야 한다는 필요성이 대두되고 있다.

유무선 혼합 통신방식을 적용하기 위해서 두 가지 방식의 구성을 할 수 있는데, 첫 번째는 DCU에 이기종 망을 직접 연결하는 방법, 두 번째는 DCU 하위에 중계브릿지들을 설치하고, 중계브릿지에 각각 다른 통신방식을 연결하거나, 여러 가지를 통신방식을 동시에 연결하는 방법이 있다. 첫 번째 방식에 망 관리 에이전트를 설치한다면 망 관리 에이전트가 여러 가지 통신방식을 다 지원해야 하는 문제점이 있고, 새로운 통신방식의 출현 시 또 거기에 맞도록 망 관리 에이전트를 수정 개발해야 한다. 두 번째 방식처럼 중계브릿지에 망 관리 에이전트를 설치한다면 서버에서 중계브릿지에 직접 접속 할 수 없는 문제점이 발생하게 된다.

위 두 가지 문제점을 해결하기 위해서 DCU(內)에 또 다른 망 관리 에이전트들로부터 MIB(Management Information Base, 관리정보베이스) 정보를 수집하고 중

계 역할을 하는 이중 역할 매니저를 설치한다면 해결 할 수 있다. 이렇게 이중 역할 망 관리 에이전트를 설치한다면 다양한 통신방식의 망 관리 에이전트는 각기 본연의 통신방식만 지원하면 되고, 새로운 통신방식의 출현 시에도 유연하게 대처할 수 있다.

또한 현재 고속 PLC 통신방식에서의 수 많은 모뎀들의 등록 및 상태를 감시하기 위해 망 관리 시스템을 구축해서 관리하고 있다. 망 관리 시스템 서버는 모든 모뎀들을 직접 제어하지 않고 DCU(Data Concentration Unit, 데이터 집중장치)의 망 관리 에이전트를 통해서 모뎀들의 정보를 취득, 관리하고 있다. 이 DCU(內)에 망 관리 에이전트를 설치 운영하고 있는데, DCU의 메모리 및 MCU(Main Control Unit)의 성능이 낮아서 C++로 모듈이 구현 돼 있다. 대부분의 장비들이 그렇겠지만 임베디드 장비들은 가격적인 부담 때문에 최소한의 제원으로 만들어져 있어 임베디드에 적합한 C++을 사용하여 개발하였다. 그러나 커널 버전이나 컴파일러 버전이 변경 됐을 경우 매번 소스를 수정 컴파일 해야 하는 부담이 존재한다. 그러나 요즘 기술의 발달에 따라 저가의 고속, 대용량의 장비들이 출시되고 있어, JVM(Java Virtual Machine) 탑재가 가능 할 정도로 고사양이 되고 있다. 그래서 본 논문에서는 DCU에 JVM을 탑재하고, 플랫폼 독립적인 JAVA를 이용하여 유무선 혼합 AMI 망 관리를 위한 이중 역할 망 관리 에이전트를 구현한다.

## B. 연구 내용 및 구성

본 연구의 주된 내용은 유무선이 혼합된 복잡한 AMI 구축에 필요한 수많은 DCU 및 모뎀들의 네트워크 망 관리를 위한 자바기반의 이중 역할 망 관리 에이전트 시스템 구현에 관한 것이다. 본 논문을 아래와 같이 구성하였다.

제 1장 서론에서는 연구의 배경 및 목적, 제 2장 관련 연구에서는 전력선 통신과 검침인프라 및 NMS(망 관리 시스템)에 대해서 서술한다.

제 3장에서는 유무선 혼합 AMI 망 관리를 위한 자바기반의 이중 역할 망 관리 에이전트 어플리케이션 설계 및 구현에 관해서 서술한다.

제 4장에서는 본 논문에서 제안하는 방법에 대한 구현 결과와 MIB 데이터 수집 효율성에 대해서 실험 결과를 서술한다.

제 5장에서는 본 연구의 결과와 추가적으로 연구가 필요한 부분에 대해서 서술한다.

# Ⅱ. 관련 연구

유무선 혼합 AMI 망 관리를 위한 자바기반의 이중 역할 망 관리 에이전트 시스템 구현을 위해서 먼저, 전력선 통신과 검침 인프라 구성에 대해서 알아보고, NMS에 대해서 그 내용을 기술 한다.

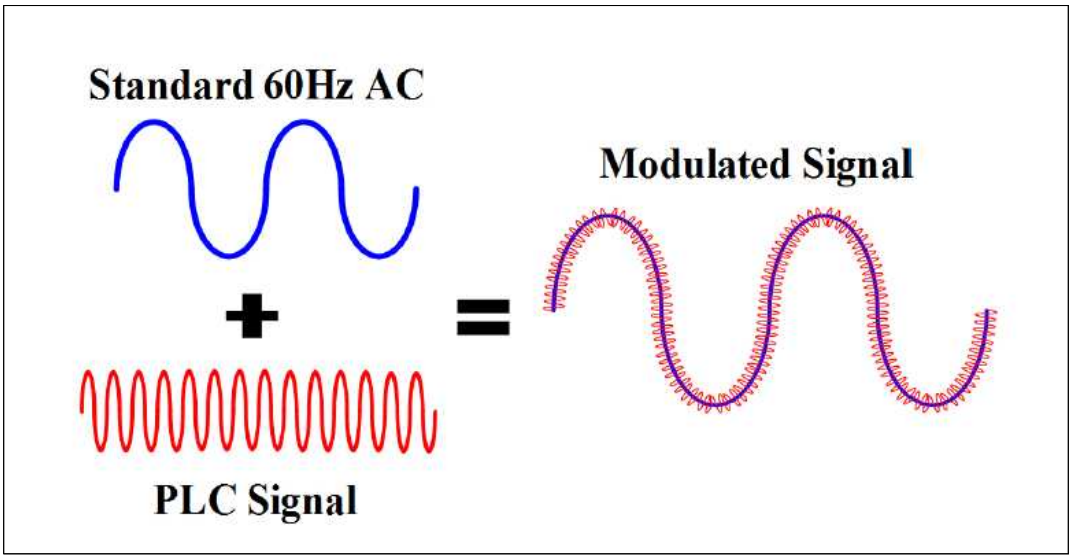
## A. 전력선 통신과 검침 인프라

### 1. 전력선 통신

(1) 전력선 통신 개요

전력선통신(PLC : Power-Line Communications)은 전력선 즉 전기선을 통신 매체로 하여 통신하는 방식으로 신호전달에서 부터 초고속 데이터 통신 네트워크까지 여러 분야에 활용될 수 있는 유선 통신 기술이다. 전력선이라 하면 철탑 등을 상상하기 마련이지만 우리가 일상적으로 전기에너지를 사용하고 있다면 이미 그곳에는 전력선이 존재하고 있으며 이렇듯 전기에너지를 사용하기 위해 필수적으로 이미 확보되어 있는 전력선을 정보 전달에도 활용하고자 하는 것이 전력선 통신의 기본 개념이다[2].

전력선 통신의 원리는 교류전류(50~60Hz)를 공급하는 전력선을 통해 수십 Mhz의 고주파 통신신호를 함께 보내 전용 접속장비로 고주파 신호만을 수신하여 통신하는 것이다[3]. 전력선 통신은 가전제품에 영향을 미치지 않는 저출력의 신호를 사용하고, 통신 신호는 고주파 신호로 바꿔 전력선에 실어 보내고 이를 고주파 필터를 이용하여 별도로 분리해 신호를 수신하는 것이 전력선 통신의 핵심이다[3].



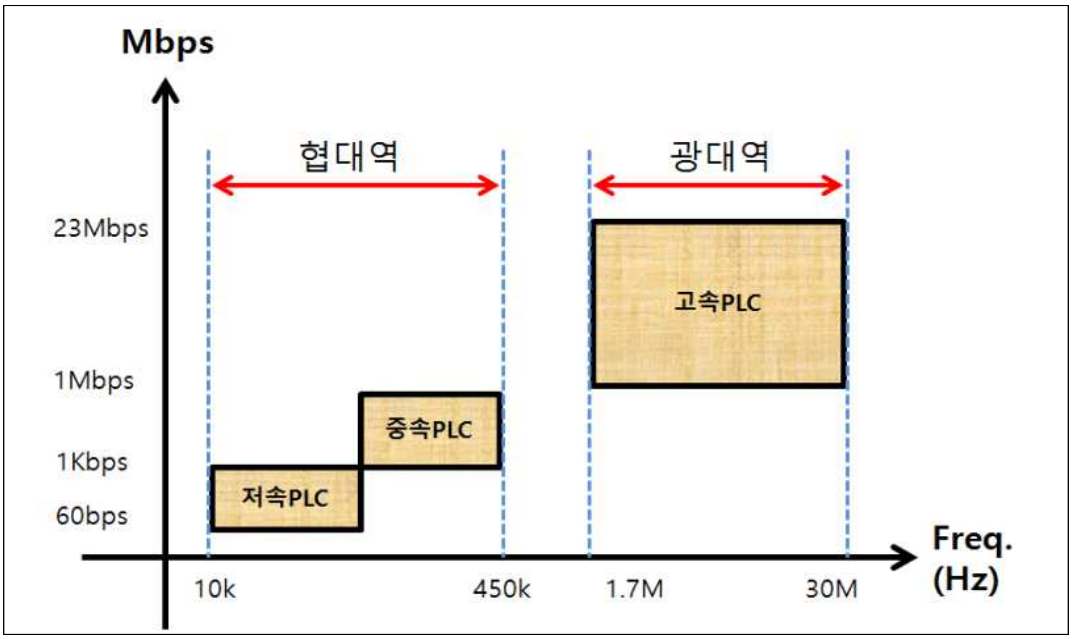

[그림 2-1] PLC 통신 원리[4]

(2) 전력선 통신 종류

PLC는 사용하는 주파수 대역에 따라 협대역(Narrow Band)과 광대역(Broad Band) PLC, 속도에 따라 저속, 중속, 고속 PLC, 사용 영역에 따라 옥내, 옥외 PLC로 구분된다[2].

협대역 PLC는 40~500㎑ 저주파 대역을 사용하고 광대역 PLC에 비해 높은 레벨의 신호 송출이 가능하고, 물리적 신호 감쇄가 비교적 적은 특징이 있어 그 통신 신뢰성이 보다 뛰어나다고 알려져 있다[5].

광대역 PLC는 1.7MHz ~ 30MHz대역을 사용하여 수 Mbps에서 수백 Mbps급의 통신 속도를 가능하게 하며 음성, 데이터, 멀티미디어 신호의 전송 서비스를 제공하는 기술로써, 고속 PLC 또는 고주파 PLC라고도 한다[6].




[그림 2-2] 사용주파수 및 대역에 따른 PLC통신 분류

[표 2-1] PLC통신 분류 기준[6]

| 사용 주파수 | 협대역 | | 광대역 |
|---|---|---|---|
| | 10 ~ 450KHz | | 1.7 ~ 30MHz |
| 전송속도 | 저속 PLC | 중속 PLC | 고속 PLC |
| | 60bps~1Kbps | 1Kbps~1Mbps | 1Mbps이상 |

사용 영역에 따라서는 옥내 및 옥외 PLC로 구분될 수 있는데 하나의 건물 내에 부설된 110/220V 저압배선 케이블을 이용하는 PLC를 옥내 PLC라고 하며 변전소에 고압(22.9kV) 배전선로를 이용하여 통신망을 구성하는 고압 PLC와 주상변압기에서 가정용 전력량계까지 저압 인입선을 이용하는 옥외 PLC가 있다[7].

(3) 전력선 통신의 장.단점

전력선 통신의 장점으로는 별도의 통신선이 없이 기존의 전력시설을 활용하기 때문에 비용이 적게 들고 설치기간이 매우 짧아서 새로운 통신망을 구축하는 것보다 비용측면에서 효율적이다[8]. 또한, 단일 인프라를 통해 음성, 영상, 데이터 및

기타 부가서비스를 보다 쉽게 통합하여 서비스를 제공할 수 있기 때문에 통신, 전력, 및 부가서비스 등을 효과적이고 일괄적으로 제공할 수 있는 장점을 가지고 있다[8]. 즉, 통합서비스를 통한 마케팅 및 고객 유지관리를 위한 비용 절감, 일괄적인 요금 청구 및 이용자 보호 등을 동시에 달성할 수 있다[9].

전력선 통신의 단점으로는 홈 네트워크나 인터넷 기반의 접속과 제어 서비스의 대안으로 등장하기 위해서는 아직도 많은 장애요인들이 존재하며, 고비용의 문제점을 가지고 있어서 현재로서는 기술의 상용화와 비용 우위를 동시에 달성하기는 어렵다[8]. 고비용의 주된 원인은 기술적 불안정성과 그것을 보완하기 위한 추가적인 소요비용과 높은 부하간섭과 잡음현상이 기술의 상용화를 더디게 하고 개발비용 측면에서 불리하다[8].

기술적으로는 가입자접속을 위한 통신선로로써 제한된 전송능력으로 인해 통신 가능 거리와 속도에 대한 제약이 존재하고 가변적이고 높은 감쇄현상, 가변 임피던스 레벨잡음, PLC는 분기가 많아 신호감쇄가 심하며 분기로 인한 주파수 선택적 페이딩 현상이 나타나고 전력선 배치의 구조적 문제로 인하여 가입자 증가시의 폭주 문제로 처리 능력 미비 등 보완이 필요하다[10].

정책적으로는 전력과 통신을 분리하여 시장을 운영하는 각국의 규제정책과 업체 및 국가 간의 표준화에 대한 이견 등이 걸림돌이 되고 있어 동일 장소에서 다른 기종 사용 시 상호 간섭으로 인한 통신 불량의 문제가 발생할 수 있다.[11]

(4) 전력선 통신의 핵심기술 및 장애요인

전력선 통신은 위에서 살펴본 것처럼 여러 가지 문제점들이 존재한다. 따라서 전력선 통신 모뎀을 개발함에 있어 이러한 문제점들을 극복할 수 있는 전송방식이 중요하며, 이와 관련된 PLC 핵심기술들은 아래와 같다.

- 채널 코딩(Channel Coding) : 전력선의 잡음특성과 감쇄특성으로 인해 전력선에 올려질 신호를 부호화 하거나 복호화 하는 기술로, 고속 전력선통신의 경우 잡음에 민감하기 때문에 중요한 기술이며, 주로 사용되는 채널 코딩으로는 Reed Solomon, Carrier Chip, CRC, Optimized FEC, Zero Cross Clocked Carrier, Carrier Chirp, Convolution, Viterb 코드를 사용하고 있다[12].

- 매체 접근 제어(Media Access Control, MAC) : 신호를 일정크기(패킷)로 분할하여 트래픽을 균등히 하고 효율적인 접속이 이루어지도록 프로토콜을 정하고, 이것을 H/W 및 S/W적인 방법으로 구현하여 신호를 빠르고 안정적으로 전송하는 기

술이다[13].

- Analog Front End(AFE) : 전력선에 신호를 실어주거나 전력선에서 통신 신호만 분리해내는 기술로 크게 대역통과 필터링(Bandpass Filtering)과 임피던스 매칭(Impedance Matching) 기술이 있다. 대역통과 필터링은 전력선에서 원하는 신호만 받아들이고 전력선의 각종 잡음을 제거하는 기술이다. 임피던스 매칭은 전력선과 모뎀의 임피던스를 비슷하게 맞추어 최대의 신호 전력이 전달되게 하는 기술이다[14].

- 변복조 기술(Modulation and Demodulation) : 열악한 전력선 채널 특성을 극복하고 전송 속도의 향상을 도모하기 위한 변조 방식으로 주파수 편이(FSK) 방식, 확산 스펙트럼(SS) 방식, 다중반송파(Multi-Carrier) 방식 등을 사용하여 신호를 변.복조 하는 기술이다[13].

## 2. 검침 인프라 구성

(1) 전자식 전력량계

전자식 전력량계는 새로운 통신 기술의 적용과 함께 양방향 통신을 지원하고 사용자에게 전력 사용 정보를 제공하여 DR을 통한 에너지 효율 향상을 촉진시키는 AMI 핵심장치 중 하나이고 프로그램이 가능한 전자식 전력량계로써 다음의 기능을 포함한다[15].

- 시간대별 요금제(TOU), 전기 사용량 데이터 수집
- 양방향 계량(Net-Metering), 정전 및 복전 알림
- 원격 부하 차단/복귀 동작, 선불요금제
- 전력 품질 감시, 도전 감지

[표 2-2] 전자식 전력량계 종류

| 구분 | E-type | 표준형 | Ea-type | G-type |
|---|---|---|---|---|
| 외 형 |  |  |  |  |
| 도 입 | '09년 | '06년 | '15년1월 | '14년9월 |
| 대 상 | 단상 5kW이하 | 단상5kW초과, 삼상 | 단상 10kW이하 | 단상 10kW초과, 삼상 |
| 용 도 | 주택, 농사용 | 상가,소규모 공장 등 | 주택, 농사용 | 상가,소규모 공장 등 |
| 기 능 | 양방향검침(×) | 양방향검침(×) | 양방향검침(O) | 양방향검침(O) |
| | 원격제어(×) | 원격제어(△) | 원격제어(O) | 원격제어(O) |
| | | | CoverOpen감시 | CoverOpen감시 |
| | | | | 전력품질감시 |
| 검침 주기 | 1시간 | 15분 | 15분 | 15분 |
| TOU | × | ○ | ○ | ○ |
| 모뎀 연결 | 외장, RS485 | 내장, IrDA(적외선) | 내장, RS485 | 내장, RS485 |

(2) 검침용 모뎀

모뎀은 전자식 전력량계에 따라서 전력량계 내부 또는 외부에 설치되며, DCU와 전자식 전력량계 사이에서 DLMS 프로토콜을 중계한다. 이에 적용되는 유무선 통신방식은 [표 2-3]과 같다. 국내 한전사업의 경우에는 DCU와 모뎀간 통신방식으로는 국제표준 ISO/IEC 12139-1 방식을 적용한 한국형 고속PLC모뎀과 전자식 전력량계 간에는 전자식 전력량계의 종류에 따라 RS485(E-Type 전력량계)와 IrDA(S-Type, G-Type 전력량계) 방식을 적용하고 있다.[16]

[표 2-3] 검침용 모뎀에 사용되는 주요 통신기술

| 구분 | DCU ↔ 모뎀 | 모뎀 ↔ 전자식 전력량계 | 비고 |
|---|---|---|---|
| 유선 | 저속PLC, 고속PLC | RS232, RS485 | |
| 무선 | B-CDMA, Zigbee,<br>Wi-SUN, TVWS, Lora, etc | IrDA(적외선통신) | |

[표 2-4] 고속PLC 모뎀

| 내장형 모뎀 | 외장형 모뎀 |
|---|---|
|  |  |

(3) 데이터집중장치(Data Concentration Unit, DCU)

DCU는 PLC 네트워크를 지원하기 위해 KEPCO 규격 PLC NMS(Network Management System) Agent를 탑재하고 있으며, 대당 최대 200대의 전자식 전력량계를 수용할 수 있는 처리능력을 보유하고 있고, 하위 전자식 전력량계로부터 취득하는 주요 정보는 정기검침, 현재검침, 계량기 구성정보(Meter ID, 현재시각, 계기정수 등), LP, 전력품질(전류,전압, 역률 등) 등이다[17]. 그리고 검침서버로부터 명령

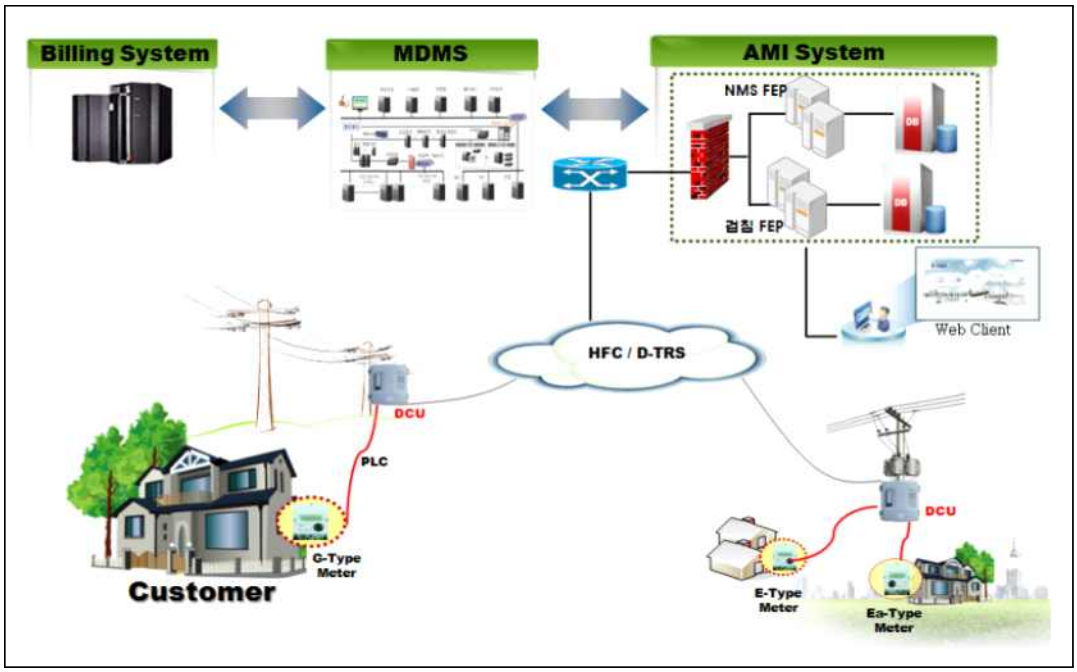
을 수신 받아 처리하고, 검침데이터를 전달하는 기능을 한다. 또한 DCU는 전력선 통신(PLC) 모뎀들을 등록하고 통신 할 수 있도록 최적의 경로를 설정하고 네트워크 망 관리를 수행한다.

[표 2-5] DCU 종류

| 구분 | '08~09년 | '10~12년 | '13~15년 | 옥내형 |
|---|---|---|---|---|
| 외 형 |  |  |  |  |

(4) 검침서버(Meter Data Collect Server)

검침서버는 DCU에 수집된 각종 데이터(정기검침, 현재검침, LP, 정.복전 등)를 HFC(High Fiber Coax)나 GPRS 통신방식을 통하여 수집하여 데이터베이스에 저장 관리하고, DCU나 전력량계의 설정이 필요할 때 원격에서 관리 기능을 수행할 수 있도록 기능을 제공한다. 또한 수집된 검침데이터를 MDMS(Meter Data Management Server)에 전달하는 역할을 수행한다. AMI시스템의 전체적인 구성도는 [그림 5]와 같다.




[그림 2-3] AMI시스템 구성도

(5) MDMS(Meter Data Management System, 계량데이터 관리시스템)

계량데이터 관리시스템인 MDMS는 검침서버로부터 수집된 계량데이터를 유효성 검증, 오검침/누락값 추정, 원격제어 관리, 집계 및 분석 등 계량데이터를 통합 관리하는 시스템이다. 빅데이터 처리기술(오픈소스) 활용한 실시간 대용량 계량데이터 처리기술(Scale-Out)과 수집/VEE(Validation Estimation, Editing)/프로파일링/집계/저장관리 등 데이터 처리 전 과정을 분산•병렬 처리 기술을 사용한다. MDMS에서 사용하는 대용량 처리를 위한 미들웨어들은 Flume, Kafaka, Spark, MongoDB, Hadoop 등이 사용된다.

## B. NMS(Network Management System, 망 관리 시스템)

인터넷과 인트라넷의 활성으로 많은 업무가 네트워크를 통하여 이루어지고 있고 의사 전달을 위해 E-Mail이나 SNS 등 네트워크가 없으면 안 될 정도로 활용성이 높아지고 있다.

이런 네트워크 환경 속에서 NMS(망 관리 시스템)란 네트워크상의 장비들을 효율적이고 통합적인 관리를 할 수 있는 중앙 감시시스템이며, 네트워크를 구성하고 있는 장비들에 대한 상태 데이터, 장애 데이터, 구성 데이터, 통계 데이터 등을 실시간으로 전송 및 모니터링을 하고 장비 상태의 이상이 발생 시, 알람신호 전송으로 관리자는 신속한 조치를 취하며, 수집된 정보로 통계 자료 및 네트워크 망의 상태를 분석 한다[19].

네트워크 관리 시스템에서 사용하는 프로토콜인 SNMP, CMIP 그리고 RMON 등이 있다. 본 논문에서는 SNMP에 대해서 자세히 알아보고자 한다.

### 1. SNMP (Simple Network Management Protocol)

(1) 배경

관리 표준 프로토콜이 만들어지기 전에는 업체들이 자기들만의 고유의 관리 프로토콜을 만들어 사용하고 있었다. 그래서 각각의 업체들마다 중앙서버 명령에 대한 클라이언트 어플리케이션은 물론 네트워크 장비를 위한 관리 프로그램도 제공해야만 했다. 소수의 큰 회사들은 서로 협약을 맺고 망 관련 통신 장비 정보를 교환하여 각자 자신들의 망 관리 어플리케이션을 표준화된 네트워크 관리 플랫폼으로 만들려고 노력했다. 그러나 호환성 있는 에이전트들이 제한돼 있고, 망 관리 시스템 업체들의 고유 정보를 공유하는 장비 이외에는 관리하기가 힘들기 때문에 성공적이지 못했다.

이러한 상황에서 SNMP(Simple Network Management Protocol)는 1988년도 말에 선구자적인 역할을 했으며 현재 대부분의 모든 망 관리 장비들이 지원하고 있다. SNMP가 빠르게 표준이 된 이유는 기업의 네트워크 관리 측면에서 증명된 능력, 적당한 가격 및 성능, 그리고 또 다른 표준 프로토콜을 없었다는 점이다.

망 관리 프로토콜인 SNMP는 초기에 TCP/IP 망을 관리하기 위해 만들어졌다. SNMP 패킷은 모든 IP 라우터를 이용하며 라우터나 브릿지의 연결된 네트워크처럼 네트워킹 환경관리에 이상적이다.

(2) SNMP의 구성

SNMP는 응용 계층 프로토콜로 설계 되었으며, UDP(User Datagram Protocol) 위에서 동작하게 된다. SNMP가 네트워크를 관리하는 형태는 관리국(Management Station)과 관리 에이전트(Management Agent), MIB(Management Information Base), 네트웍 관리 프로토콜의 4가지 요소로 구성된다.[20]

관리국 즉 망 관리 시스템(Network Management System)은 시스템 관리자에게 네트워크 모든 상황을 한눈에 볼 수 있도록 주기적으로 망 관리 데이터를 수집, 분석하여 장애관리 등의 기능수행하기 위해 데이터베이스를 구축하고, 관리자에게 인터페이스를 제공하고 있다.

SNMP 에이전트는 라우터, 브릿지, 허브와 같은 네트워크 장비나 피관리대상장비에 설치되어 망 관리 시스템의 요청이 있을 때 망 관리 정보를 전송하거나 망 관리 시스템으로부터 설정 등 명령을 수행하며 장애 발생 시 자동적으로 장애 상황을 Trap으로 망 관리 시스템에 통보한다.

SNMP(Simple Network Management Protocol)은 매니저와 망 관리 에이전트간 통신하는 표준 통신 프로토콜이다.

MIB(Management Information Base)는 관리되어야 할 정보들을 객체라고 하며, 이 객체들을 모아놓은 집합체이다. 또한 관리되고 있는 객체들은 계층적 트리구조로 이루며 이 객체 식별자를 OID(Object Identifier)라고 부른다. MIB은 [표 2-6]처럼 5가지 기능분야로 분류된다.

[표 2-6] MIB의 5가지 기능[21]

| 기 능 | 설 명 |
|---|---|
| 구성관리 (Configuration Management) | 네트워크상의 장비와 전반적인 물리구조를 Mapping 하는 기능을 말한다. |
| 성능관리 (Performance Management) | 가용성, 응답시간, 사용량, 에러량, 처리속도 등 성능 분석에 필요한 통계 데이터를 제공하는 기능 |
| 고장관리 (Fault Management) | 문제의 검색, 추출 및 해결을 제공하는 기능 |
| 보안관리 (Security Management) | 정보의 제어 및 보호 기능 |
| 계정관리 (Accounting) | 각 노드별로 사용현황을 측정하는 기능 |

(3) SNMP의 동작

SNMP는 매니저와 에이전트 사이에 관리정보를 주고 받기위한 프로토콜이다. [표 2-7]에서 보듯이 SNMPv1인 경우 5가지(GetRequest, GetNextRequest, SetRequest, GetResponse, Trap)와 SNMPv2는 4가지(GetBulkRequest, InformRequest, Trapv2, Report)가 추가로 더 있다.

1) GetRequest : 매니저에서 지정한 OID 객체의 값 한 개를 요청한다.

2) GetNextRequest : GetRequest로 지정한 OID 객체 한 개의 값을 가져온 후에 그 다음의 값을 요청하여 가져올 때 사용한다.

3) GetResponse : 매니저로부터 지정한 OID 객체의 값을 요청한후 그 결과로 에이전트가 응답할 때 사용하는 타입이다.

4) SetRequest : 매니저로부터 지정한 특정 OID 객체의 값을 변경할 때 사용한다.

5) TrapRequest : 에이전트가 임계치나 알람 등이 발생했을 때 서버로 알릴 때 사용한다[22].

6) GetBulkRequest : GetNextRequest처럼 응답을 하나씩 받지 않고, 묶음 응답으로 받아서 네트워크 트래픽 절약에 효과적이며 빠르다. SNMP v2부터 추가된 메시지이며, 특별인자로 비반복과 최대반복 설정 값을 가지고 있다.

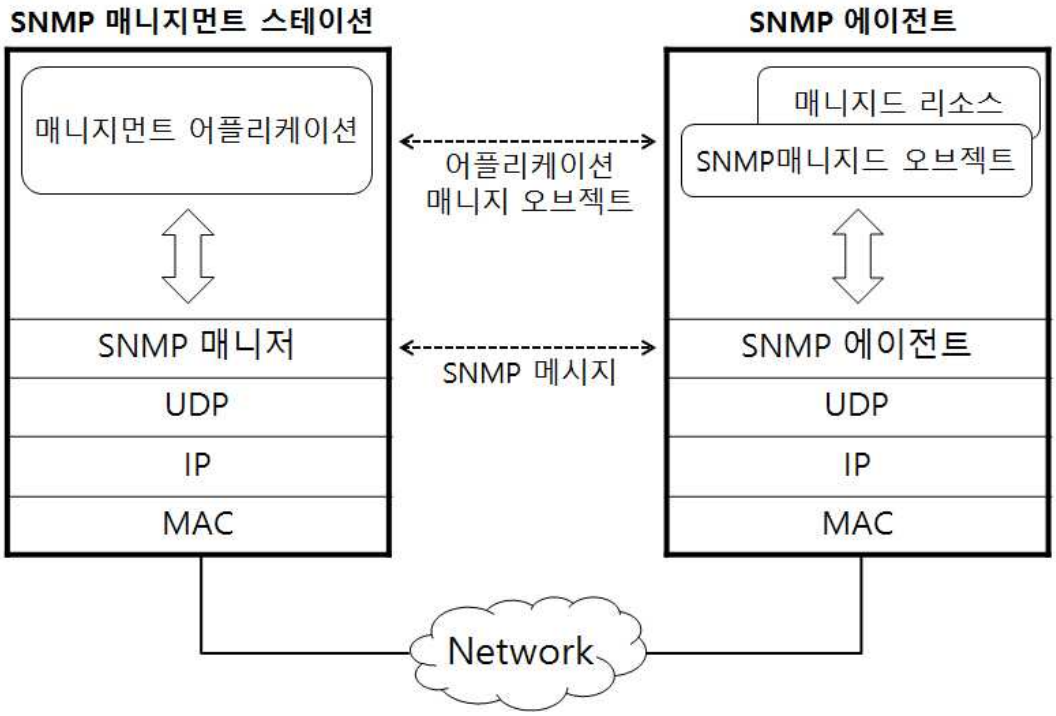
7) InformRequest : 미리 설정된 객체의 상태를 서버로 전송할 때 사용한다.

8) Trapv2 : 이벤트 데이터 전송 시 SNMP v1의 메시지를 사용하지 않고 SNMP v2의 메시지를 사용하여 이벤트 데이터를 전송한다.

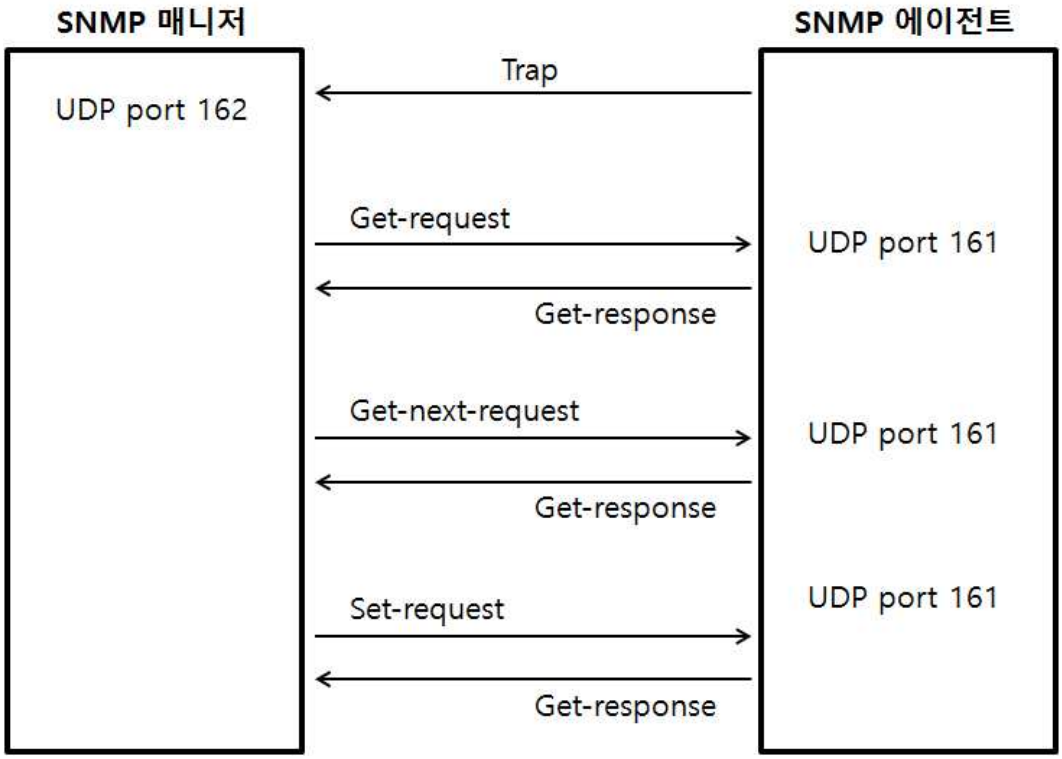
9) Report : 관리자들 간에 어떤 오류 유형들을 보고할 때 사용한다.

[그림 2-5]는 SNMP 프로토콜 아키텍처이다. 매니저와 에이전트간 통신하는 프로토콜 스택이다.

[그림 2-6] SNMP 매니저와 에이전트간의 통신 흐름을 나타낸다. 에이전트의 이벤트 발생시 Trap으로 장애 상황을 매니저에게 알린다. 또한 매니저에서 망 관리 정보 요청 시에는 Get-request를 요청하고 에이전트는 Get-response로 응답한다. 그리고 다음 객체(OID)를 요청 시에는 Get-next-request로 요청하고 에이전트는 Get-response로 응답한다. 그리고 에이전트의 정보를 변경 시에는 Set-request 명령을 전송하고 그 응답으로는 Get-response로 처리한다.



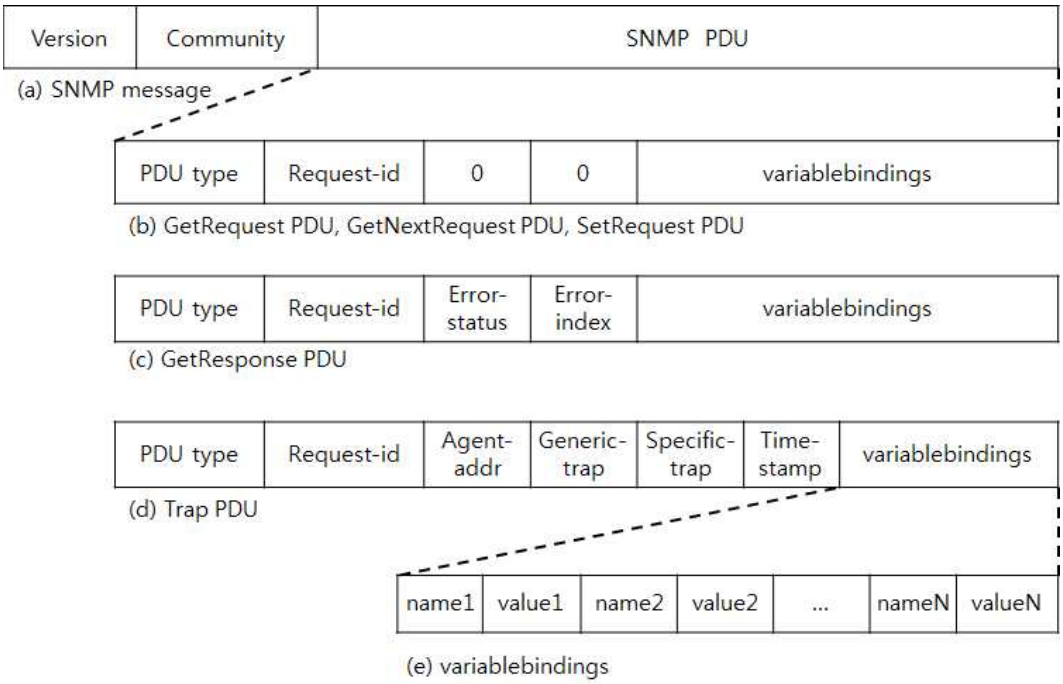

[그림 2-4] SNMP 프로토콜 아키텍처




[그림 2-5] SNMP 매니저와 에이전트간의 통신

아래 그림은 SNMPv1 프로토콜 구조이며, 첫 번째 필드는 SNMP의 버전, 두 번째는 community 필드로 에이전트를 접근하기 위해서 매니저에서는 community를 맞도록 전송해야 에이전트가 동작한다.

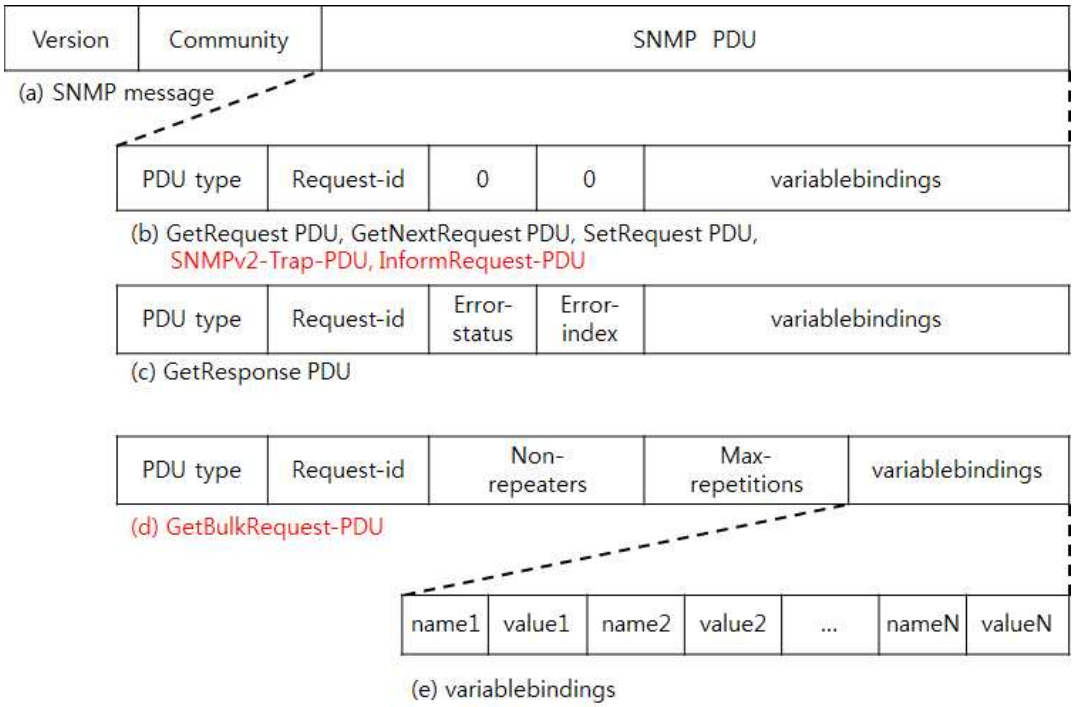
다음 SNMP PDU 타입에 따라 3 종류가 존재한다. (b)의 PDU 타입은 GetRequest, GetNextRequest, SetRequest 등 3가지 이고, (c)의 PDU 타입은 GetResponse 이며, (d)의 PDU 타입은 Trap 데이터를 전송할 때 사용한다. PDU 타입별로 마지막에 variablebindings 필드가 존재한다.




[그림 2-6] SNMP v1 PDU Formats

다음 그림은 SNMPv2 프로토콜 구조이며, 첫 번째 필드는 SNMP의 버전, 두 번째는 community 필드로 에이전트를 접근하기 위해서 매니저에서는 community를 맞도록 전송해야 에이전트가 동작한다.

다음 SNMP PDU 타입에 따라 3 종류가 존재한다. (b)의 PDU 타입은 GetRequest, GetNextRequest, SetRequest, SNMPv2-Trap, InformRequest 등 5가지 이고, (c)의 PDU 타입은 GetResponse, (d)의 PDU 타입은 GetBulkRequest 데이터를 요청할 때 사용한다. PDU 타입별로 마지막에 variablebindings 필드가 존재한다.




[그림 2-7] SNMP v2 PDU Formats

[표 2-7] PDU Type 종류

| Version1 | Version2 | PDU Type value | PDU Type |
|---|---|---|---|
| ○ | ○ | 0 | GetRequest-PDU |
| ○ | ○ | 1 | GetNextRequest-PDU |
| ○ | ○ | 2 | Response-PDU |
| ○ | ○ | 3 | SetRequest-PDU |
| ○ | | 4 | Trap-PDU in SNMPv1 |
| | ○ | 5 | GetBulkRequest-PDU |
| | ○ | 6 | InformRequest-PDU |
| | ○ | 7 | Trapv2-PDU |
| | ○ | 8 | Report-PDU |

(4) MIB 구조

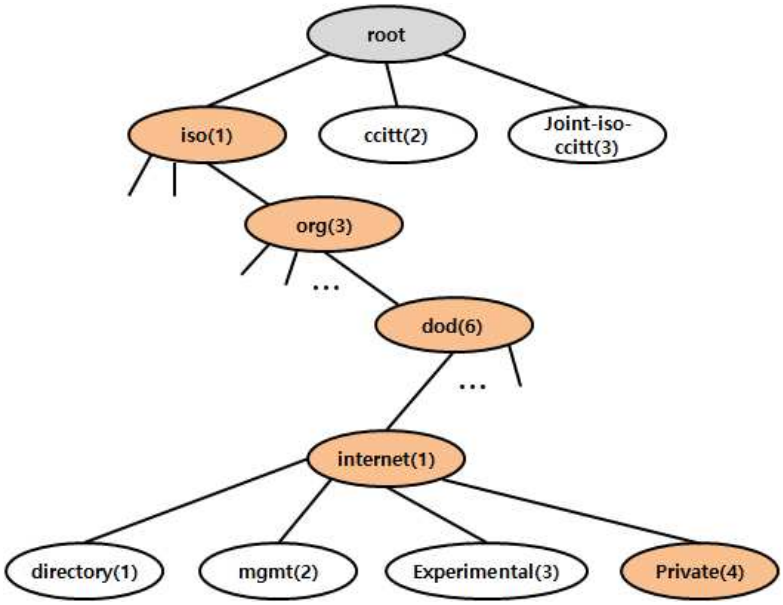
SNMP 각 오브젝트는 계층적인 구조 또는 트리 구조로 나열되어 있다. MIB객체들은 ASN.1문법에 따라 정의하며 이름과 표기 문법, 전송을 위한 인코딩 규칙을 갖는다. ASN.1은 부호화 규칙에 상관없이 데이터를 표현하기 위해 일반적인 데이터 구조를 가진 단순한 형태의 구문을 가진 표기법이며, ASN.1을 이용하여 응용프로그램 계층 간에 정보를 교환하고, 사용자로 하여금 그 내용을 읽을 수 있도록 하며, 다양한 부호화 규칙들을 제공함으로서 특정 데이터를 변환하여 상대방 시스템에 전송할 수 있도록 도와주는 일종의 추상구문을 포함하는 표기법이다[23].

각 오브젝트는 계측적인 OBJECT IDENTIFIER로서 식별하고, 정수의 연속된 나열로 표시한다.

각 오브젝트는 root에서 시작해서, 첫 번째 레벨에는 [그림 2-9]처럼 iso, ccitt, joint-iso-ccitt의 3가지 노드가 있다.

iso, ccitt, joint-iso-ccitt의 iso 노드에서, 하나의 서브트리는 조직집단(org)을 위해서 사용된다. 그리고 인터넷 노드는 1.3.6.1이라는 오브젝트 ID를 가지고 있다. 이것은 트리구조의 experimental 노드 앞에 붙은 타이틀이다. 아래와 같이 internet 노드 밑에는 4개의 노드를 정의된다.

- directory : 이 서브트리는 나중의 OSI directory(X.500)를 위해 예약된다[24].
- mgmt : 이 서브트리는 IAB에서 정의한 오브젝트들 용으로 사용된다[24].
- experimental : 이 서브트리는 Internet에서 실험적인 오브젝트를 나타낸다[24].
- private : 이 서브트리는 주로 특정 벤더/개인의 오브젝트를 나타내기 위해 사용된다[24].




[그림 2-8] MIB 구조

## 2. 네트워크 관리 구성

(1) 구성 요소

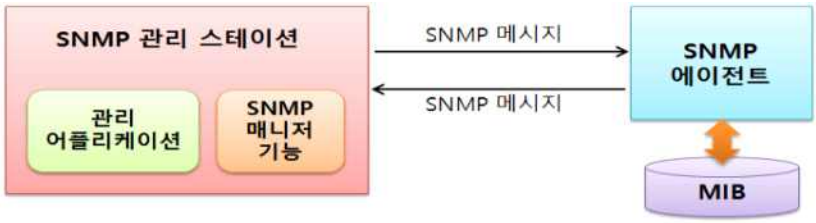
1) SNMP 매니저 : SNMP 매니저는 get, getnext, set, getbulk 등의 SNMP 명령 메시지를 에이전트로 전달하고 수신 받은 결과를 분석하여 사용자에게 제공한다.

2) SNMP 에이전트 : SNMP 매니저로부터 전달받은 get, getnext, set, getbulk 등의 SNMP 명령 메시지를 수신하여 처리하고 그 결과를 SNMP response 메시지를 통해서 전달하는 기능을 수행한다.

3) SNMP 관리 스테이션 : SNMP 매니저와 동일하지만, 실제적인 관리 어플리케이션이 탑재되어 운영된다는 점이 SNMP 매니저와는 다른 점이다.

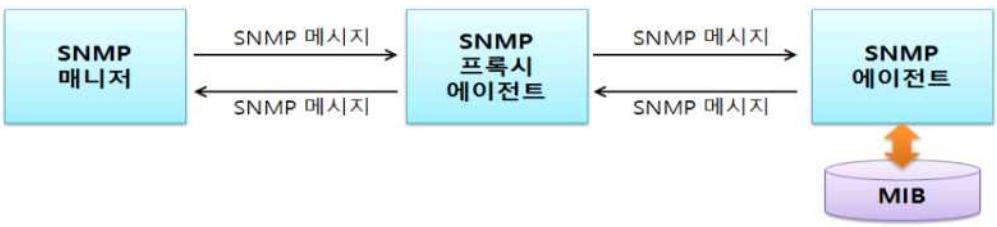
(2) 관리 구성 종류

1) SNMP 기본 구성 : SNMP 관리 스테이션과 SNMP 에이전트로 구성된 기본적인 구성이다. SNMP 관리 스테이션은 SNMP 에이전트로부터 망 관리 정보들을 수집 관리하며, SNMP 에이전트는 장비에 설치되어 망 관리 정보들을 생성하고 MIB에 저장 관리한다.



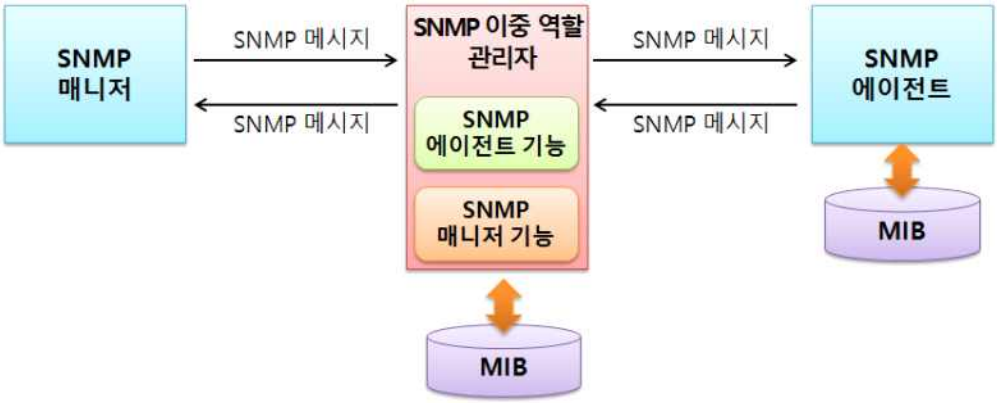

[그림 2-9] SNMP 네트워크 관리 기본 구성

2) SNMP 프록시 에이전트 : SNMP 에이전트에 대한 프록시 기능을 수행한다. 즉, SNMP 매니저로부터 오는 요청을 에이전트로 전달하고, SNMP 에이전트로부터 돌아오는 response를 매니저에게 중간에서 전달하는 역할을 수행한다.



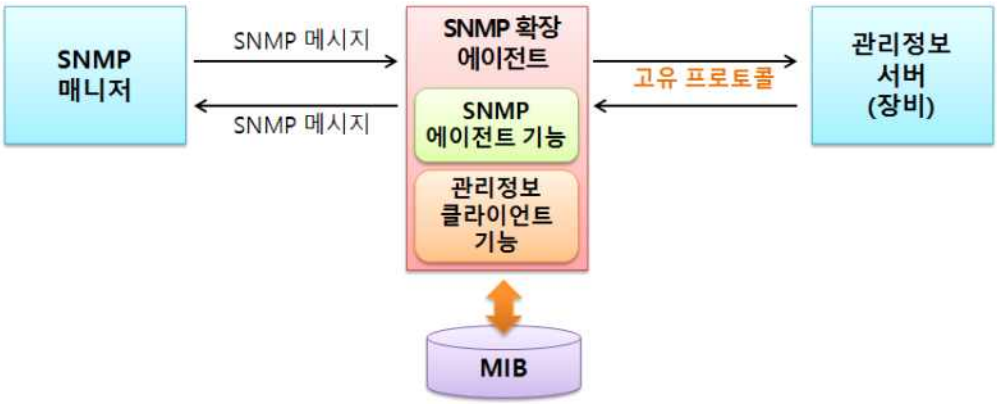

[그림 2-10] SNMP 프록시 에이전트가 포함된 구성

3) SNMP 이중 역할 매니저 : SNMP 매니저 기능과 SNMP 에이전트 기능을 모두 수행한다.




[그림 2-11] SNMP 이중 역할 매니저가 포함된 구성

4) SNMP 확장 구성 : SNMP는 IP계층 이상에서 동작하기 때문에 IP계층이 없는 장비들의 관리하기 위해 사용된다. 확장 에이전트의 클라이언트는 벤더 고유의 프로토콜을 사용하여 장비들의 관리정보를 수집하여 MIB에 저장 관리하며, 매니저로부터 SNMP메시지 응답에 대응하기 위한 에이전트 기능을 동시에 수행한다.




[그림 2-12] SNMP 네트워크 관리 확장 구성

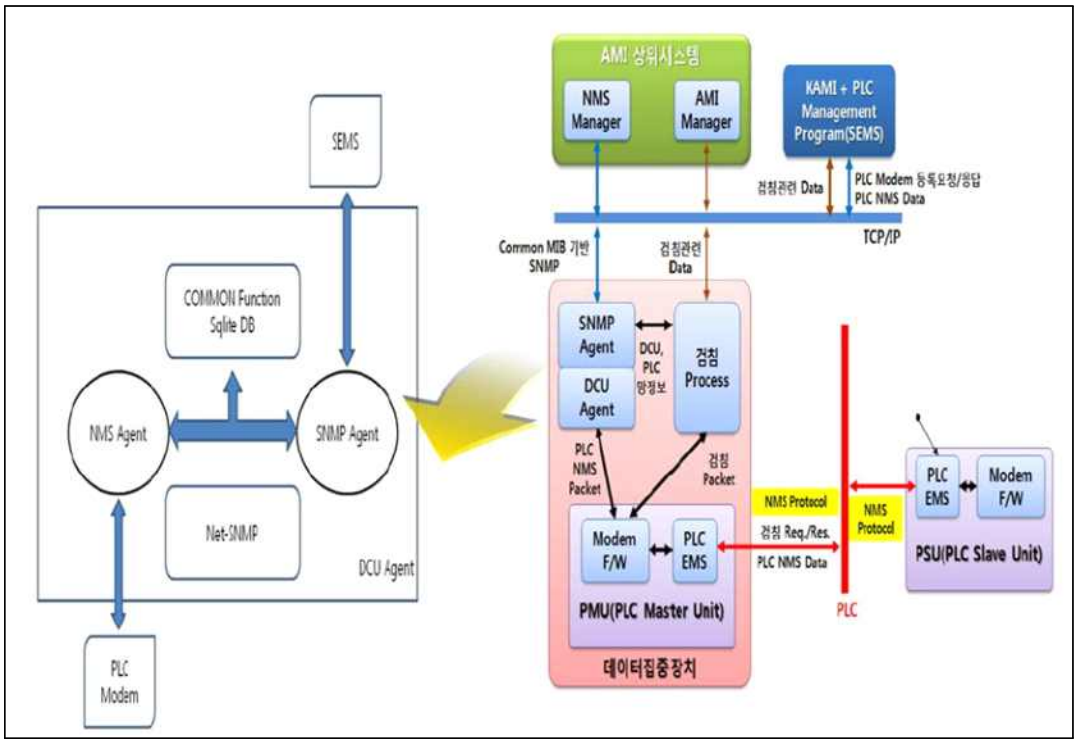
## C. 저압원격검침 망 관리 에이전트에 관한 연구

저압원격검침 망 관리 에이전트는 신규 모뎀의 등록, 삭제 명령, 모뎀 리셋 명령 등을 수행하고 원격검침을 위한 PLC 통신이 가능하도록 하기 위해서 최적의 라우팅 경로를 결정하고 통신을 가능하도록 연결해 주는 역할을 수행한다. 또한 DCU 및 모뎀 정보 등 고객의 PLC 모뎀 상태에 대한 상세한 정보를 주기적으로 수집하는 기능을 수행하고 안정적인 검침을 수행할 수 있도록 관리한다.

### 1. 저압원격검침 망 관리 에이전트 구조

(1) 블록 다이어그램

다음의 그림은 저압AMI시스템에서 사용되고 있는 DCU(內) 망 관리 에이전트의 블록 다이어그램이다.




[그림 2-13] AMI시스템 망 관리 에이전트 블록 다이어그램[18]

(2) 망 관리 에이전트 프로세스

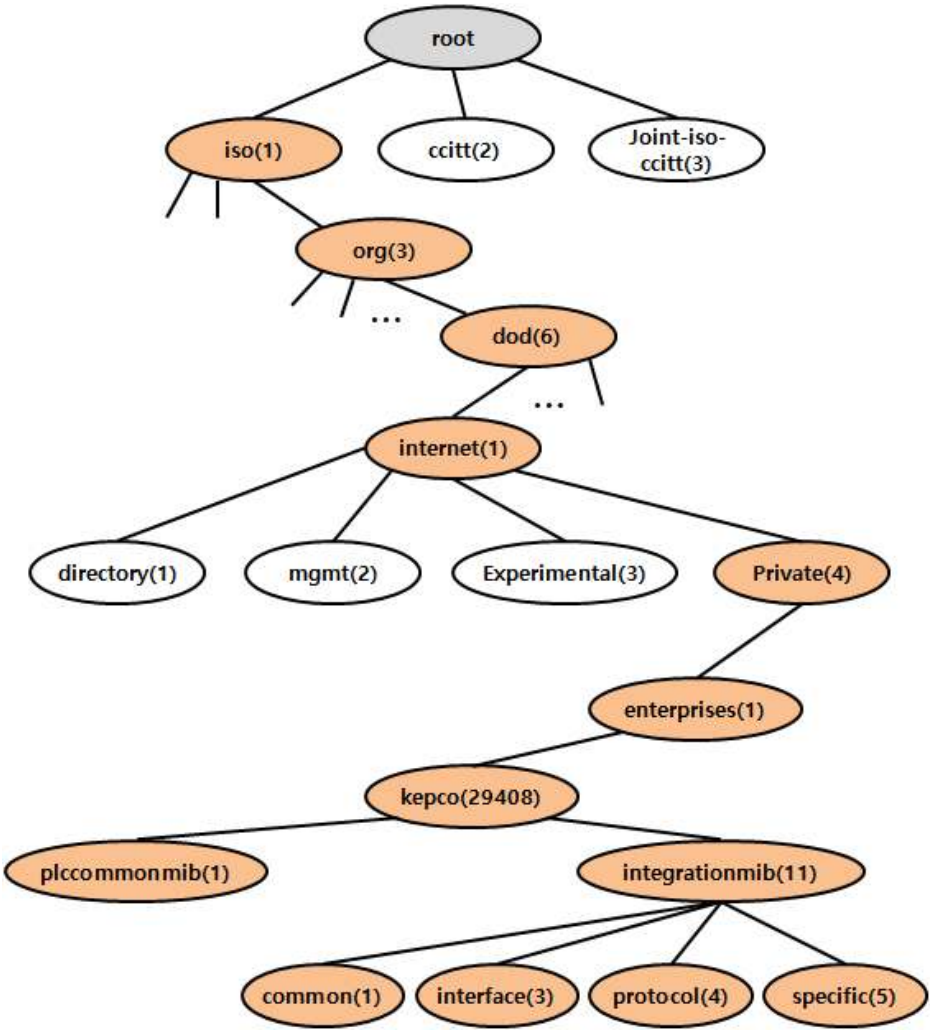
아래 그림은 DCU(內)에 탑재돼서 돌아가는 저압원격검침 망 관리 에이전트 프로세스가 동작중인 모습이다.


[그림 2-14] AMI시스템 망 관리 에이전트 프로세스[18]

## 2. 저압원격검침 망 관리 MIB 구조

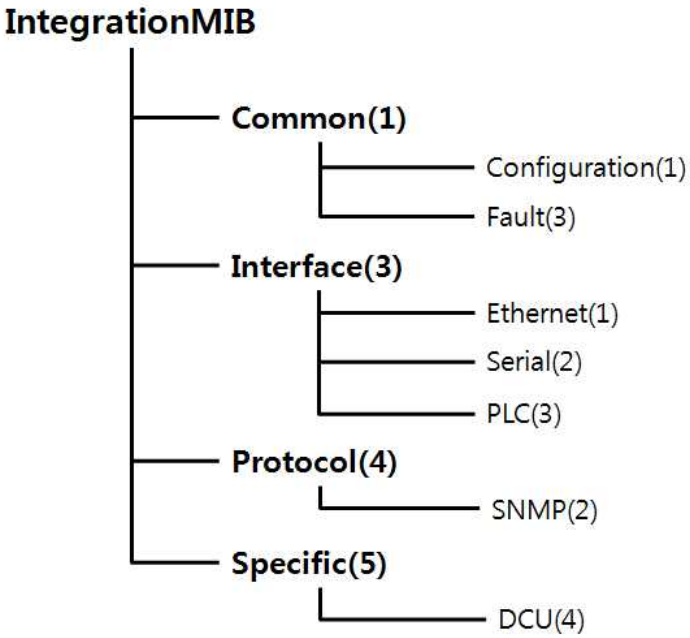
(1) MIB 구조

저압원격검침 망 관리 에이전트에서 사용하는 MIB구조는 [그림 2-15]과 같이 Private 영역에 kepco 오브젝트를 하나 만들고 그 하위에 각종 정보들을 고유하게 만들어서 사용하고 있다.




[그림 2-15] AMI시스템 망 관리 MIB 구조

아래 그림은 IntegrationMIB OID(1.3.6.1.4.1.29408.11) 하위 구조를 상세하게 표현하였다. Common, Interface, Protocol, Specific 등 크게 4가지로 구분했으며, Interface MIB 하위에 Ethernet, Serical, PLC 등 총 3가지로 구분하여 PLC모뎀 정보를 이 PLC 하위에 저장 관리하도록 하고 있다. 그런데 이 구조는 무선 등 각종 다른 통신방식의 모뎀 정보를 저장 관리할 수 없는 구조이다. 유무선 혼합 통신방식을 적용하기 위해서는 여러 가지 통신방식의 모뎀 정보들을 저장 할 수 있도록 MIB 설계를 하여야 할 것이다.

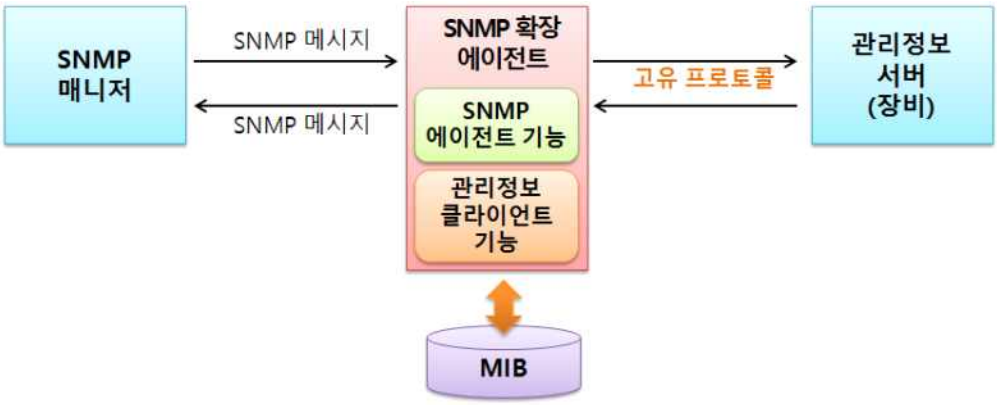



[그림 2-16] AMI시스템 망 관리 IntegratonMIB 구조

## 3. 저압원격검침 망 관리 네트워크 구성 방식

저압원격검침시스템은 SNMP 네트워크 관리 구성 방식을 [그림 2-17]처럼 SNMP 확장 구성 방식을 사용하고 있으며, 관리정보서버인 모뎀의 경우 SNMP메시지를 주고 받을 수 있는 UDP 통신을 하지 않고, 고속 PLC칩을 사용한 MAC 통신방식을 사용하기 때문에 SNMP 표준 프로토콜을 사용할 수가 없다.

유무선 혼합 통신방식을 적용하기 위해서 각종 통신방식의 모뎀들을 하위에 설치해야 하는데 그 경우 각 모뎀별 고유의 프로토콜을 사용할 수 밖에 없고, 그러면 에이전트 또한 각 통신방식에 맞는 전용 에이전트가 필요하게 될 것이다. 이러한 여러 가지 망 관리 에이전트들의 정보를 취득하기 위해서 이중역할 망 관리 에이전트가 필요하게 될 것이다.




[그림 2-17] AMI시스템 SNMP 망 관리 확장 구성

# Ⅲ. 이중 역할 망 관리 에이전트 개발

본 논문에서는 유무선 혼합 AMI시스템의 DCU(內)에 탑재되는 이중 역할 망 관리 에이전트를 JAVA기반의 snmp4j api를 이용하여 개발하고자 한다.

## A. 시스템 설계

### 1. 시스템 개요

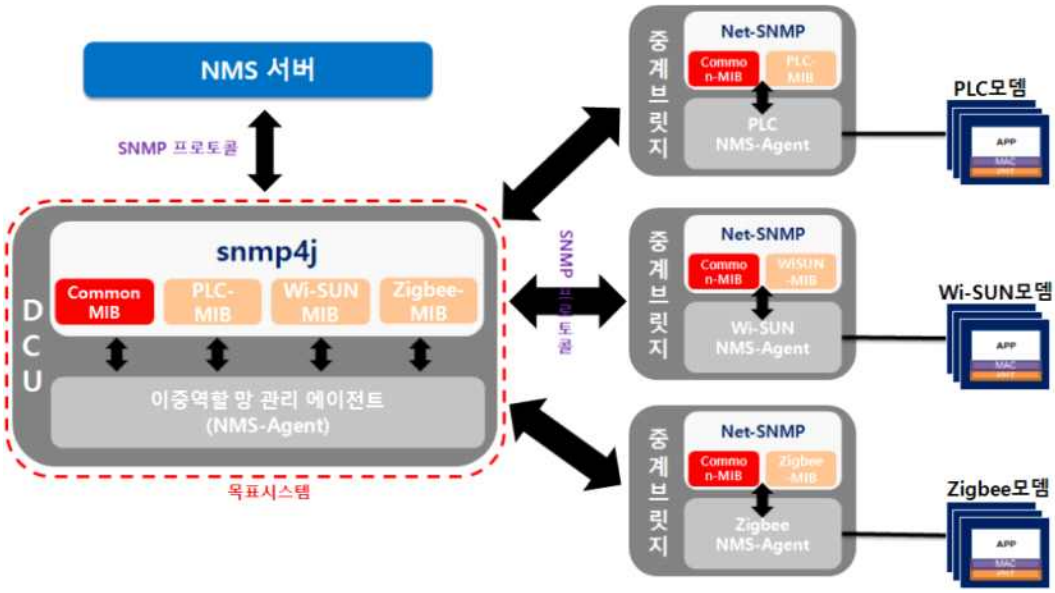
서론에서 언급했듯이 AMI를 구축하기 위해서는 다른 통신방식에 비해 PLC가 절대적으로 유리하다. 그러나 PLC 통신방식만을 이용해서 구축하기에는 한계가 있어 무선 등 다양한 통신방식을 병행해서 구축할 필요가 있다. 그런데 PLC 통신방식만을 이용해서 구축하는 기존 방식에서의 망 관리 에이전트는 한 가지 역할만 하면 되는 단순한 방식 이었다. [그림 3-1]처럼 DCU 하위에 여러 가지 통신방식이 이중으로 연결 될 경우 기존 방식의 망 관리 에이전트는 또 다른 에이전트들로부터 망 관리 데이터를 수집하는 매니저 기능이 존재하지 않고 직접 모뎀을 컨트롤 하도록 돼 있어서 망 관리 기능이 불가능하다. 그래서 본 에이전트는 DCU 아래에 있는 중계브릿지들로부터 망 관리 정보를 수집하는 매니저 기능과 서버의 요청에 응답하는 에이전트 기능을 함으로써 망 관리 서버에서 말단의 통신 모뎀 정보까지 수집이 가능하며, 이렇게 설계하므로써 말단의 수 많은 에이전트들을 제어하며 중간에서 전단처리 기능을 하여 망 관리 서버의 부담을 줄여주고 서버의 가용성을 높일 수 있도록 설계하고자 한다.

[그림 3-1]은 NMS서버부터 최종 말단 모뎀까지의 전체시스템 구성도이며, 본 구성도는 NMS서버, DCU, 중계브릿지, 모뎀 등 4개의 영역으로 나뉜다. 모뎀 하위에는 전자식 전력량계가 485통신 케이블을 통해서 연결돼 있으나 본 논문에서는 모뎀까지가 네트워크 관리 대상이므로 전자식 전력량계는 구성도에서 제외시켰다.

첫 번째 중계브릿지의 경우 PLC용 망 관리 에이전트가 탑재돼 있으며, 그 하위에 연결된 여러 대의 PLC모뎀들을 등록하고 최적의 통신경로를 결정하여 통신을 가능하도록 해주며, 중계브릿지에 대한 기본정보인 Common-MIB과 PLC용 MIB만

을 보유하고 망 관리 에이전트는 중계브릿지 및 하위 모뎀들의 정보들을 스캔하여 해당 MIB에 저장 관리하는 역할을 한다. 두 번째 중계브릿지의 경우 Wi-SUN모뎀에 대한 전용 망 관리 에이전트가 탑재 되여 Wi-SUN모뎀들의 관한 정보들을 스캔하고 저장 관리한다. 세 번째 중계브릿지의 경우 Zigbee모뎀에 대한 전용 망 관리 에이전트가 탑재 되여 Zigbee모뎀들의 관한 정보들을 스캔하고 저장 관리한다.

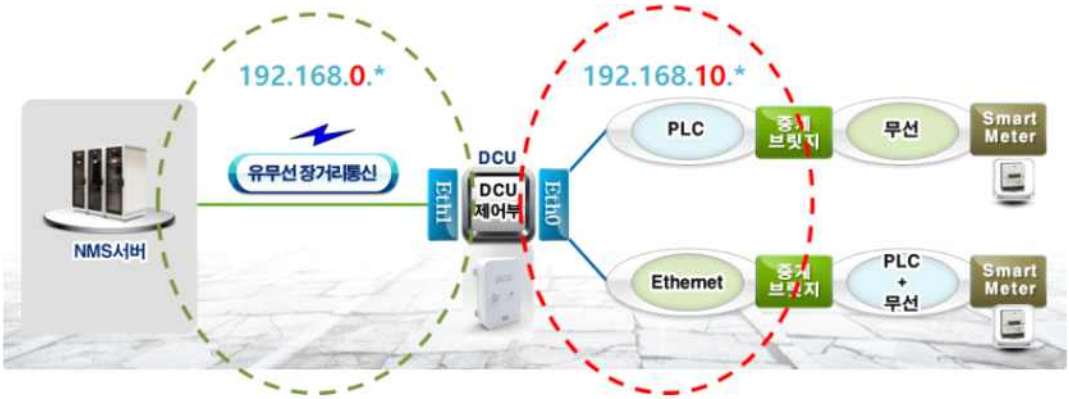
목표시스템인 DCU(內)의 탑재돼 있는 이중 역할 망 관리 에이전트는 DCU에 대한 기본정보인 Common-MIB과 중계브릿지 하위의 모뎀들의 종류들에 따라 PLC-MIB, Wi-SUN MIB, Zigbee-MIB을 정의하고 중계브릿지들로부터 주기적으로 MIB정보를 수집하여 저장관리한다. 또한 NMS서버로 부터의 망 관리 정보 요청에 응답하는 이중 역할 매니저 역할을 하는 에이전트로 구성된다.




[그림 3-1] 시스템 구성도

[그림 3-2]는 전체적인 시스템의 네트워크 구성도를 나타낸다. DCU의 Eth1 인터페이스에 IP주소를 “192.168.0.100”으로 설정하여 NMS서버와 DCU사이에 네트워크 주소를 192.168.0.* 으로 C class에서 설정하였다. 그리고 Eth0 인터페이스에는 IP 주소를 “192.168.10.100”으로 설정하여 DCU와 중계브릿지 사이의 네트워크 주소를 192.168.10.* 인 C class에서 설정하였다. DCU를 기준으로 상위와 하위 네트워크

주소를 다르게 설정하였다. 그래서 서버에서는 중계브릿지까지 직접적인 접속을 할 수가 없다. 물론 DCU에 라우팅 기능을 설정하여 직접 연결이 가능하도록 할 수도 있지만 보안적인 측면에서도 네트워크를 분리하는 것이 좋고, 본 논문에서도 네트워크를 다르게 설정하는 것으로 가정하였다.




[그림 3-2] 네트워크 구성도

망 관리 시스템 서버와 DCU(內)에 탑재되는 이중 역할 망 관리 에이전트간 통신은 망 관리 표준 프로토콜인 SNMP를 사용하고, 또한 DCU와 중계브릿지간의 통신도 SNMP 프로토콜을 사용한다. 중계브릿지와 PLC모뎀간은 한전 PLC규격을 사용 하는 등 중계브릿지와 각종 하위 이기종 통신모뎀들은 각 모뎀 제작사에서 제공하는 프로토콜을 사용하며 해당 모뎀 제작사에서 에이전트를 제공 탑재한다. 단, MIB정보는 서로 충돌되지 않도록 Interface 테이블에 OID(Object IDentifier, 객체ID)를 다르게 하여 구성하도록 한다.

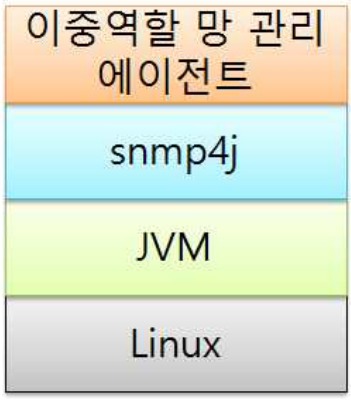
## 2. 어플리케이션 구성

본 절에서는 개발하는 목표시스템의 이중 역할 망 관리 에이전트의 구성을 설명한다.

(1) 어플리케이션 구성도

이중 역할 망 관리 에이전트의 동작하는 관계도는 [그림3-3]에서 보듯이 Linux

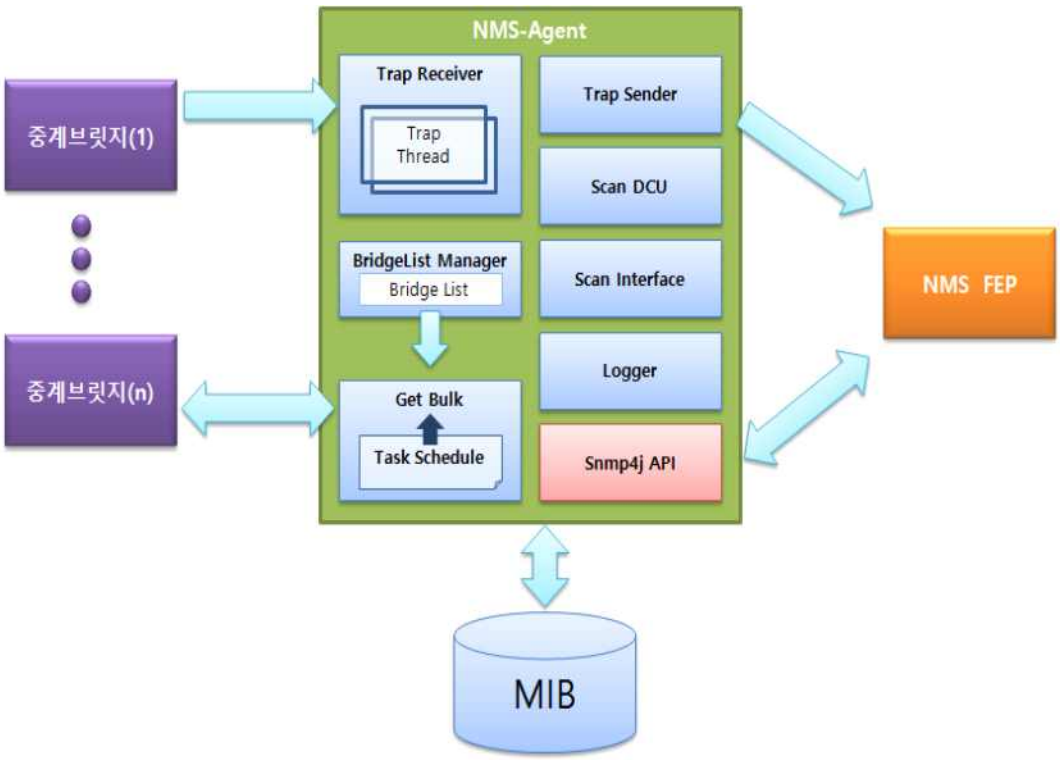
O/S 위에 JVM(Java Virtual Machine)을 탑재하였으며, JVM기반으로 본 S/W 즉 이중 역할 망 관리 에이전트가 동작하는 방식인데, snmp4j의 JAVA기반의 API를 사용한다.




[그림 3-3] 어플리케이션 구성도

(2) S/W 아키텍처

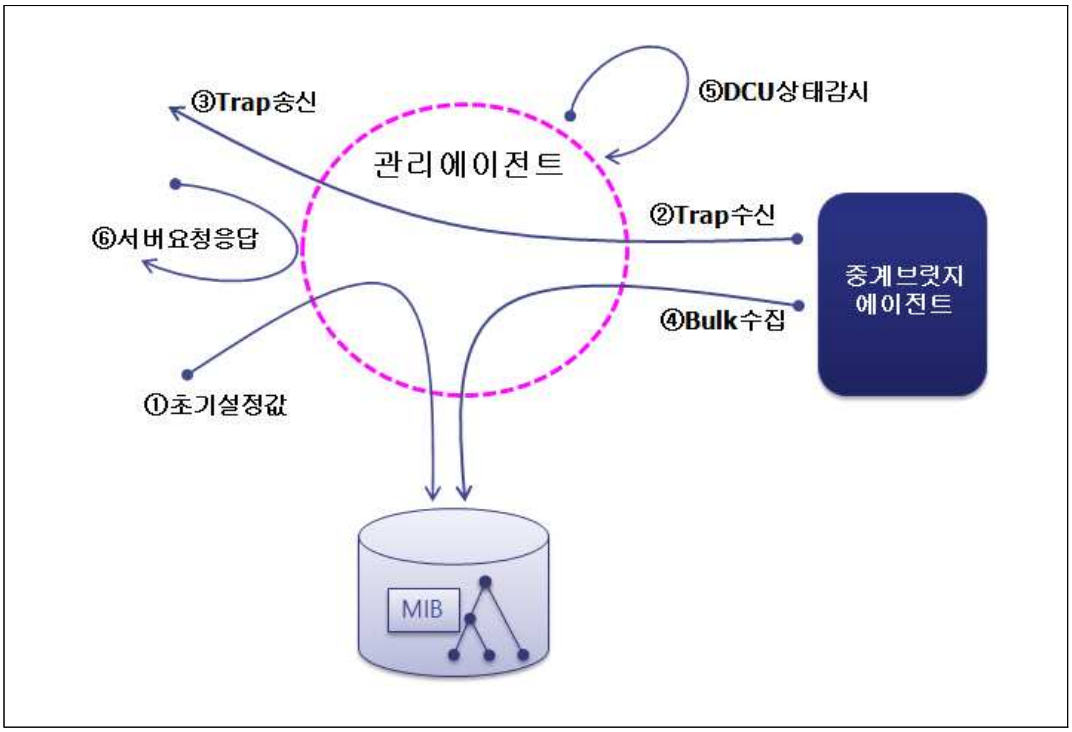
이중 역할 망 관리 에이전트의 S/W 아키텍처의 구성도는 [그림3-4]와 같다. 중계브릿지의 망 관리 에이전트들로부터 Trap 데이터를 수신 받아 처리하는 Trap Receiver, 이중 역할 망 관리 에이전트의 전원 부팅 시나 이벤트 상태를 서버로 알려주는 Trap Sender, DCU의 CPU 부하율이나, 메모리 사용률 등을 체크하는 Scan DCU, DCU의 각종 인터페이스 상태를 체크하는 Scan Interface, 어플리케이션의 로그를 기록하는 Logger, SNMP java 라이브러리인 Snmp4j API, 중계브릿지의 접속정보를 관리하는 BridgeList Manager, 중계브릿지들로부터 주기적으로 MIB정보를 읽어서 저장하는 Get Bulk로 구성 돼 있다.




[그림 3-4] S/W 아키텍처

(2) 기능

이중 역할 망 관리 에이전트는 Configuration 파일로부터 DCU의 기본정보를 MIB에 설정하고, 중계브릿지로부터 주기적으로 MIB을 수집하여 저장하는 역할을 수행하고, 서버로부터의 요청에 응답하는 역할을 수행 할 뿐만 아니라 각종 이벤트 발생 시 서버로 Trap정보를 전송하는 역할을 수행한다. 본 에이전트의 기능은 [그림3-5]와 같다.



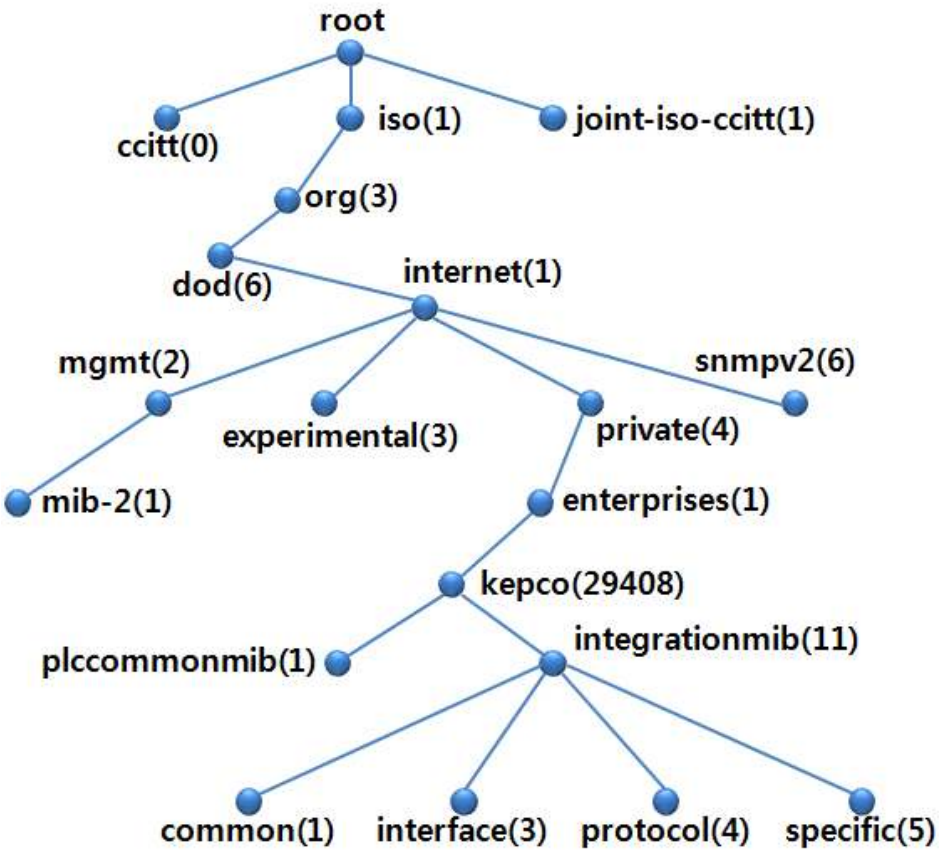

[그림 3-5] 이중 역할 망 관리 에이전트 기능도

이중 역할 망 관리 에이전트는 신규 중계브릿지의 Trap 데이터를 수신하여 MIB 및 중계브릿지 정보에 저장 관리한다. 본 에이전트가 재 기동 시에는 MIB정보는 초기화되므로 중계브릿지에 대한 정보는 중계브릿지 정보 파일을 참조하여 DCU하위 중계브릿지에 대한 정보를 계속 유지하는 역할을 수행한다. 또한 중계브릿지 정보 파일을 참조하여 중계브릿지 정보 리스트에 있는 모든 중계브릿지에서 망 관리 데이터를 수집하여 저장하는 역할을 수행한다.

## 3. MIB 정의

(1) MIB Model and Structure

아래 그림은 본 에이전트에서 정의한 MIB 모델과 구조를 표현했다. 인터넷 영역 하위 private 영역에 kepco용 mib을 정의해서 기 사용중에 있다. 여기에 유무선 혼합 MIB을 정의하기 위해서 interface 부분에 각종 인터페이스를 추가 정의 하였다.

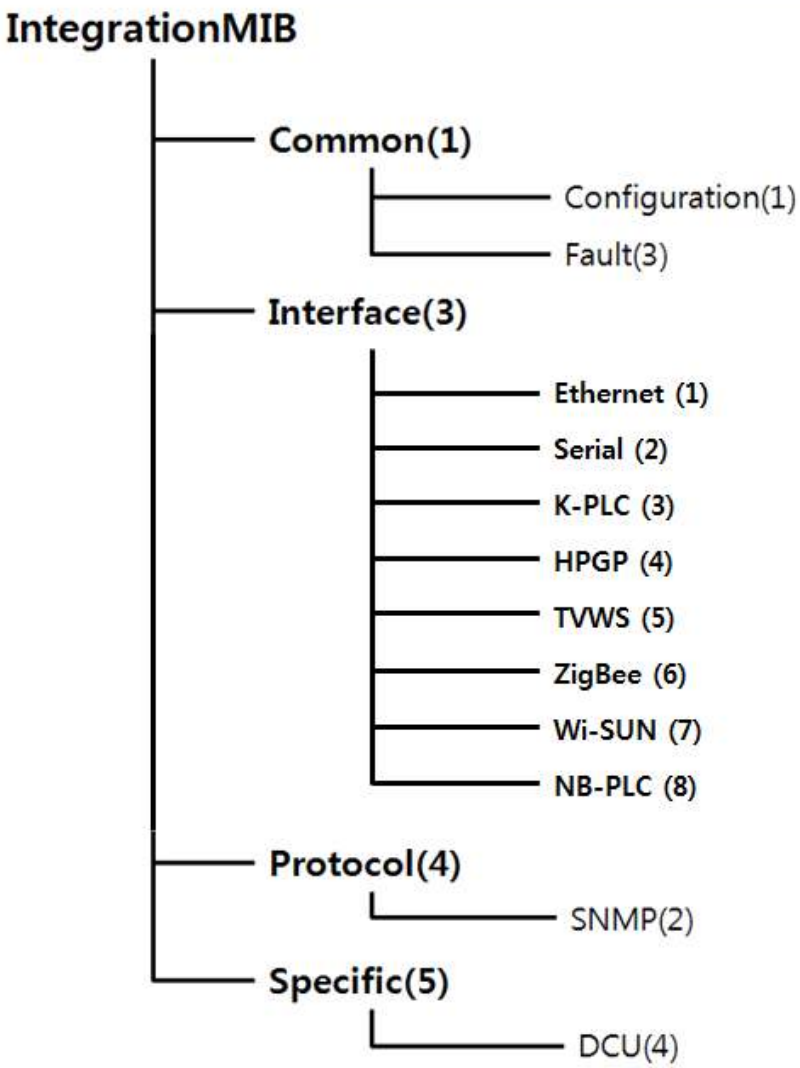



[그림 3-6] OID의 Hierarchy Tree[25]

- KEPCO Managed Object : 한국전력공사에서 사용되는 통신 단말장치 관리 객체[25]
- Common Information : 통신 단말장치 관리의 공통 기능으로 납품되는 모든 통신 단말장치들이 만족해야 할 필수항목으로 구성된다[25].
- Interface Information : 대상이 되는 통신단말장치가 제공하는 인터페이스 정보(Ethernet, Serial, PLC 등)를 정의한다[25].
- Protocol Information : 대상이 되는 통신단말장치가 제공하는 프로토콜(SNMP 등) 정보를 정의한다[25].
- Specific Information : 제조사 통신단말장치마다 제공되는 특성 정보를 정의한다[25].

(2) Object Identify Structure

kepco용 mib 하위에 통합 MIB인 IntegrationMIB를 [그림 3-7]처럼 정의 하였다. 아래 mib은 기 운영중인 고속PLC모뎀과 호환성을 유지하기 위해서 Common, Protocol, Specific 영역과 Interface 부분에 Ethernet, Serial, K-PLC(고속PLC) 등을 변경 없이 사용하고, HPGP, TVWS, ZigBee, Wi-SUN, NB-PLC(저속PLC)를 추가했다.



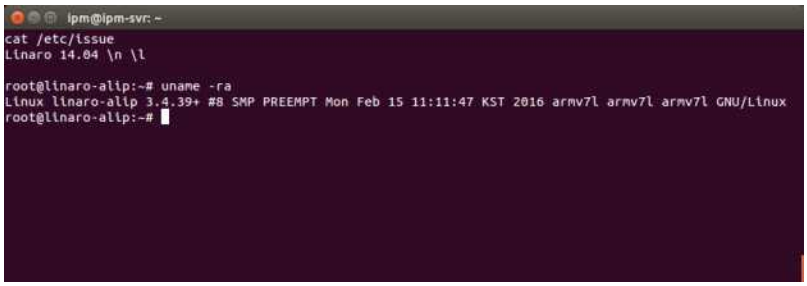

[그림 3-7] OID의 Structure

## 4. 개발 환경

본 절에서는 이중 역할 망 관리 에이전트를 개발하고 테스트하는 H/W 및 S/W 개발 환경에 대해서 설명한다.

(1) H/W 환경

[표 3-1] H/W 환경

| | |
|---|---|
| H/W 플랫폼 | ARM Corex A9 |
| RAM | 1GB RAM |
| Flash | 4GB Flash |
| O/S | Linux 3.4.39 |
| |  |

(2) S/W 환경

[표 3-2] S/W 환경

| | |
|---|---|
| 운영체제 | 리눅스 3.4.39 |
| JDK version | jdk-8u111-linux-arm32 |
| 통합 개발 환경(IDE) | Eclipse Java EE IDE<br>Version: Mars.2 Release (4.5.2) |
| snmp4j version | 2.5.3 |

## B. 시스템 구현

본 절에는 이중 역할 망 관리 에이전트의 구현과정을 상세히 설명한다.

### 1. 서비스 메인 구현

[그림 3-7]은 이중 역할 망 관리 에이전트를 기동 시에 처음 서비스를 시작하는 순서도를 나태내고 있다. 가장 먼저 DCU의 환경 설정 내용이 들어 있는 파일로부터 각종 configuration 내용들을 읽어 들여서 유효성 검사를 한 뒤에 변수에 할당하는 작업을 한다. 그리고 configuration정보에 있는 IP와 Port를 인자로 하여 Agent class를 생성하고, 즉 에이전트 서비스를 기동한다.

다음으로 [표 3-3]처럼 사용자 정의 MIB정보 class를 추가로 생성한다. 여기에 본 에이전트에서 구현하고자 하는 내용의 MIB정보들을 정의해서 사용하게 된다. 또한 중계브릿지들로부터 Trap을 받아서 처리할 수 있도록 Trap수신 서비스를 기동한다. 본 에이전트 기동시 DCU의 부팅을 서버에 알리도록 Trap을 서버로 전송한다. 그리고 주기적으로 DCU의 CPU 부하율이나 메모리 사용률 및 DCU(內) 인터페이스 정보를 스캔하고 사용자 정의 MIB에 저장하는 서비스를 기동한다. 마지막으로 스케줄에 따라서 중계브릿지들로부터 모뎀들에 대한 정보를 취득하고 사용자 정의 MIB에 저장하는 서비스를 기동한다.

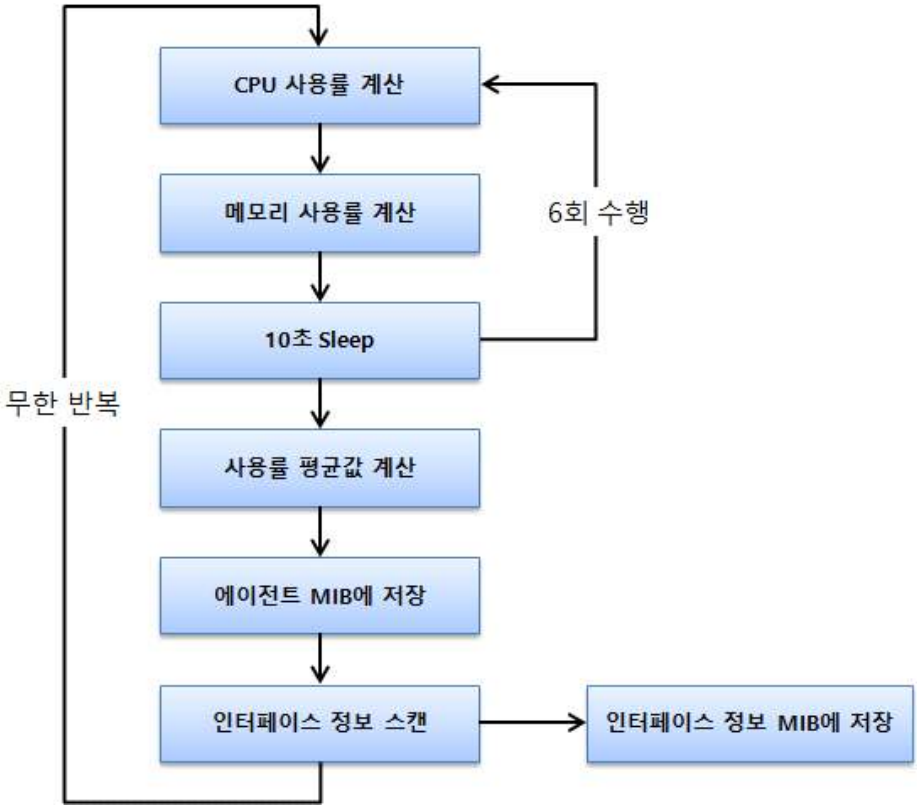

[그림 3-8] 서비스 메인 흐름도

[표 3-3] 사용자 정의 MIB 정보 class

| Class명 | 설명 |
|---|---|
| Mib_11_1_1_DcuInfo | DCU의 기본정보 |
| Mib_11_3_InterfaceInfo | DCU의 부착돼있는 인터페이스에 대한 정보 |
| Mib_11_4_ProtocolInfo | DCU와 서버간의 프로토콜 정보 |
| Mib_11_5_4_1_DcuConf | DCU의 설정정보 |
| Mib_11_3_3_PlcInterface | 중계브릿지 하위의 PLC모뎀들의 대한 정보 |
| Mib_11_3_7_WiSunInterface | 중계브릿지 하위의 WiSUN모뎀들의 대한 정보 |
| Mib_11_3_9_Bridge | 중계브릿지들의 대한 기본정보 |

## 2. DCU 상태 감시 기능 구현

[그림 3-9]는 DCU의 상태 감시 기능 흐름도를 나타내고 있다. CPU 사용률 및 메모리 사용률을 10초에 1번씩 6회 측정 후 그 평균값을 계산하여 이중 역할 망 관리 에이전트 MIB에 저장한다. 또한 DCU의 인터페이스 정보를 스캔하여 그 내용을 MIB에 저장한다. 그리고 이 모든 과정을 무한 반복 수행한다.



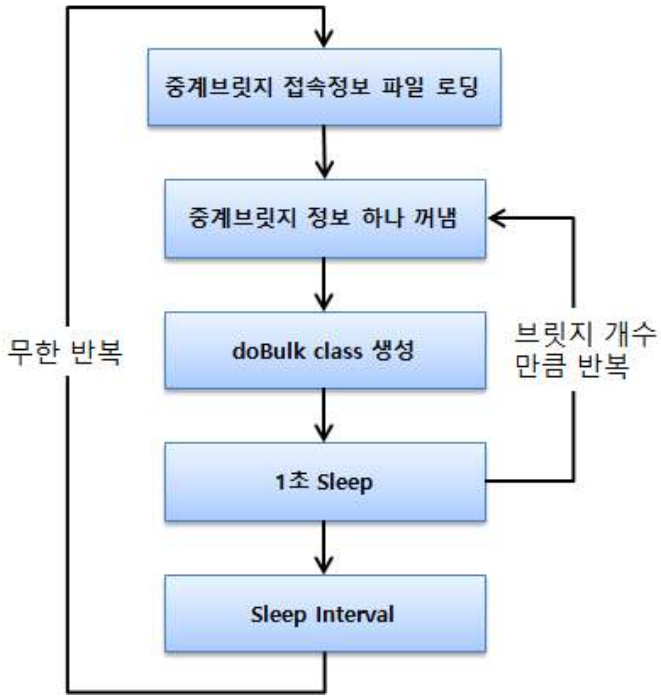

[그림 3-9] DCU 상태 감시 기능 흐름도

## 3. 매니저 기능 구현

다음 [그림 3-10]은 중계브릿지들로부터 망 관리 데이터를 수집하는 GetBulk class의 순서도를 나타내고 있다.

GetBulk class의 run 메서드는 while 루프를 무한 반복하게 되는데, 첫 번째로

중계브릿지의 ip와 port정보를 파일로부터 읽어 들여서 ArrayList 에 저장한 후 그 ArrayList를 doBulk 메서드를 호출시 인자로 넘긴다. 그리고 모든 중계브릿지로부터 MIB정보를 수집한 후 config.xml에서 정의한 수집 주기만큼 sleep한 후 또 처음부터 이러한 행위를 무한 반복 수행한다.

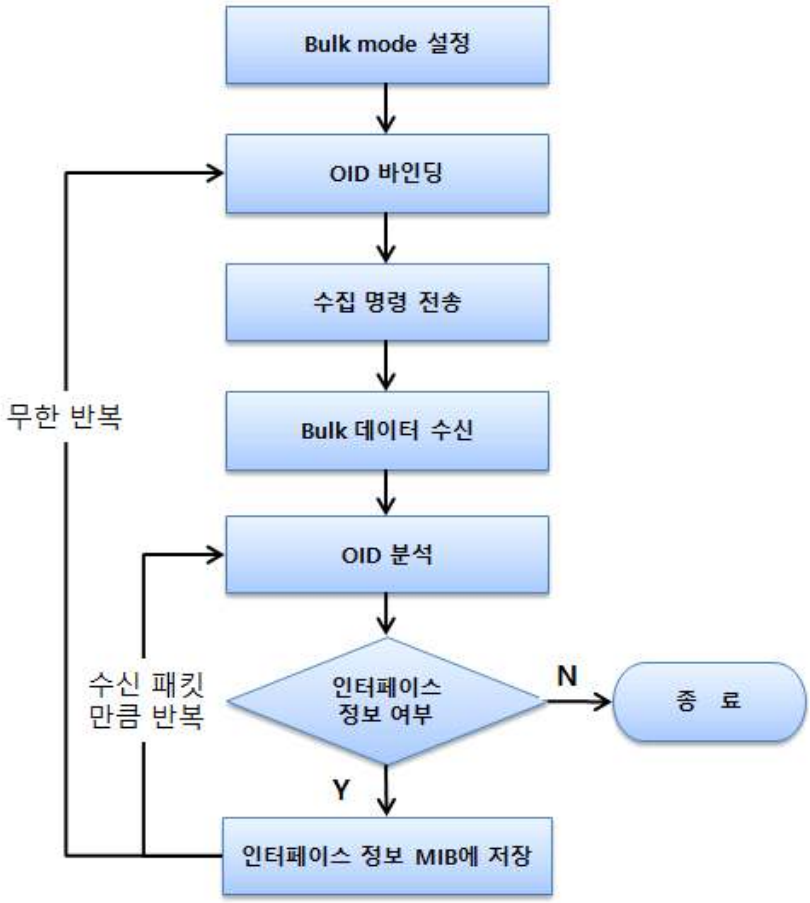



[그림 3-10] GetBulk class 순서도

다음 [그림 3-11]은 중계브릿지 한대로부터 실제적인 망 관리 데이터 수집 처리를 하는 doBulk class에 대한 설명이다.

우선 Bulk mode를 설정하고, 벌크로 전송 요청하는 개수를 100개로 설정하여, MIB의 OID개수를 최대 100개씩 전송하도록 설정한다. 그리고 전송 시작하는 OID 번호를 설정한 후 중계브릿지에 요청 명령을 전송한다. 그러면 중계브릿지는 최대 100개의 OID 데이터를 응답한다. 이 100개의 OID데이터를 하나씩 읽어 들여 분석하고 OID값이 모뎀의 원하는 인터페이스 정보인지를 판단한 후 유효한 값이면 해당 정보를 DCU(內)의 MIB에 저장한다. 이렇게 한 번에 수신 받은 벌크 데이터의 모든 분석과 저장이 끝나면 마지막으로 수신된 OID번호를 저장하고 있다가 거기서

부터 다음 100개를 요청해야 하므로 마지막 OID번호을 다시 바인딩하고 중계브릿지에 데이터 요청 명령을 전송한다. 이렇게 하여 모든 모뎀의 인터페이스 정보를 수신한 후에는 클래스를 종료 처리한다.




[그림 3-11] doBulk class 순서도

# Ⅳ. 실험 및 결과

본 장에서는 DCU(內)에 탑재하여 돌아가는 이중 역할 망 관리 에이전트를 설치하기 위하여 DCU에 JVM설치하고 어플리케이션을 기동하여 그 실행결과에 대해서 설명한다. 먼저 기존의 망 관리 에이전트들로부터 망 관리 정보를 수집하고 그 데이터들의 정확성을 검증하기 위해 중계브릿지에 있는 MIB정보와 DCU에 있는 MIB정보를 비교하여 일치하는 지를 검증하고, 상위 망 관리 서버에서 본 에이전트를 통해서 말단 모뎀들의 망 관리 정보들을 가져올 수 있는지 실험하였다.

또한 말단의 수 많은 에이전트들을 제어하며 중간에서 전단처리 기능을 하여 망 관리 서버의 부담을 줄여주고 서버의 가용성을 높일 수 있는지를 검증하기 위해 DCU에서 여러 대의 중계브릿지로부터 MIB 데이터를 수집할 때 걸리는 시간과 한 대의 DCU에서 수집된 똑 같은 양의 MIB데이터를 매니저(즉 PC)에서 수집할 때 걸리는 소요시간을 비교하는 실험을 하였다.

## A. 이중 역할 망 관리 에이전트 실행 결과

본 논문의 목표가 JAVA기반의 어플리케이션 개발이므로 우선 DCU [그림 4-1]에서 보듯이 oracle 사이트에서 32비트 리눅스 ARM MCU용 JVM을 다운받았다.


| Product / File Description | File Size | Download |
|---|---|---|
| Linux ARM 32 Hard Float ABI | 77.86 MB | jdk-8u121-linux-arm32-vfp-hflt.tar.gz |
| Linux ARM 64 Hard Float ABI | 74.83 MB | jdk-8u121-linux-arm64-vfp-hflt.tar.gz |
| Linux x86 | 162.41 MB | jdk-8u121-linux-i586.rpm |
| Linux x86 | 177.13 MB | jdk-8u121-linux-i586.tar.gz |
| Linux x64 | 159.96 MB | jdk-8u121-linux-x64.rpm |
| Linux x64 | 174.76 MB | jdk-8u121-linux-x64.tar.gz |
| Mac OS X | 223.21 MB | jdk-8u121-macosx-x64.dmg |
| Solaris SPARC 64-bit | 139.64 MB | jdk-8u121-solaris-sparcv9.tar.Z |
| Solaris SPARC 64-bit | 99.07 MB | jdk-8u121-solaris-sparcv9.tar.gz |
| Solaris x64 | 140.42 MB | jdk-8u121-solaris-x64.tar.Z |
| Solaris x64 | 96.9 MB | jdk-8u121-solaris-x64.tar.gz |
| Windows x86 | 189.36 MB | jdk-8u121-windows-i586.exe |
| Windows x64 | 195.51 MB | jdk-8u121-windows-x64.exe |

[그림 4-1] JVM 다운로드 사이트

그리고 다운받은 모듈을 DCU에 복사해서 넣고, [그림 4-2]처럼 압축을 푼다.

```
192.168.10.100:22 - root@linaro-alip: /home/linaro VT
메뉴(E) 수정(E) 설정(S) 제어(O) 창(W) 도움말(H)
root@linaro-alip:/home/linaro# ls -l
total 80292
drwxr-xr-x  2 linaro linaro     4096 Sep  1 09:00 Desktop
drwxr-xr-x  2 linaro linaro     4096 Sep  1 09:00 Templates
drwxr-xr-x 10 root   root       4096 Sep  1  2009 etc
-rwxr-xr-x  1 linaro linaro   257157 Sep  1  2009 hybrid_dcu_install_0.95.tar.gz
-rwxr-xr-x  1 linaro linaro 81562420 Jan  9  2017 jdk-8u111-linux-arm32-vfp-hflt.tar.gz
drwxr-xr-x  2 root   root       4096 Sep  1  2009 kdn_security
-rwxr-xr-x  1 linaro linaro   374832 Sep  1  2009 plc_modem_install_0.95.tar.gz
drwxr-xr-x  3 root   root       4096 Sep  1  2009 usr
root@linaro-alip:/home/linaro# tar xvfz jdk-8u111-linux-arm32-vfp-hflt.tar.gz
jdk1.8.0_111/COPYRIGHT
tar: jdk1.8.0_111/COPYRIGHT: time stamp 2016-09-23 09:15:33 is 222826226.583779599 s in the future
jdk1.8.0_111/LICENSE
tar: jdk1.8.0_111/LICENSE: time stamp 2016-09-23 09:15:33 is 222826226.581992799 s in the future
jdk1.8.0_111/README.html
tar: jdk1.8.0_111/README.html: time stamp 2016-09-23 09:15:33 is 222826226.581084799 s in the future
jdk1.8.0_111/THIRDPARTYLICENSEREADME.txt
tar: jdk1.8.0_111/THIRDPARTYLICENSEREADME.txt: time stamp 2016-09-23 09:15:33 is 222826226.572661699 s in the future
jdk1.8.0_111/bin/
jdk1.8.0_111/bin/jarsigner
tar: jdk1.8.0_111/bin/jarsigner: time stamp 2016-09-23 09:15:34 is 222826227.571015799 s in the future
jdk1.8.0_111/bin/xjc
tar: jdk1.8.0_111/bin/xjc: time stamp 2016-09-23 09:15:34 is 222826227.569972699 s in the future
jdk1.8.0_111/bin/pack200
tar: jdk1.8.0_111/bin/pack200: time stamp 2016-09-23 09:15:34 is 222826227.568955599 s in the future
jdk1.8.0_111/bin/javac
tar: jdk1.8.0_111/bin/javac: time stamp 2016-09-23 09:15:34 is 222826227.567914099 s in the future
jdk1.8.0_111/bin/java
tar: jdk1.8.0_111/bin/java: time stamp 2016-09-23 09:15:34 is 222826227.566885899 s in the future
jdk1.8.0_111/bin/javap
tar: jdk1.8.0_111/bin/javap: time stamp 2016-09-23 09:15:34 is 222826227.565830599 s in the future
jdk1.8.0_111/bin/wsgen
tar: jdk1.8.0_111/bin/wsgen: time stamp 2016-09-23 09:15:34 is 222826227.564835099 s in the future
jdk1.8.0_111/bin/unpack200
tar: jdk1.8.0_111/bin/unpack200: time stamp 2016-09-23 09:15:34 is 222826227.550672999 s in the future
jdk1.8.0_111/bin/keytool
tar: jdk1.8.0_111/bin/keytool: time stamp 2016-09-23 09:15:34 is 222826227.550230799 s in the future
jdk1.8.0_111/bin/jps
tar: jdk1.8.0_111/bin/jps: time stamp 2016-09-23 09:15:34 is 222826227.549871499 s in the future
jdk1.8.0_111/bin/jstat
tar: jdk1.8.0_111/bin/jstat: time stamp 2016-09-23 09:15:34 is 222826227.549501399 s in the future
jdk1.8.0_111/bin/jcmd
tar: jdk1.8.0_111/bin/jcmd: time stamp 2016-09-23 09:15:34 is 222826227.549201699 s in the future
```

[그림 4-2] DCU(內)에 JVM설치 과정

다음으로는 어느 경로에서나 JAVA가 실행될 수 있도록 '/etc/profile' 파일에 [그림 4-3]처럼 PATH 설정을 추가한다. 그리고 java가 작동하지는 버전을 확인하여 java버전이 '1.8.0_111'이 맞는지 확인한다. 이렇게 DCU에 JAVA가 작동할 수 있도록 JVM을 설치하고 환경설정을 마무리 하였다.

```
192.168.10.100:22 - root@linaro-alip: ~ VT
메뉴(F) 수정(E) 설정(S) 제어(O) 창(W) 도움말(H)
root@linaro-alip:~# cat /etc/profile
# /etc/profile: system-wide .profile file for the Bourne shell (sh(1))
# and Bourne compatible shells (bash(1), ksh(1), ash(1), ...).

if [ "$PS1" ]; then
  if [ "$BASH" ] && [ "$BASH" != "/bin/sh" ]; then
    # The file bash.bashrc already sets the default PS1.
    # PS1='\h:\w\$ '
    if [ -f /etc/bash.bashrc ]; then
      . /etc/bash.bashrc
    fi
  else
    if [ "`id -u`" -eq 0 ]; then
      PS1='# '
    else
      PS1='$ '
    fi
  fi
fi

# The default umask is now handled by pam_umask.
# See pam_umask(8) and /etc/login.defs.

if [ -d /etc/profile.d ]; then
  for i in /etc/profile.d/*.sh; do
    if [ -r $i ]; then
      . $i
    fi
  done
  unset i
fi

export PATH=$PATH:/usr/jdk1.8.0_111/bin


root@linaro-alip:~# java -version
java version "1.8.0_111"
Java(TM) SE Runtime Environment (build 1.8.0_111-b14)
Java HotSpot(TM) Client VM (build 25.111-b14, mixed mode)
root@linaro-alip:~#
root@linaro-alip:~#
```

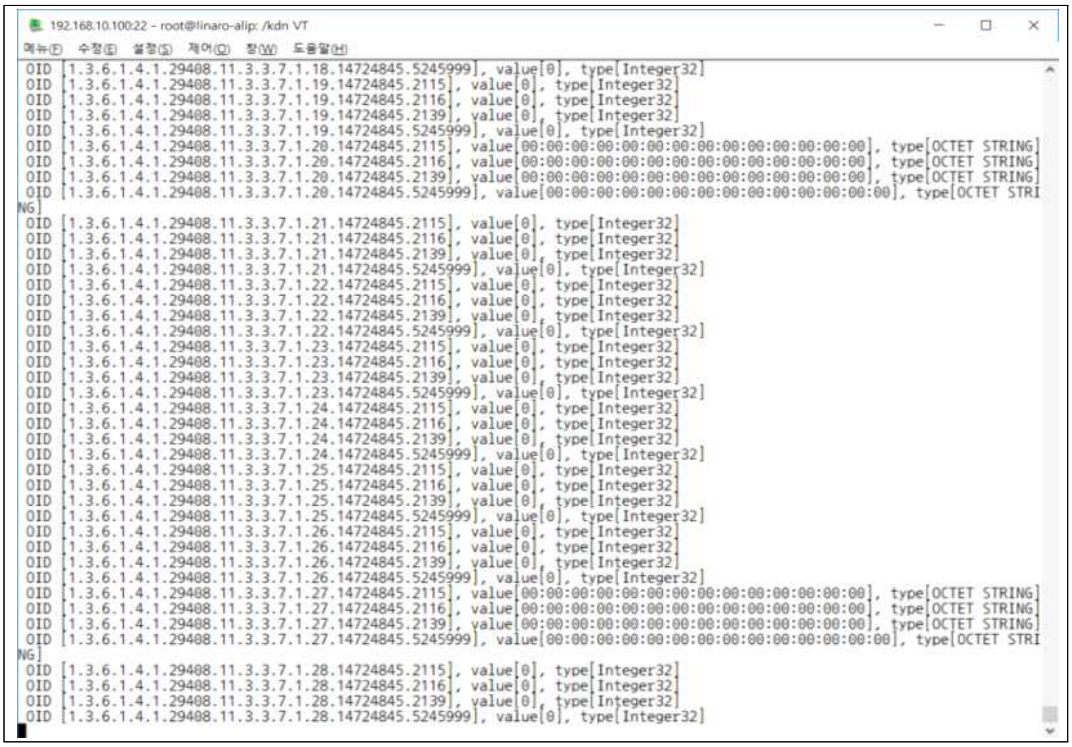
[그림 4-3] 환경 설정 및 JAVA 버전 확인

다음은 이중 역할 망 관리 에이전트 어플리케이션을 기동한 후 프로세서를 확인한 결과이다.

```
192.168.10.100:22 - root@linaro-alip: /kdn/bin VT
메뉴(F) 수정(E) 설정(S) 제어(O) 창(W) 도움말(H)
root@linaro-alip:/kdn/bin# ps -ef | grep NmsAgentKdn.jar
root     26870     1  8 15:42 pts/1    00:00:03 java -classpath ./NmsAgentKdn.jar:./snmp4j-agen
t-2.5.3.jar:./snmp4j-2.5.3.jar:./log4j-1.2.17.jar com.kdn.nms.Agent
root     26893 26850  0 15:43 pts/1    00:00:00 grep --color=auto NmsAgentKdn.jar
root@linaro-alip:/kdn/bin#
```

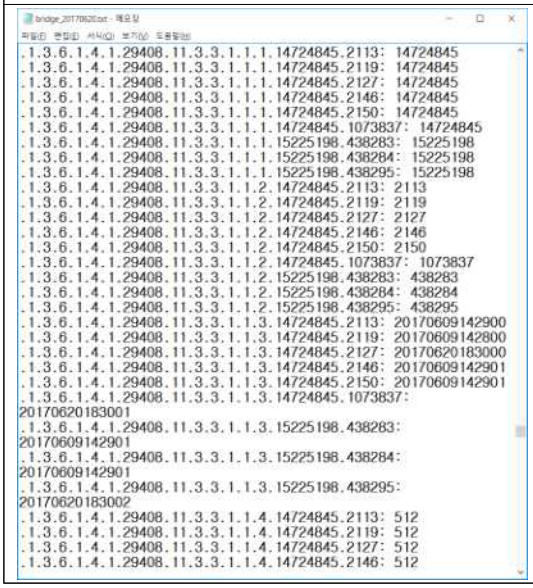
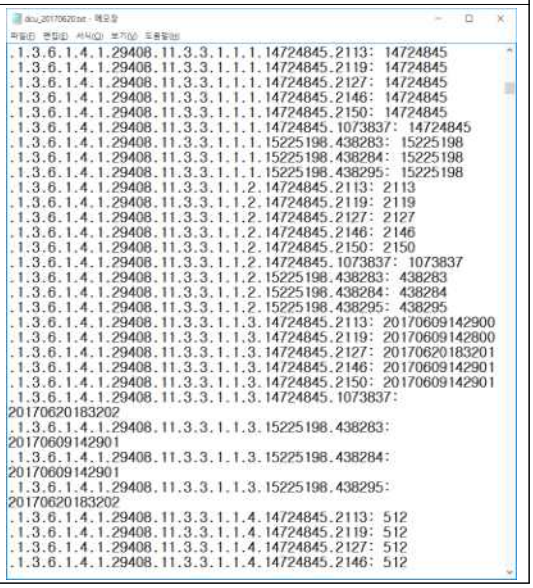
[그림 4-4] 어플리케이션 프로세스 기동 확인

[그림 4-5]는 DCU(內) 이중 역할 망 관리 에이전트가 동작 중일 때의 중계브릿지의 MIB 데이터를 수집하여 저장하는 로그 화면이다.




[그림 4-5] DCU(內) 이중 역할 망 관리 에이전트 동작 로그

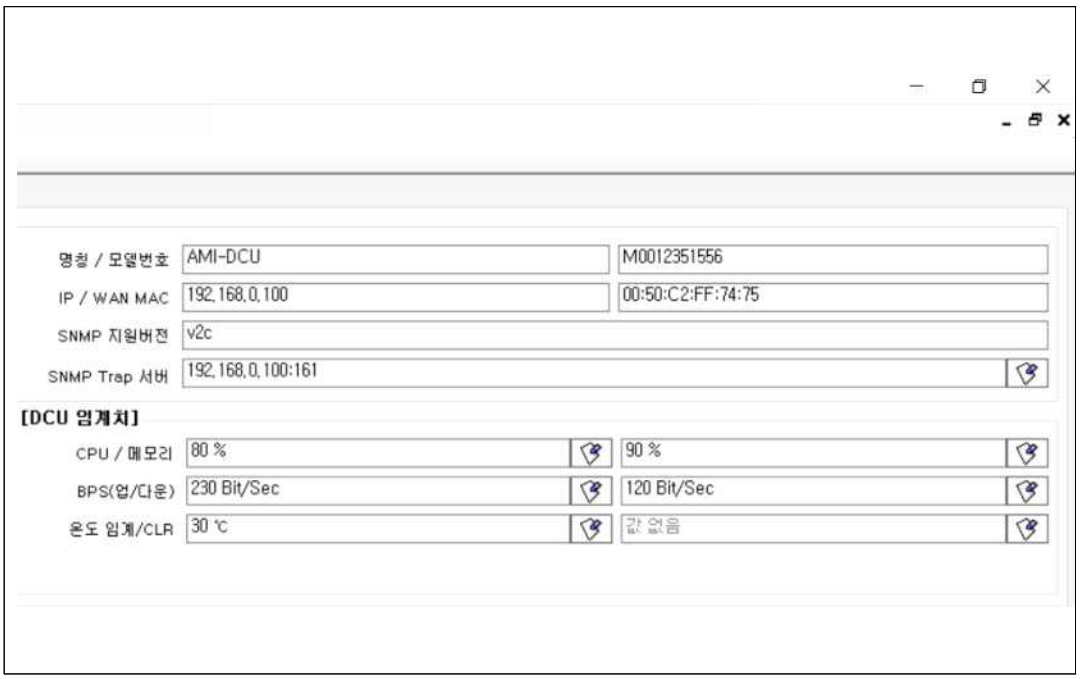
아래 그림은 중계브릿지 MIB정보와 그 중계브릿지의 MIB정보를 DCU(內)에 있는 이중 역할 망 관리 에이전트에서 수집한 결과와 비교하여 정확히 일치하는지 검증했다. 결과 검증을 위해서 점검PC를 중계브릿지에 직접 연결하고 중계브릿지에 탑재돼서 하위 모뎀들에 대한 정보를 취득 제어하는 망 관리 에이전트에 접속하여 망 관리 정보를 취득했고 그 결과는 왼쪽 그림이다. 다음은 점검PC를 DCU에 직접 연결하고 이중 역할 망 관리 에이전트에 접속하여 망 관리 정보를 취득 했고 그 결과는 오른쪽 그림이다. 보이는 것처럼 두 결과가 일치하는 것을 증명하였다.

| 중계브릿지에서의 모뎀 인터페이스 정보 | DCU內의 이중 역할 망 관리 에이전트에서의 모뎀 인터페이스 정보 |
|---|---|
|  |  |

[그림 4-6] 중계브릿지 에이전트와 이중 역할 망 관리 에이전트의 MIB정보 비교

[그림 4-7]은 망 관리 시스템 매니저로부터 DCU(內) 이중 역할 망 관리 에이전트에게 망 관리 정보를 요청하여 수신한 데이터를 화면에 보여주고 있다.
DCU ID 등 기본정보는 본 에이전트의 환경설정 파일에서 정의한 내용을 보여주고 있고, 하단부분의 DCU상태부분에 있는 CPU 사용률 및 메모리 사용률 등은 주기적으로 본 에이전트가 측정하여 기록한 부분이다. [그림 4-8]에 있는 DCU 임계치 부분도 default로 설정한 값을 보여주고 있다.


[그림 4-7] DCU 기본정보 Ⅰ




[그림 4-8] DCU 기본정보 Ⅱ

아래 그림은 통신단말 특성명세를 조회한 모습니다. 특히 관리PLC 모뎀 개수가 46대로 하위 모뎀들이 몇 개가 있는지를 보여주고 있는데, 이는 본 에이전트가 주기적으로 MIB정보를 스캔하여 모뎀의 총 수량을 정의된 OID에 기록함으로써 아래와 같은 결과를 확인했다.

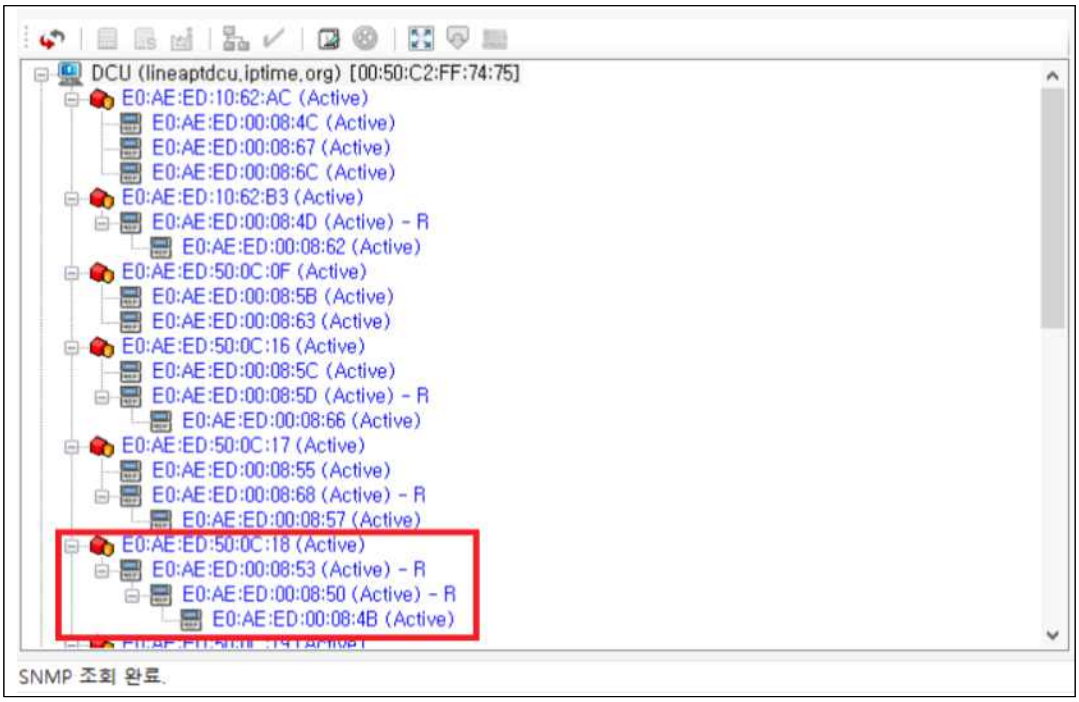

[그림 4-9] 통신단말 특성명세

[그림 4-10]은 이더넷(Ethernet)정보를 조회한 결과이며, 본 에이전트가 주기적으로 DCU에 등록 돼 있는 인터페이스 정보들을 스캔하여 그 결과를 미리에 정의된 OID에 기록 관리하고 있는 것이며, 왼쪽 eth1 인터페이스를 클릭하면 오른쪽에 eth1 인터페이스에 대한 자세한 정보들(IP 주소, MAC 주소, 들어온 패킷수, 나간 패킷수 등)을 보여주고 있다.

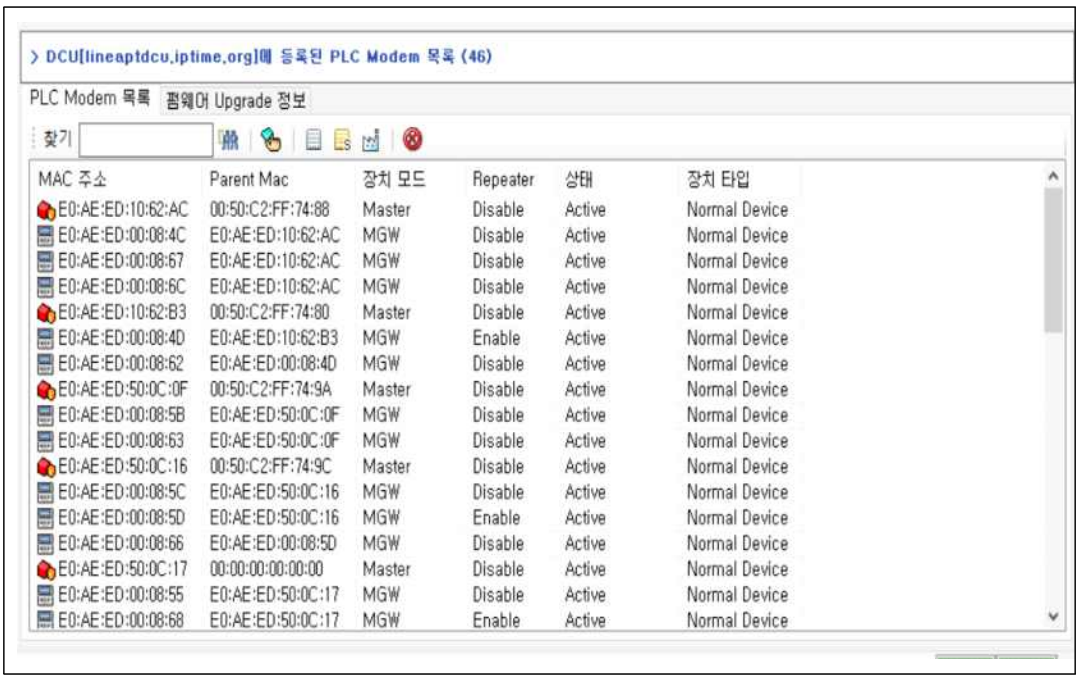

[그림 4-10] 이더넷(Ethernet) 정보

이제 부터는 망 관리 시스템 매니저의 하단부분에 대해서 설명하고자 한다. 먼저 [그림 4-11]은 본 에이전트가 관리하고 있는 PLC 모뎀 목록이다. 먼저 첫줄에는 DCU의 MAC 주소이고, 그 아래로 빨간색 아이콘이 표시돼 있는 MAC 주소들은 중계브릿지들의 마스터 모뎀 MAC 주소이며, 그 하위로 Tree 형태로 슬레이브 모뎀의 MAC 주소들이 붙어 있는 모습니다. 그리고 하단 빨간 박스부분은 중계브릿지의 MAC 주소 “E0:AE:ED:50:0C:18 (Active)” 이며, “E0:AE:ED:00:08:53 (Active) -R” MAC주소는 중계브릿지에 연결돼 있는 첫 번째 슬레이브 모뎀의 MAC주소이고 “-R”은 이 슬레이브 모뎀이 Repeater 역할을 수행하고 있다는 표시이며 이 슬레이브 모뎀을 통해서 “E0:AE:ED:00:08:50 (Active) -R” 이 연결돼 있고, 다시 그 아래로 “E0:AE:ED:00:08:4B (Active)”가 연결돼 있는 모습이다.

그리고 [그림 4-12]는 PLC 모뎀 목록을 리스트 형태로 상세히 보여 주고 있다.



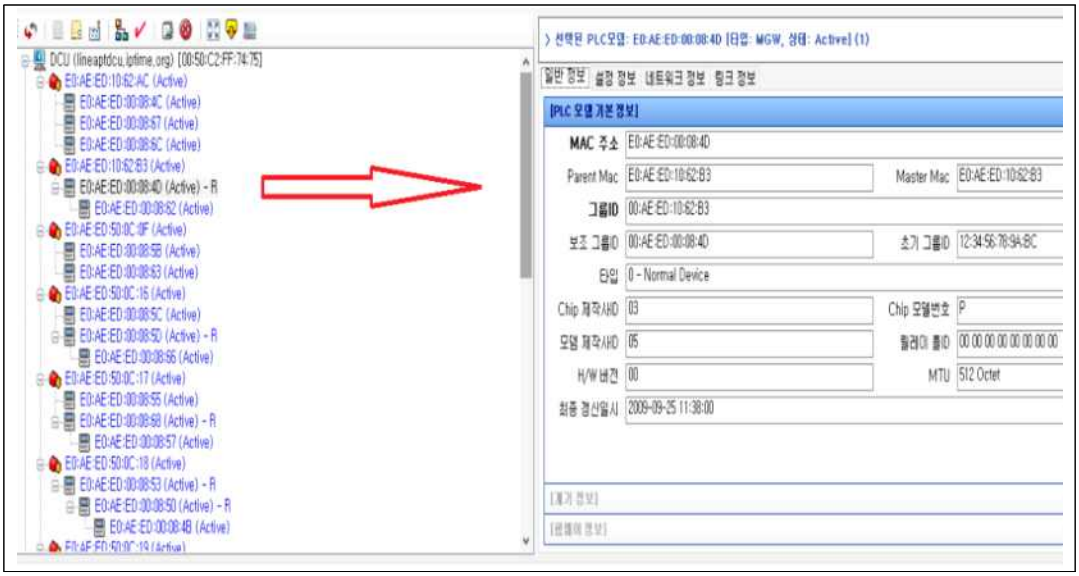

[그림 4-11] PLC 모뎀 목록 Ⅰ




| MAC 주소 | Parent Mac | 장치 모드 | Repeater | 상태 | 장치 타입 |
|---|---|---|---|---|---|
| E0:AE:ED:10:62:AC | 00:50:C2:FF:74:88 | Master | Disable | Active | Normal Device |
| E0:AE:ED:00:08:4C | E0:AE:ED:10:62:AC | MGW | Disable | Active | Normal Device |
| E0:AE:ED:00:08:67 | E0:AE:ED:10:62:AC | MGW | Disable | Active | Normal Device |
| E0:AE:ED:00:08:6C | E0:AE:ED:10:62:AC | MGW | Disable | Active | Normal Device |
| E0:AE:ED:10:62:B3 | 00:50:C2:FF:74:80 | Master | Disable | Active | Normal Device |
| E0:AE:ED:00:08:4D | E0:AE:ED:10:62:B3 | MGW | Enable | Active | Normal Device |
| E0:AE:ED:00:08:62 | E0:AE:ED:00:08:4D | MGW | Disable | Active | Normal Device |
| E0:AE:ED:50:0C:0F | 00:50:C2:FF:74:9A | Master | Disable | Active | Normal Device |
| E0:AE:ED:00:08:5B | E0:AE:ED:50:0C:0F | MGW | Disable | Active | Normal Device |
| E0:AE:ED:00:08:63 | E0:AE:ED:50:0C:0F | MGW | Disable | Active | Normal Device |
| E0:AE:ED:50:0C:16 | 00:50:C2:FF:74:9C | Master | Disable | Active | Normal Device |
| E0:AE:ED:00:08:5C | E0:AE:ED:50:0C:16 | MGW | Disable | Active | Normal Device |
| E0:AE:ED:00:08:5D | E0:AE:ED:50:0C:16 | MGW | Enable | Active | Normal Device |
| E0:AE:ED:00:08:66 | E0:AE:ED:00:08:5D | MGW | Disable | Active | Normal Device |
| E0:AE:ED:50:0C:17 | 00:00:00:00:00:00 | Master | Disable | Active | Normal Device |
| E0:AE:ED:00:08:55 | E0:AE:ED:50:0C:17 | MGW | Disable | Active | Normal Device |
| E0:AE:ED:00:08:68 | E0:AE:ED:50:0C:17 | MGW | Enable | Active | Normal Device |

[그림 4-12] PLC 모뎀 목록 Ⅱ

[그림 4-13]은 왼쪽 PLC 모뎀 목록에서 “E0:AE:ED:00:08:4D” MAC 주소를 클릭했을 때 선택된 PLC 모뎀에 대한 일반정보를 보여주고 있다.

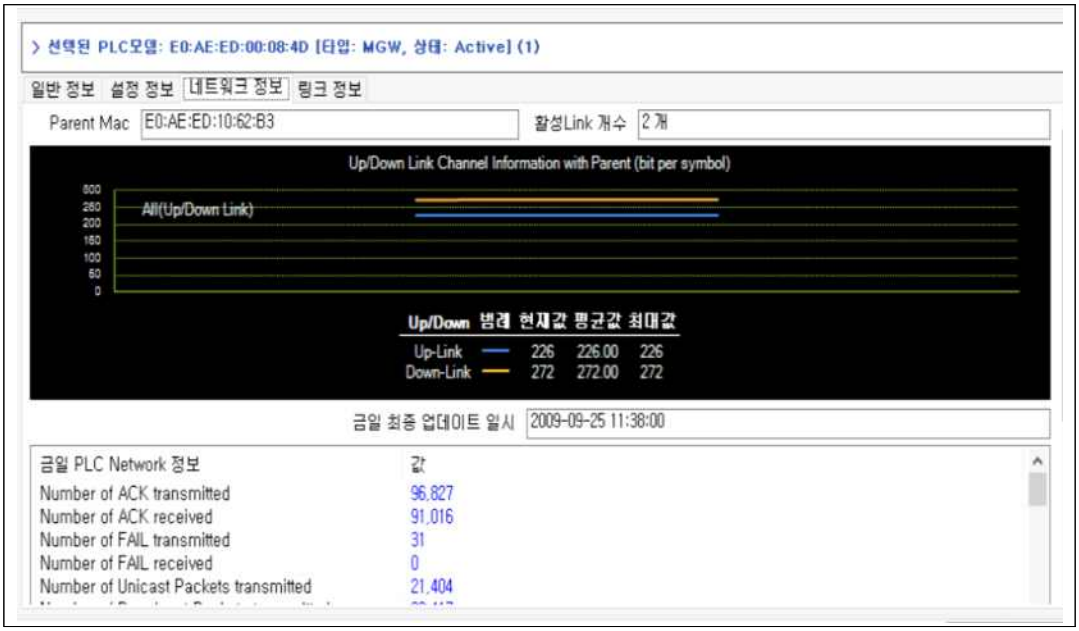



[그림 4-13] PLC 모뎀 일반정보

[그림 4-14]는 선택된 PLC 모뎀의 설정정보를 보여주고 있다.


[그림 4-14] PLC 모뎀 설정 정보

아래 그림은 선택된 PLC 모뎀에 대한 네트워크 상태 정보를 상세히 보여주고 있다.




[그림 4-15] PLC 모뎀 네트워크 정보

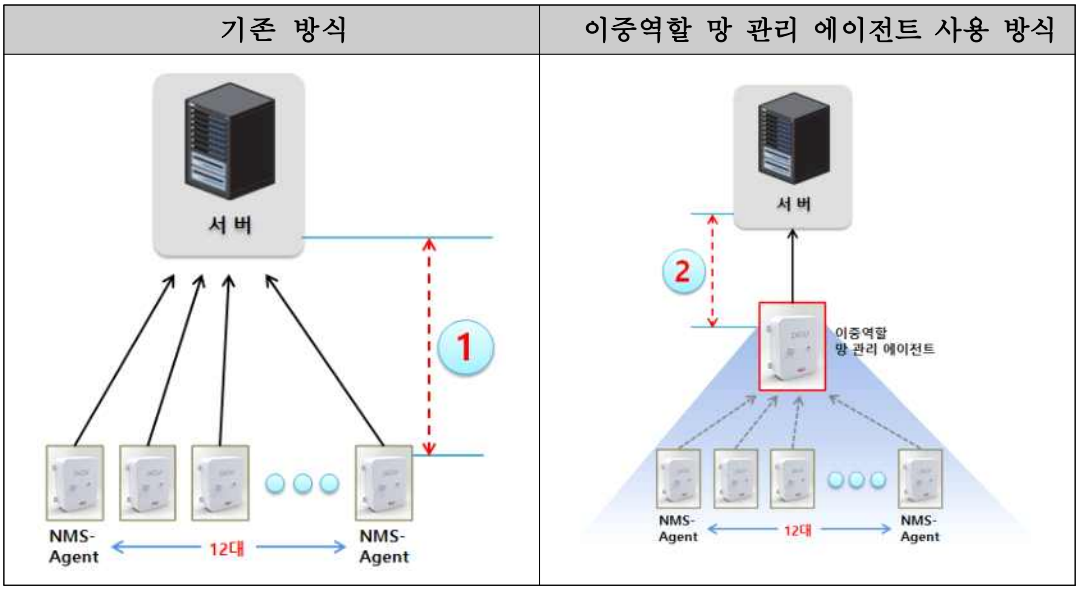
아래 그림은 선택된 PLC 모뎀에 대한 링크 상태 정보를 상세히 보여주고 있다.


[그림 4-16] PLC 모뎀 링크 정보

본 이중 역할 망 관리 에이전트 실행 결과에서 보듯이 크게는 DCU하위의 중계 브릿지들과 통신하여 하위 모뎀들에 대한 망 관리 정보를 수집하는 기능과 DCU(內)의 기본정보 및 성능정보들을 주기적으로 모니터링 하여 그 결과를 관리 정보 베이스에 저장하는 기능을 성공적으로 수행하였음을 알 수 있다. 또한 그렇게 수집 관리한 정보들에 대해서 서버나 매니저 등으로부터 정보 조회 요청에 응답하여 그 결과를 전송하는 기능을 충실히 수행하였다. 그리고 [그림 3-2]의 네트워크 구성도처럼 DCU상위와 하위의 네트워크를 분리하여 서버에서 중계브릿지까지 직접적인 접속이 불가능한 상태에서 중계브릿지 하위의 모뎀들의 망 관리 정보들까지도 서버에서 취득이 가능함을 증명하였다.

## B. MIB 데이터 수집 효율 실험

두 번째 시험으로는 같은 양의 MIB데이터를 여러 대의 중계브릿지로부터 수집할 때와 하나의 DCU로부터 수집할 때의 수집 시간을 비교를 하였다. 먼저 이 실험을 수행할 때의 네트워크 구성도는 [그림 4-17]과 같다. 실험 방법은 기존 방식처럼 서버에서 망 관리 에이전트가 탑재 돼 있는 중계브릿지 12대로부터 망 관리 MIB 데이터를 취득한다. 그리고 두 번째로 이중역할 망 관리 에이전트 사용 방식처럼 이미 수집되어 있는 MIB데이터를 서버로부터 이중역할 망 관리 에이전트가 탑재 돼 있는 DCU 1대로부터 똑 같은 양의 MIB 데이터를 취득하여 두 결과를 비교한다.




[그림 4-17] 네트워크 구성도

[표 4-1] MIB데이터 수집 효율 실험 결과

| 실험 방식 | 수집시작시각 | 수집종료시각 | 소요시간 | 차이 비교 |
|---|---|---|---|---|
| ①기존 방식 | 23:07:13 | 23:08:07 | 54 | 약 1/3 |
| ②이중역할 망 관리 에이전트 방식 | 10:03:59 | 10:04:15 | 16 | |

```
agentLogs20170703_최종로그.txt
1  [2017/07/03 23:07:12.245] [INFO]  Initial Config Loading success..
2  [2017/07/03 23:07:13.513] [INFO]  DCU NMS Agent Start... IP[192.168.0.100], Port[161]
3  [2017/07/03 23:07:13.582] [INFO]  Optional MIB Infomation added..
4  [2017/07/03 23:07:13.589] [INFO]  EDN NMS Agent all started..
5  [2017/07/03 23:07:13.615] [INFO]  Discovery Interface Name : [eth0]
6  [2017/07/03 23:07:13.622] [INFO]  Bridge [192.168.10.111/161] getBulk start...
7  [2017/07/03 23:07:13.623] [INFO]  Discovery Interface Name : [eth1]
8  [2017/07/03 23:07:13.629] [INFO]  Discovery Interface Name : [lo]
9  [2017/07/03 23:07:13.646] [INFO]  Discovery Interface Name : [wisun0]
10 [2017/07/03 23:07:14.298] [INFO]  Master Modem : OID [1.3.6.1.4.1.29408.11.3.3.1.1.31.14724845.5245999], value[1], Mac[E0:AE:ED:50:0C:2F]
11 [2017/07/03 23:07:16.736] [INFO]  Bridge [192.168.10.111/161] getBulk end...
12 [2017/07/03 23:07:17.738] [INFO]  Bridge [192.168.10.112/161] getBulk start...
13 [2017/07/03 23:07:18.369] [INFO]  Master Modem : OID [1.3.6.1.4.1.29408.11.3.3.1.1.31.14724845.5245978], value[1], Mac[E0:AE:ED:50:0C:1A]
14 [2017/07/03 23:07:21.031] [INFO]  Bridge [192.168.10.112/161] getBulk end...
15 [2017/07/03 23:07:22.032] [INFO]  Bridge [192.168.10.113/161] getBulk start...
16 [2017/07/03 23:07:22.636] [INFO]  Master Modem : OID [1.3.6.1.4.1.29408.11.3.3.1.1.31.14724845.5246001], value[1], Mac[E0:AE:ED:50:0C:31]
17 [2017/07/03 23:07:25.486] [INFO]  Bridge [192.168.10.113/161] getBulk end...
18 [2017/07/03 23:07:26.488] [INFO]  Bridge [192.168.10.114/161] getBulk start...
19 [2017/07/03 23:07:27.306] [INFO]  Master Modem : OID [1.3.6.1.4.1.29408.11.3.3.1.1.31.14724845.5245967], value[1], Mac[E0:AE:ED:50:0C:0F]
20 [2017/07/03 23:07:30.827] [INFO]  Bridge [192.168.10.114/161] getBulk end...
21 [2017/07/03 23:07:31.829] [INFO]  Bridge [192.168.10.102/161] getBulk start...
22 [2017/07/03 23:07:32.067] [INFO]  Master Modem : OID [1.3.6.1.4.1.29408.11.3.3.1.1.31.14724845.5245988], value[1], Mac[E0:AE:ED:50:0C:24]
23 [2017/07/03 23:07:32.598] [INFO]  Bridge [192.168.10.102/161] getBulk end...
24 [2017/07/03 23:07:33.602] [INFO]  Bridge [192.168.10.121/161] getBulk start...
25 [2017/07/03 23:07:34.195] [INFO]  Master Modem : OID [1.3.6.1.4.1.29408.11.3.3.1.1.31.14724845.5245974], value[1], Mac[E0:AE:ED:50:0C:16]
26 [2017/07/03 23:07:36.759] [INFO]  Bridge [192.168.10.121/161] getBulk end...
27 [2017/07/03 23:07:37.764] [INFO]  Bridge [192.168.10.122/161] getBulk start...
28 [2017/07/03 23:07:38.357] [INFO]  Master Modem : OID [1.3.6.1.4.1.29408.11.3.3.1.1.31.14724845.5245977], value[1], Mac[E0:AE:ED:50:0C:19]
29 [2017/07/03 23:07:40.973] [INFO]  Bridge [192.168.10.122/161] getBulk end...
30 [2017/07/03 23:07:41.975] [INFO]  Bridge [192.168.10.123/161] getBulk start...
31 [2017/07/03 23:07:42.454] [INFO]  Master Modem : OID [1.3.6.1.4.1.29408.11.3.3.1.1.31.14724845.1073836], value[1], Mac[E0:AE:ED:10:62:AC]
32 [2017/07/03 23:07:44.479] [INFO]  Bridge [192.168.10.123/161] getBulk end...
33 [2017/07/03 23:07:45.481] [INFO]  Bridge [192.168.10.124/161] getBulk start...
34 [2017/07/03 23:07:46.255] [INFO]  Master Modem : OID [1.3.6.1.4.1.29408.11.3.3.1.1.31.14724845.1073843], value[1], Mac[E0:AE:ED:10:62:B3]
35 [2017/07/03 23:07:49.627] [INFO]  Bridge [192.168.10.124/161] getBulk end...
36 [2017/07/03 23:07:50.629] [INFO]  Bridge [192.168.10.131/161] getBulk start...
37 [2017/07/03 23:07:51.110] [INFO]  Master Modem : OID [1.3.6.1.4.1.29408.11.3.3.1.1.31.14724845.5245976], value[1], Mac[E0:AE:ED:50:0C:18]
38 [2017/07/03 23:07:52.907] [INFO]  Bridge [192.168.10.131/161] getBulk end...
39 [2017/07/03 23:07:53.909] [INFO]  Bridge [192.168.10.132/161] getBulk start...
40 [2017/07/03 23:07:54.488] [INFO]  Master Modem : OID [1.3.6.1.4.1.29408.11.3.3.1.1.31.14724845.5246000], value[1], Mac[E0:AE:ED:50:0C:30]
41 [2017/07/03 23:07:57.093] [INFO]  Bridge [192.168.10.132/161] getBulk end...
42 [2017/07/03 23:07:58.096] [INFO]  Bridge [192.168.10.133/161] getBulk start...
43 [2017/07/03 23:07:58.878] [INFO]  Master Modem : OID [1.3.6.1.4.1.29408.11.3.3.1.1.31.14724845.5245975], value[1], Mac[E0:AE:ED:50:0C:17]
44 [2017/07/03 23:08:02.283] [INFO]  Bridge [192.168.10.133/161] getBulk end...
45 [2017/07/03 23:08:03.285] [INFO]  Bridge [192.168.10.134/161] getBulk start...
46 [2017/07/03 23:08:04.053] [INFO]  Master Modem : OID [1.3.6.1.4.1.29408.11.3.3.1.1.31.14724845.5246002], value[1], Mac[E0:AE:ED:50:0C:32]
47 [2017/07/03 23:08:07.232] [INFO]  Bridge [192.168.10.134/161] getBulk end...
48 [2017/07/03 23:08:13.714] [INFO]  Cpu Usage : [28.09%], Memory Usage : [32.18%]
```

[그림 4-18] 기존 방식 수집 로그

```
agentLogs20170705_서버로그.txt
41 [2017/07/05 10:03:58.766] [INFO]  Initial Config Loading success..
42 [2017/07/05 10:03:58.767] [INFO]  DCU NMS Agent Start... IP[192.168.0.20], Port[161]
43 [2017/07/05 10:03:59.210] [INFO]  DCU NMS Agent Start... IP[192.168.0.20], Port[161]
44 [2017/07/05 10:03:59.217] [INFO]  Optional MIB Infomation added..
45 [2017/07/05 10:03:59.222] [INFO]  EDN NMS Agent all started..
46 [2017/07/05 10:03:59.225] [INFO]  Bridge [192.168.0.100/161] getBulk start...
47 [2017/07/05 10:04:13.873] [INFO]  Master Modem : OID [1.3.6.1.4.1.29408.11.3.3.1.1.31.14724845.1073836], value[1], Mac[E0:AE:ED:10:62:AC]
48 [2017/07/05 10:04:13.875] [INFO]  Master Modem : OID [1.3.6.1.4.1.29408.11.3.3.1.1.31.14724845.1073843], value[1], Mac[E0:AE:ED:10:62:B3]
49 [2017/07/05 10:04:13.876] [INFO]  Master Modem : OID [1.3.6.1.4.1.29408.11.3.3.1.1.31.14724845.5245967], value[1], Mac[E0:AE:ED:50:0C:0F]
50 [2017/07/05 10:04:13.877] [INFO]  Master Modem : OID [1.3.6.1.4.1.29408.11.3.3.1.1.31.14724845.5245974], value[1], Mac[E0:AE:ED:50:0C:16]
51 [2017/07/05 10:04:13.878] [INFO]  Master Modem : OID [1.3.6.1.4.1.29408.11.3.3.1.1.31.14724845.5245975], value[1], Mac[E0:AE:ED:50:0C:17]
52 [2017/07/05 10:04:13.879] [INFO]  Master Modem : OID [1.3.6.1.4.1.29408.11.3.3.1.1.31.14724845.5245976], value[1], Mac[E0:AE:ED:50:0C:18]
53 [2017/07/05 10:04:13.880] [INFO]  Master Modem : OID [1.3.6.1.4.1.29408.11.3.3.1.1.31.14724845.5245977], value[1], Mac[E0:AE:ED:50:0C:19]
54 [2017/07/05 10:04:13.882] [INFO]  Master Modem : OID [1.3.6.1.4.1.29408.11.3.3.1.1.31.14724845.5245978], value[1], Mac[E0:AE:ED:50:0C:1A]
55 [2017/07/05 10:04:13.883] [INFO]  Master Modem : OID [1.3.6.1.4.1.29408.11.3.3.1.1.31.14724845.5245999], value[1], Mac[E0:AE:ED:50:0C:2F]
56 [2017/07/05 10:04:13.884] [INFO]  Master Modem : OID [1.3.6.1.4.1.29408.11.3.3.1.1.31.14724845.5246000], value[1], Mac[E0:AE:ED:50:0C:30]
57 [2017/07/05 10:04:13.885] [INFO]  Master Modem : OID [1.3.6.1.4.1.29408.11.3.3.1.1.31.14724845.5246001], value[1], Mac[E0:AE:ED:50:0C:31]
58 [2017/07/05 10:04:13.887] [INFO]  Master Modem : OID [1.3.6.1.4.1.29408.11.3.3.1.1.31.14724845.5246002], value[1], Mac[E0:AE:ED:50:0C:32]
59 [2017/07/05 10:04:15.593] [INFO]  Bridge [192.168.0.100/161] getBulk end...
60 [2017/07/05 10:04:59.589] [INFO]  Cpu Usage : [4.31%], Memory Usage : [14.52%]
```

[그림 4-19] 이중역할 망 관리 에이전트 방식 수집 로그

MIB 데이터 수집 효율에 대한 실험 결과는 먼저 [그림 4-18]처럼 기존 방식처럼 서버에서 망 관리 에이전트가 탑재 돼 있는 중계브릿지 12대로부터 망 관리 MIB 데이터를 수집한 결과는 23:07:13 초에 시작해서 23:08:07 에 수집 완료 됐으므로 54초

소요됐음을 알 수 있다. 두 번째는 [그림 4-19]처럼 이중역할 망 관리 에이전트 사용 방식처럼 이미 수집되어 있는 MIB데이터를 서버로부터 이중역할 망 관리 에이전트가 탑재 돼 있는 DCU 1대로부터 똑 같은 양의 MIB 데이터를 수집하였다. 그 결과 10:03:59 초에 시작하여 10:04:15 초에 수집 완료 됐으므로 16초가 소요됐음을 알 수 있다. 본 실험에서 알 수 있듯이 본 에이전트가 중간에 위치하여 하위 에이전트들로부터 망 관리 데이터를 수집 관리한다면 서버의 성능 면에서 월등히 효과적인 것을 알 수 있다. 여러 장비를 접속하여 데이터를 수집하는 것이 느린 이유는 각 장비를 접속할 때 세션을 열고 닫는 시간이 더 소요되기 때문이다. 단, 중계브릿지의 망 관리 에이전트로부터 DCU의 이중역할 망 관리 에이전트까지 수집돼는 시간은 배제하였다. 이유는 우선 이중역할 망 관리 에이전트는 하위에 망 관리 에어전트로부터 주기적으로 MIB데이터를 수신 관리하고 있으므로 배제했고 또한 서버의 가용성 측면에서 고려했기 때문이다.

# V. 결론 및 제언

본 논문에서는 AMI시스템 구축에 걸림돌이 되고 있는 PLC망의 음영구간에 대한 문제점을 해결하기 위해 유무선 혼합으로 네트워크를 복잡하게 구성하였을 때의 망 관리 시스템을 유연하게 구성하기 위하여 기존의 고속PLC용 망 관리 에이전트를 중계브릿지에 설치하고, DCU(內)에는 본 에이전트를 설치하여 서버와 에이전트간 단일 구조가 아닌 이단구조 즉 이중 역할 매니저 형태로 구성하여 실험 및 실증을 하였고 그 결과 서버에서 네트워크가 직접적으로 연결이 되지 않은 최하위 말단 모뎀들의 망 관리 정보들까지 정상적으로 수집이 가능함을 증명하였다.

그래서 첫 번째 얻는 장점은 위 실험에서 증명하였듯이 저압AMI시스템에서 사용하던 방식인 DCU의 망 관리 에이전트와 망 관리 서버 간 직접 통신방식에 비해서 이중 역할 망 관리 에이전트가 중간에 끼어 들면서 서버에서는 무수히 많은 망 관리 에이전트들로부터 직접 망 관리 데이터를 수집하던 것 보다 훨씬 적은 이중 역할 망 관리 에이전트들과 통신하므로 서버의 부담을 주여 주고, 서버의 가용성을 높일 수 있다.

두 번째로 얻는 장점은 하나의 장비에 이기종 에이전트 탑재가 가능하며, 서버에서 TCP/IP 네트워크가 직접 연결되지 않은 하위 PLC망 및 Wi-SUN 무선망 등의 망 관리 정보를 본 에이전트를 통해 TCP/IP망 서버에 전송함으로써 최하위 말단 장비까지 관리가 가능해 IoT실현이 가능하다는 점이다.

그러나 이러한 장점들 외에도 단점이 존재한다. 모뎀들을 제어하기 위해서 종전의 방식은 에이전트 하나만 거치면 제어가 될 수 있었으나 이중 역할 망 관리 에이전트가 추가되면서 에이전트를 두 개를 거치게 돼 모뎀의 제어나 업그레이드 등 수행 시에 제어 속도가 떨어질 수 있고, 네트워크 구조가 복잡해지면서 장애 포인트가 더 늘어 관리에 어려움을 겪을 수 있다.

그래서 본 논문에서 제시한 이중 역할 망 관리 에이전트는 기존 저압AMI에서 사용하고 있는 DCU와 고속PLC용 모뎀을 직접 연결하는 구조를 사용하면서 PLC 통신의 음영지역이 생겨 무선 통신방식 등의 혼합방식이 불가피한 일부 구간에만 적용하는 것이 바람직할 것으로 판단된다.

본 연구에서 제시한 에이전트의 새로운 구조인 이중 역할 망 관리 에이전트는

유무선 혼합 AMI시스템을 통해서 검증을 했지만 더 나아가 이와 유사한 구조 등 무수히 많은 IoT단말들을 제어하는데 활용한다면 큰 효과를 볼 수 있을 것이며 이런 측면에서 상당히 의미 있는 연구라고 생각된다.

국내에서는 JAVA를 이용한 망 관리 에이전트 개발 사례가 많지 않아서 참고 문헌 등의 레퍼런스가 없었다. 앞으로 국내에서도 JAVA를 이용한 망 관리 에이전트 개발이 활발히 진행되었으면 한다. 이와 더불어 JAVA보다 더 가볍고 소형장비에 탑재가 용이한 Python을 사용한 망 관리 에이전트 개발의 대한 연구가 필요하다.

# 참고문헌

[1] [국내. 저자 있음] seokjooson. (2013), “AMI란 무엇인가요 ?” 네이버블로그, http://blog.naver.com/PostView.nhn?blogId=seokjooson&logNo=30169458468&parentCategoryNo=&categoryNo=23&viewDate=&isShowPopularPosts=false&from=postView, (2017-5-31 방문).

[2] 李熙珍. "광무선 센서네트워크를 위한 전력선통신기반 가시광통신 시스템 구현." 석사학위, 한국해양대학, 2012.

[3] 김현종, 윤상흠, 김호. "전력선통신 기술동향 및 전망" 전자공학회지 30:1 (2003.1): 7.

[4] 김재문, 김양수, 안승호, 이희준, 이종구. "전기철도의 전력선 통신망 설계에 관한 연구" 한국산학기술학회 논문지 11:10 (2010): 6.

[5] 전력연구원 배전연구소, “지중구간 PLC AMI 및 신형전력량계 통신기술 개발(최종보고서)”, 한국전력공사 전력연구원 (2015.10): 1.

[6] 박병석. "PLC(전력선 통신) 기술개발 및 표준화 동향" 대한전기협회 전기저널 1:426 (2012.6): 12.

[7] 김영일, 박소정, 최문석, 장경석, 박병석, 정남준. "AMI기반의 검침네트워크관리 시스템 개발" 대한전기학회 학술대회 논문집 1 (2015.7): 2.

[8] 이해준. "스마트그리드 통신단말을 위한 유니버설미들웨어플랫폼 연구." 석사학위, 한국산업기술대학교 지식기반기술에너지대학원, 2014.

[9] 이승대. "전력선 통신을 이용한 원격 디밍 제어 시스템 설계" 한국컴퓨터산업교육학회 10:3 (2009): 6.

[10] 주종철. "전력선통신의 국내 표준화방향에 關한 硏究." 석사학위, 인제대학교 첨단산업기술대학원, 2007.

[11] 김지호, 이향범. "전력선 통신을 위한 시장 동향 및 발전 전망" 한국정보통신설비학회 학술대회 1 (2008): 6.

[12] 이홍희, 김관수. "변복조 및 채널코딩 기능을 가진 전력선 통신용 ASIC 구현" 전력전자학술대회 논문집 1 (2002): 4.

[13] 강영석. "전력선통신 동향과 상용화 전망" 한국전자파학회 전자파기술 15:4 (2004.10): 14.

[14] 권영괄. "전력선 모뎀을 이용한 선박엔진 성능분석기의 신호전달방식의 개선."

석사학위, 한국해양대학교, 2006.

[15] 하승우. "스마트그리드 구현을 위한 AMI 적용 사례 및 효과 분석에 관한 연구." 석사학위, 건국대학교 대학원, 2014.

[16] 한전KDN, "전력ICT원론 (Ver.3)", 한전KDN (2014.9): 6.

[17] 정준홍, 이정옥. "데이터집중장치의 효율적인 검침데이터 수집방법" 대한전기학회 학술대회 논문집 1 (2016.7): 2.

[18] 서충기. "경로 최적화 알고리즘을 적용한 전력선 통신 기반 저압원격검침 네트워크 관리 에이전트에 관한 연구." 박사학위, 인하공대, 2015.

[19] 김명완. "NMS 환경의 효율적 운영 검증을 위한 테스트 관리 시스템 구축." 석사학위, 홍익대학교, 2010.

[20] 이수정, 이현숙, 정휘석. "SNMP를 이용한 인터넷 장비(Internet Device)의 관리 기법" 한국정보과학회 학술발표 논문집 29 (n.d.): 3.

[21] [국내. 저자 없음] AoneNetwork_ IDC Service (2016.3), "SNMP 프로토콜의 구성과 네트웍관리", AoneNetwork_ IDC Service, http://aonenetworks.tistory.com/555, (2017-5-31 방문).

[22] 윤병수, 김채영. "SNMP 기반의 효율적인 데이터 수집 방식에 대한 연구" 대한전자공학회 학술대회 1 (2002): 4.

[23] 안병호, 김형천, 민체류, 김대영, 최재호, 류민우, "OID(Object Identifier)적용을 위한 응용 분야 연구", 한국인터넷진흥원 (2009.12): 5.

[24] [국내. 저자 있음] cobus, (2013.4), "MIB (Management Information Base)" 네이버블로그, http://m.blog.naver.com/cestlavie_01/40187742174, (2017-5-31 방문).

[25] ICT운영처, "저압 AMI용 Zigbee 통신설비", 한국전력공사 한전일반구매규격 (2017.1): 1.