2017 SEPTEMBER 28

[졸업과제 최종 보고서]

IoT 무선 네트워크기반 스마트팩토리 작업자 지원 서비스

<2 조> 유동화(201224484), 김나현(201424412), 김남호(201224413)

부산대학교 정보컴퓨터공학부

지도교수 : 정상화 교수

1 개요

1.1 목표

IoT 플랫폼 연동을 통한 작업자 접근 감지 및 설비 제어 시스템을 구축하여 스마트 팩토리에 적용한다.

1.2 선정 배경

1) 작업자 부주의에 의한 오작동 사고 방지

지난해 발생한 화학사고 발생 원인 2건 중 1건은 작업자 부주의에서 비롯된 것으로 나타났다. 화학 공장뿐만 아니라 발전소, 원전 같은 곳에서는 단순한 밸브 조작을 잘못해서 전체 설비를 정지시키고 심지어는 심각한 문제를 많이 발생했다. 아래는 중국의 원전에서 오조작으로 문제를 발생한 기사의 일부를 캡처한 것이다.

> 닝더 원전 직원이 실수로 배기굴뚝 벨브를 잘못 열어 방사선의 일종인 베타(β)선이 기준치를 초과하게 만든 사건, 광섬유케이블을 플러그를 잘못 뽑아서 컴퓨터 시스템과 원전통제시스템의 연결이 중단되게 만든 사건 등이 포함됐다.

그림. 1 작업자 부주의로 인한 문제 발생 관련 기사

이러한 상황에서 우리는 작업자가 실시간으로 지시사항을 인지해서 설비의 오조작을 최소한으로 방지하는 것이 중요하다는 것을 알게 되었다.

2) 스마트 팩토리 도입의 필요성

우리나라는 최근 몇 년 전부터 제조 효율성 저하, 생산 거점의 해외 이전, 인구 감소 및 노령화 등으로 '제조업 위기설'이 나오고 있다. 국내 생산 가능 인구는 2016년 3,704만 명을 정점으로 2017년 3,702만 명으로 줄어들면서 노동력 부족 현상이 심화될 것으로 예상된다. 이에 따라 정부는 제조업과 ICT의 융합을 통한 스마트 팩토리를 구성하여 제조업 경쟁력강화를 주도하고 있다.

스마트 팩토리는 제조업의 관련된 물품 조달, 물류, 소비자 등 다양한 엔티티가 존재하며, 각각의 객체는 지능을 가지고 IoT로 연결되어 자율적으로 데이터를 연결·수집·분석할 수 있다. 이는 공장 자동화와는 다른 개념으로 공장은 하나의 지능을 가지고 최소한의 인원으로 최대의 제조 효율을 낼 수 있다. 현재 한국의 스마트 팩토리는 초기 단계로, 현재는 핵심 부분에 적용 된다기 보다 환경 제어와 같이 영향이 적은 보조 분야부터 조금씩 실행되고 있다. '라이프 이즈 온 이노베이션 서밋'에서 홍승호 한양대 교수는 공장에도 IoT(Industrial Internet of Thing)를 도입이 스마트 팩토리 도입의 중요한 일부가 될 것이라고 했다. 이에 따라 우리는 IoT 통신으로 WSN을 구축하여 스마트 팩토리에 네트워크 인프라를 제공하도록 한다.

2 요구 사항

2.1 요구 조건 분석

1) 작업자 인증 및 접근 권환 관리

공장의 설비를 단말기를 가진 누구나 와서 조작하는 것은 상당한 리스크가 있다. 이를 방지하기 위해서 시스템은 사용자 인증을 거칠 수 있도록 한다. 작업자가 단말기를 사용하기 위해서는 id, password를 입력하여 접근 승인을 받아야한다. 또한, 공장의 존재하는 다양한 설비 중에서 실제 작업자가 제어할 수 있는 설비가 제한 될 수 있기 때문에 사용자 보안 수준을 적용하여, 설비의 보안 레벨보다 높은 보안레벨을 가진 사용자만 해당 설비의 접속하여 조회 및 조작을 실시할 수 있다.

2) 설비 상태 모니터링

작업자는 지정된 단말기를 통해 설비의 이름, 종류, 센서 정보, 지시사항 등의 정보를 지속적으로 확인 할 수 있어야한다. 이를 통해 설비의 현 상태 데이터를 감시하여 설비의 동작이 정상적으로 동작하는 지 확인한다.

3) 지시사항 전달

서버는 설비의 센서데이터를 지속적으로 조회하여 비정상적인 상황이 발생했을 때, 해당 상황에 가장 적적한 대응 지시 사항을 작업자에게 전달하여 상태를 조치를 취하게 한다. 또한 설비가 정상적으로 동작하는 경우에는 특정한 지시사항이 발생하지 않아 작업자는 편리하게 설비의 이상 유무를 판단할 수 있다.

4) 설비 원격 제어

작업자는 공장의 다양한 설비를 조작 할 수 있다. 기계 설비에 접근하여 사용자 인증을 받은 이후에, 조작이 가능한 기능을 작업자 단말기에서 즉시 조작할 수 있게 한다. 프로젝트에서는 모터의 동작을 제어하는 것으로 구현하였다.

5) 범용적으로 사용가능한 플랫폼화

본 프로젝트는 스마트 팩토리를 위한 시스템이기 때문에 이종의 공장에 적용함을 고려해야한다.

2.2 제약 사항

1) 다양한 센서 추가

TelosB 보드로 설비 노드가 센서의 데이터를 측정한다. 그러나 보드의 용량의 문제로 많은 센서의 데이터를 측정하고 이를 서버로 전송하는데 문제가 있다. 48K Byte의 플래시 메모리 용량과 10K Byte의 램용량으로는 OpenWSN에서 제공하는 프로토콜 스택 및 연결과 관련된 어플리케이션을 구동하는데에는 충분하지만 추가적으로 새로운 어플리케이션을 추가하거나 센서 구현을 위하여 bsp코드를 수정하는데 있어 용량에 대한 부담이 따른다. 따라서 추가적으로 여러 개의 센서를 구현하는 것보다 기본적인 TelosB 자체내에 내장된 센서를 활용하여 기본적인 기능을 구현함을 보인다.

2) 공장 규모에 따라 중계 노드 필요

IEEE 802.15.4 기반의 LLN(Low-Power & Lossy Network)의 특성상 통신 거리에 제약이 있다. 우리가 사용 중인 TelosB는 open-space에서 최대 통신거리가 25m이기 때문에, 규모가 큰 공장에서는 서버와 설비간 통신을 위해 여러 레벨의 중계 노드가 요구된다.

우리의 목표는 서버에서 작업자 단말기까지 중간 노드를 거쳐 통신이 가능함을 보이는 것이므로 중계 노드를 하나의 레벨만으로 구성하였다.

3 개발 환경

3.1 TelosB

TelosB는 Texas Instrument사에서 제작한 MSP430F1611 MCU와 CC2420 RF칩을 장착한 Wireless network 장비이다. 조도센서, 온 습도 센서, USB, 안테나를 기본으로 장착하고 있으며 총 8개의 센서 채널을 가지고 있다. IEEE 802.15.4 통신을 사용하며 2.4GHz에서 2.485GHz의 주파수 대역을 사용한다. 수신 감도는 최대 -95dBm이며 동작전원은 2.1V ~ 3.6V로 AA 건전지 2개나 USB 전원을 통하여 동작한다. 주로 연구나 실험 분야에서 널리 사용되며 합리적인 비용과 여러 센서를 더하여 사용할 수 있는 점, 사용의 편리함을 장점으로 가진다.

3.2 Raspberry Pi

영국 라즈베리 파이 재단에서 만든 초소형/초저가 PC이다. 교육용 프로젝트의 일환으로 개발되었다. LPDDR2 1GB의 RAM과 Broadcom BCM2836/2837 Soc, ARM Cortex-A53의 CPU로 구성되며 802.11n 무선 네트워크, 100Mbps의 이더넷 블루투스 4.1을 지원한다. 운영체제는 NOOBS와 RASBIAN이 공식적인 운영체제이며 Ubuntu mate, Windows 10 IOT core등의 다른 운영체제도 지원한다.

3.3 OpenWSN

OpenWSN은 UC 버클리 대학교에서 진행하는 프로젝트이다. 무선 센서 네트워크 및 IoT 구성에 필요한 CoAP 체계를 오픈소스 코드로 제공한다. OpenWSN은 IEEE 802.15.4e TSCH(Time Synchronized Channel Hopping)의 개념을 기반으로 하는 MAC 계층으로 구성된다. MAC 계층 위에서 저전력 손실 네트워크 스택은 IETF의 6TiSCH 관리 및 적응 계층(6top 프로토콜)이 있다. 스택은 6LowPAN, RPL, UDP, CoAP로 구현되어 개방형 표준을 통해 IPv6에서 원할 하게 엑세스 할 수 있다.

3.4 Node.js

Node.js는 자바스크립트 엔진 'V8'에서 동작하는 이벤트 처리 I/O 프레임워크다. 서버 환경에서 자바스크립트로 애플리케이션을 작성할 수 있도록 도와준다.

특징

- 비동기 I/O 처리 / 이벤트 위주 : Node.js 라이브러리의 모든 API는 비동기식으로 Non-blocking이다. 이는 조금 더 직관적인 사용을 제공할 수 있다
- 단일 쓰레드 / 확장성 : Node.js는 이벤트 루프와 함께 단일 쓰레드 모델을 사용하며, 이벤트 메커니즘은 서버가 멈추지않고 반응하도록 하여 서버의 확장성을 키운다.
- MIT License : 라이선스와 저작관 명시만 하면 자유롭게 사용이 가능하다.
- NPM(Node Package manager)은 Node.js 패키지/모듈 저장소를 제공하며, 패키지 설치 및 버전/호환성 관리를 할 수 있다. 이를 통해 수십만개의 Node.js 라이브러리를 사용할 수 있다.

본 프로젝트에 사용된 서버, 애플리케이션은 모두 Node.js에 기반한 프레임워크를 사용하고 있다. 아래는 핵심적으로 사용된 프레임워크 종류와 설명이다.

(ㄱ) Express

웹 및 모바일 애플리케이션을 위한 일련의 강력한 기능을 제공하는 간결하고 유연한 Node.js 웹 애플리케이션 프레임워크이다. HTTP 유틸리티 메소드와 미들웨어를 통해 쉽고 빠르게 API를 작성할 수 있다. 본 프로젝트에서는 주로 API를 제공하여 요청을 처리한다.

(ㄴ) passport

Node.js용 범용 인증 모듈이다. HTTP Basic Auth에서 HTTP digest auth, OAuth 등 다양한 인증 프로토콜을 지원하며, Facebook이나 Google등과의 연계된 SSO 인증을 포함한 140가지의 인증 모듈을 포함한다. 본 프로젝트에서는 Local Strategy를 사용하여 자체 인증을 처리하도록 하였다.

(ㄷ) Mongoose

MongoDB 기반 ODM(Object Data Mapping) Node.js 전용 라이브러리다. ODM은 데이터베이스와 객체지향 프로그래밍 언어 사이 호환되지 않는 데이터를 변화하는 프로그래밍 기법이다. 즉, MongoDB에 있는 데이터를 JavaScript 객체로 사용할 수 있다.

(ㄹ) express-queue

Express에서 동시간에 처리되는 요청의 개수를 큐를 이용해서 제한한다. 작업자 단말기에서 메인 서버로 요청을 보내기 위해, 작업자 단말은 socket 통신 전용 Node.js 서버를 로컬로 실행하고 있다. 이 로컬 서버는 python을 이용해서 socket을 사용하여 TelosB로 데이터를 전달한다.

(ㅁ) Python-shell

Node.js에서 Python 코드를 불러와서 실행시킨 후, 그 결과값을 반환한다. TelosB와의 socket 통신에 사용된다.

3.5 MongoDB

MongoDB는 C++로 작성된 오픈소스 문서지향(Document-Oriented) 적 Cross-platform 데이터베이스이며, 뛰어난 확장성과 성능을 갖추고있다. 인지도 있는 NoSQL이다.

- NoSQL

  Not Only SQL, 기존의 RDBMS의 한계를 극복하기 위해 만들어진 새로운 형태의 DB이다. 관계형 모델이 아니므로 고정된 스키마가 존재하지 않는다.

- Document-Oriented

  Document는 RDBMS의 레코드와 비슷한 개념이다. 데이터 구조는 아래처럼 한 개 이상의

Key-value pair로 구성된다.

```
{
        "_id": ObjectId("5099803df3f4948bd2f98391"),
        "username": "velopert",
        "name": { first: "M.J.", last: "Kim" }
}
```

- Schema-less

  RDBMS처럼 스키마가 없기 때문에 Key는 임의로 지정할 수 있으며, value는 문자열, 정수 뿐만 아니라 object, array도 포함할 수 있다.

본 프로젝트의 대상인 스마트 팩토리는 설비의 형태나 정보가 정해진 것이 아닌 다양한 데이터를 생산한다는 이유때문에, MongoDB를 사용하여 다양한 데이터를 저장할 수 있게 한다.

3.6 Ionic

Ionic은 하이브리드 앱을 만들어 주는 프레임워크다. Ionic은 AngularJS2, Cordova, TypeScript로 구성되어 있다. 작업자 애플리케이션을 Ionic으로 구현하였기 때문에 효율적인 개발을 할 수 있었다.

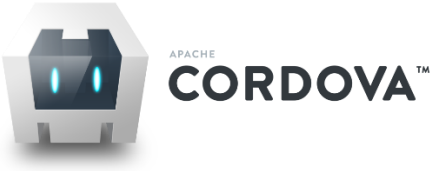
- AngularJS

  JavaScript 기반의 프론트엔드 웹 앱 프로게임워크이다. MVC(model-view-controller) 모델을 쉽게 구현할 수 있다.

- TypeScript

컴파일하면 JavaScript가 되는 언어이며, 컴파일 시점에 타입 체크를 하고 전통적인 객체 지향적 프로그래밍 패턴이 도입되어있다. 객체지향 프로그램이 가능하기 때문에 javascript 로만 개발하는 것보다 코드 재사용성이 증가하여 개발시간을 단축할 수 있다.

- Cordova



Apache 재단의 오픈 소스 모바일 개발 프레임워크다. Android, iOS 등 모바일 플랫폼의 네이티브 개발 언어만을 사용해야하는 것을 피하고 HTML5, CSS3, JavaScript 등 표준 웹 기술을 사용하여 크로스 플랫폼 개발을 할 수 있다. 이를 통해 본 프로젝트에서는 작업자 단말기가 라즈베리파이에서 뿐만 아니라 Android, iOS에서도 사용하는 경우를 고려했다.

작업자 애플리케이션을 Ionic으로 구현하였기 때문에 효율적인 개발 시간을 가지고 사용자에게 친숙한 UI를 제공 할 수 있다.

4 시스템 구조

4.1 시스템 네트워크 구성도

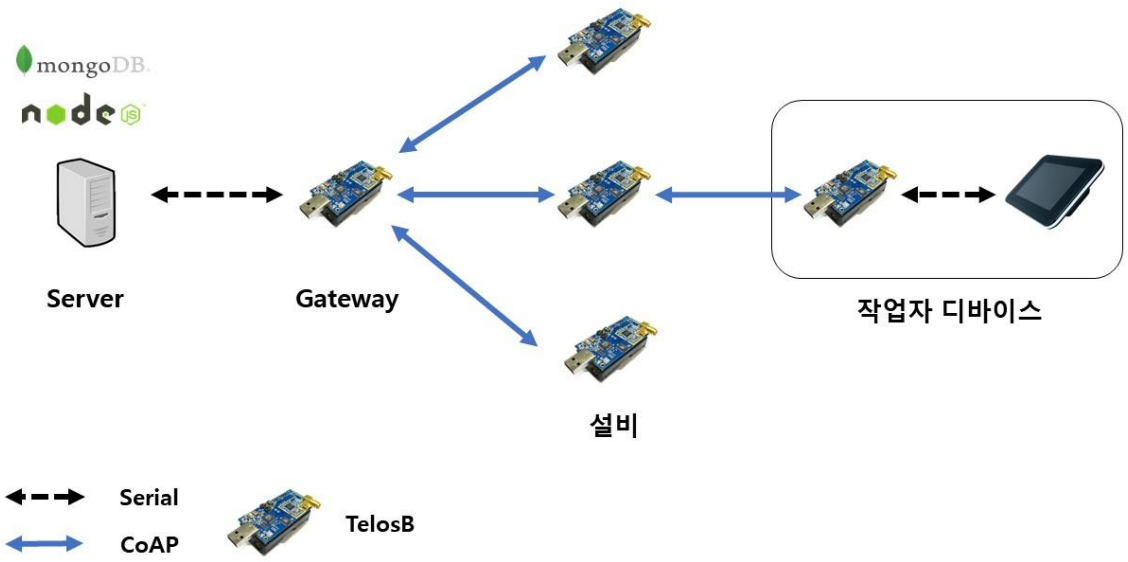


그림. 2 전체 시스템 구성도

전체 시스템은 작업자 디바이스와 설비 노드들, 게이트웨이 및 서버로 이루어 진다.

각 설비 노드들은 TelosB로 구성되며 내장되어 있는 센서를 이용하여 각 기계 설비에서의 정보를 수집한다. 수집된 데이터들은 OpenWSN 기반의 CoAP를 이용하여 루트 노드 역할을 수행하는 Gateway 및 서버로 전송된다. 작업자 디바이스는 모바일 노드로써 각 설비 노드와의 거리에 따라 자식 노드가 된다. 이는 네트워크의 신호세기를 비교하여 결정된다. 특정 설비 노드에 편입된 이후에 작업자 디바이스는 자신의 부모 노드에 대한 정보를 서버로 전송한다. 서버는 작업자의 로그인, 작업 지시사항, 설비의 센서 정보 요청을 전달 받아 처리된 데이터를 반환한다.

4.2 Server 구성도

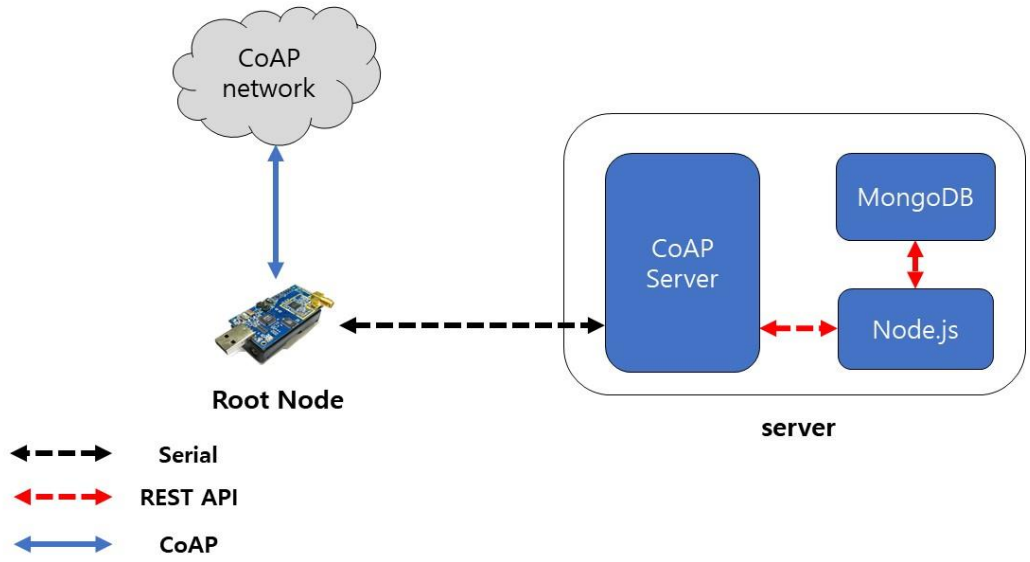


그림. 3 Server측 네트워크 구성

TelosB를 통해 받은 분해된 패킷들은 CoAP Server에서 조립되어 Web Application Server로 요청을 전달한다. 루트 노드와 리프 노드 간의 통신은 IEEE 802.15.4 기술에 기반한다. Web Application Server는 수신한 REST API에 대한 처리를 진행하고 결과를 반환한다. CoAP Server는 Web Application Server에서 반환된 결과를 패킷으로 분해하여 TelosB를 통해 발신지로 전송한다.

4.3 작업자 디바이스 구성도

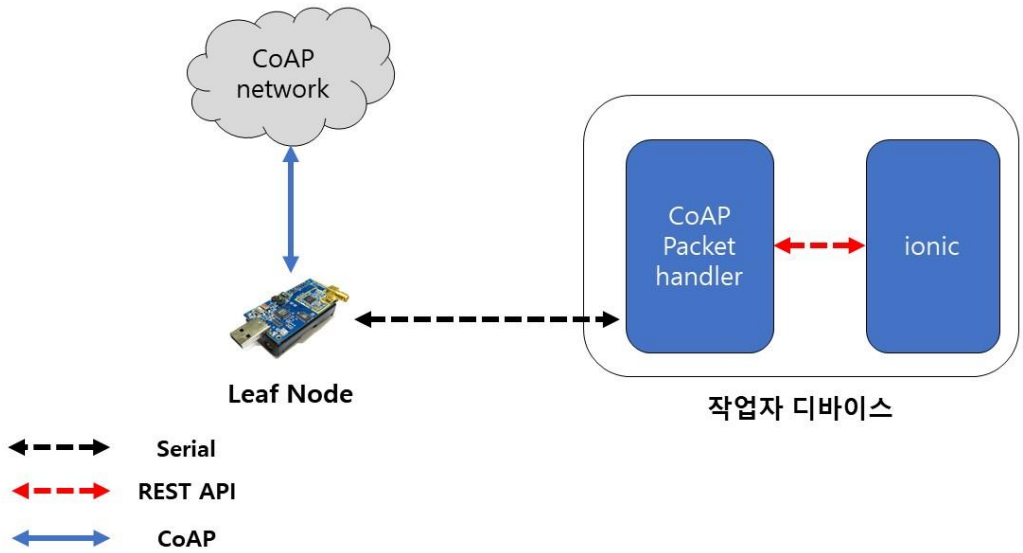


그림. 4 작업자 측 네트워크 구성

작업자 디바이스는 라즈베리파이, TelosB, 7-inch 터치 스크린으로 구성되어 있다. 작업자는 디바이스로 인접한 설비의 상태 모니터링, 제어권 획득, 동작 제어를 할 수 있다. 작업자 단말기가 루트 노드의 네트워크에 인접하면 루트 노드는 자동으로 작업자 노드를 자식 노드로 편입한다. 이를 통해 작업자 노드는 Server와의 통신이 가능하다.

Web Application은 설비 조작, 지시사항, 설비 정보 등을 REST API로 요청하고 응답 받는다.

5 개발 내용

5.1 펌웨어

1) 802.15.4 통신 topology

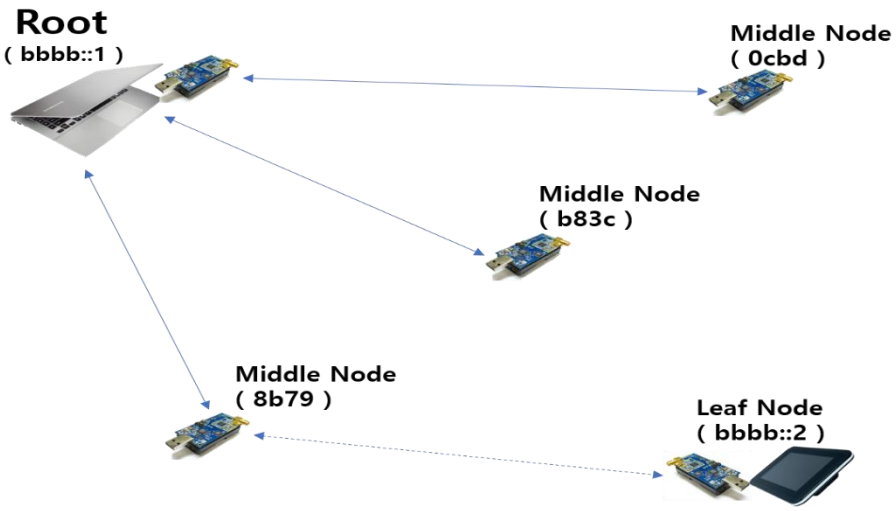


그림. 5 시스템 전체의 topology

Topology는 root node(server), middle node(공장 설비), leaf node(작업자 단말기) 총 3개의 level로 이루어져 있으며, 3대의 공장 설비와 한 명의 작업자가 있는 상황을 가정한다.

각 노드는 Rank (= (3*numTx/numTxAck)-2)*minHopRankIncrease)를 이용하여 거리가 가장 가까운 이웃 노드를 판별하게 되고, 이는 Tx와 Ack의 비를 통해 신뢰성을 계산한 수치이다. 신뢰성이 높을수록 가까운 노드라고 판별하게 되는 것이다. 가까운 노드를 판별하게 되면 IP주소를 이용하여 어떤 노드인지 확인하고 middle node는 root node를 부모로, leaf node는 middle node만 부모로 결정하도록 topology를 강제한다.

Root node와 middle node는 고정된 상황이고 leaf node는 이동하면서 가장 가까운 middle node를 판별하여 달라붙게 된다. 이러한 hand-over는 다음과 같이 구현하였다. TelosB는 리셋 버튼(빨간 버튼)과 프로그램이 가능한 유저 인터럽트 버튼(흰색 버튼)이 있다. 흰색 버튼을 이용하여 preferred-parent를 초기화하는 인터럽트를 발생 시키고 다시 부모를 찾는 과정을 반복한

다.

2) RPL (IPv6 Routing Protocol for Low Power Lossy Networks)

DODAG(Destination Oriented Directed Acyclic Graph)를 구성하기 위해서 root node와 leaf node는 DIO(Directed Information Object)와 DAO(Destination Advertisement Option)를 서로 주고 받는다. DIO는 root node에서 leaf node 방향으로 broadcast 하는 정보로서, 그래프 전체에 대한 정보를 담고 있다. 이를 통해 leaf node는 상위 노드들에 대한 정보를 얻게 된다. DAO는 leaf node 에서 root node 방향으로 올라가는 정보로서, 출발지에서 도착지까지의 경로 정보를 담고 있다. 이를 통해 root node는 해당 leaf node가 어떤 설비의 자식 노드로 편입되었는지를 알 수 있다. 모든 노드는 이러한 DIO와 DAO를 생성하여 전송하지만, leaf node는 최하위 node로 만들기 위해 DIO는 생성하지 않도록 설정하였다.

middle node의 RAM 용량 부족 현상을 해결하기 위해 DIO주기를 조절하였다. leaf node가 연결 되지 않았으면 빠른 연결을 위해 주기를 10초로 짧게 하고, 연결 되면 DODAG의 갱신 정보가 없을 것이므로 주기를 30초로 길게 설정하였다.

5.2 센서 및 GPIO

1) 조도 센서

TelosB 보드에 내장되어 있는 S1087 센서와 S1087-01 센서를 이용하여 각각 560nm, 960nm 의 파장까지의 조도를 측정한다. 가시광선을 발산하는 일반 LED와 적외선을 발산하는 IR LED를 이용하여 공장에서의 설비의 센서 값을 묘사하며 이를 조도센서들을 이용하여 측정한다.

2) GIO

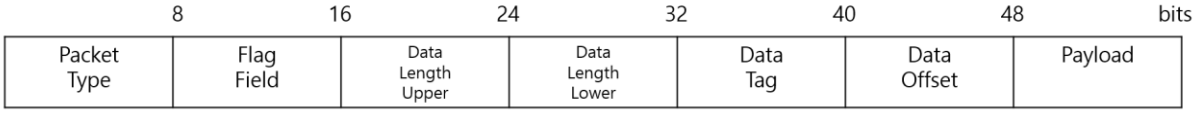
모바일 노드의 이동 후 자식 노드로 편입된 이후 라우팅이 완료 되었다는 사실을 알리기 위

하여 TelosB 보드의 GIO를 이용하여 LED를 점등한다. MSP430F1611 MCU의 포트 6.2를 이용하여 LED를 제어한다. 또한 설비에 부착된 모터를 제어하기 위하여 LED와 동일한 방법으로 MCU의 포트 6.3을 이용하여 작업자가 단말기를 통하여 모터를 제어 할 수 있는 기능을 제공한다.

5.3 패킷 단편화

1) 사용자 정의 헤더

사용자 정의 헤더의 구성은 아래의 그림과 같다.



사용자 정의 헤더의 처음에 위치하는 패킷 타입 부분은 모든 패킷에 공통적으로 포함되어 있지만 뒤의 부분들은 패킷 타입의 종류에 따라 존재 여부가 결정된다.

가) Packet Type

사용자 정의 헤더의 처음에 위치하는 패킷 타입 부분이다. 현재 자신의 payload에 어떤 종류의 데이터를 담고 있는지를 표시한다. 5종류의 패킷 타입이 존재하면 각각에 관한 설명은 다음과 같다.

Connection	작업자 디바이스와 서버측 미들웨어 간의 연결 설정에 관한 패킷
Login	작업자의 로그인에 관한 패킷
Sensor	설비 노드들에서 주기적으로 측정된 설비의 센서 데이터 패킷
Machine	설비의 정보, 설비에서 측정된 센서 데이터, 설비의 조작에 관한 패킷
Instruction	작업자 지시사항에 관한 패킷 임을 의미한다.

나) Flag Field

패킷 타입에 따라 다른 그룹의 Flag를 담고 있으며 경우에 따라 본 영역을 가지지 않은 패킷도 존재한다. Flag 그룹은 연결 설정 그룹과 단편화 그룹으로 이루어져 있으며 각 그룹에 관한 설명은 다음과 같다.

① 연결 설정 Flag

SYN(64)	0	1	0	0	0	0	0	0
ACK(32)	0	0	1	0	0	0	0	0
FIN(127)	0	1	1	1	1	1	1	1
SYN+ACK(96)	0	1	1	0	0	0	0	0

연결에 관련된 Flag는 SYN, FIN, ACK, SYN+ACK 가 있다. 연결 설정 및 해제의 과정에서 서버 측 미들웨어와 작업자 디바이스 측 미들웨어는 Flag의 내용을 보고 현재 상태를 확인하며 연결에 관련된 변수의 상태를 변화시킨다.

② 단편화 Flag

FRAG_1(24)	0	0	0	1	1	0	0	0
FRAG_N(28)	0	0	1	1	1	1	0	0

첫 번째 단편화 패킷을 나타내는 FRAG_1과 N번째 단편화 패킷을 나타내는 FRAG_N이 있다. 단편화 된 패킷을 주고 받을 시에 현재 패킷이 단편화의 처음인지 아닌지를 구분하는데 사용된다.

다) Data Length Upper

단편화 된 패킷의 크기 중 256보다 큰 값의 비트가 저장된다.

라) Data Length Lower

단편화 된 패킷의 크기 중 256보다 작은 값의 비트가 저장된다.

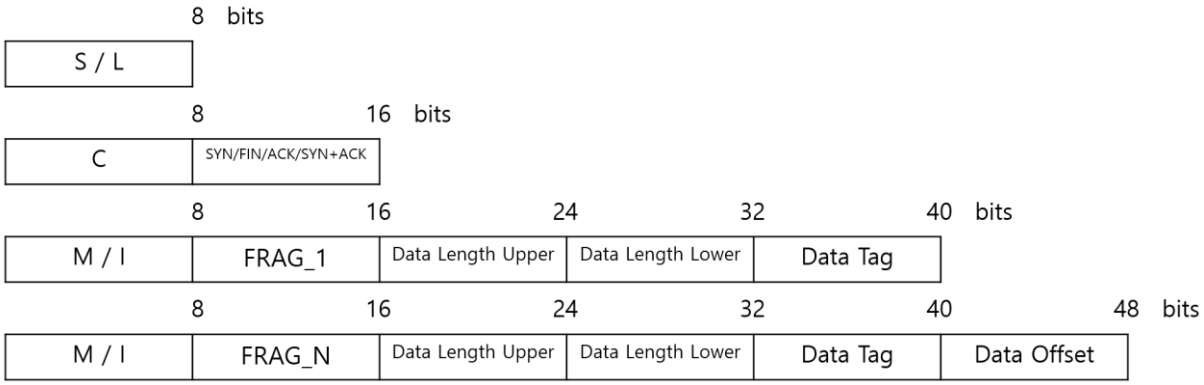
마) Data Tag

단편화 된 패킷의 그룹을 구분한다. 현재 수신 받은 패킷이 모두 하나의 패킷이 단편화 된 것인지를 검사하는데 사용된다.

바) Data Offset

단편화 된 패킷의 순서를 구분한다. 단편화 되어 받은 패킷을 순서대로 다시 복구할 때 Offset 영역을 보고 순서를 판별한다. Flag 영역이 FRAG_1, 즉 첫번째 단편화 패킷인 경우 본 영역은 생략되며 N번째 단편화 패킷에만 존재한다. 최소 1에서 최대 255의 값을 가질 수 있다.

2) 각 패킷 타입 별 사용자 정의 헤더 형식



가) 패킷 타입이 센서(S)이거나 로그인(L)인 경우 뒤의 추가적인 헤더 없이 패킷 타입 이후 바로 데이터가 담긴 payload가 위치한다.

나) 패킷 타입이 연결 설정(C)인 경우 뒤에 연결 설정 Flag 그룹이 위치하며 payload는 존재하지 않는다.

다) 패킷 타입이 설비(M)이거나 지시사항(I)이며 첫번째 단편화 패킷인 경우 단편화 Flag 그룹에 FRAG_1 Flag가 위치하며 Data Offset 필드를 제외하고 payload가 위치한다.

패킷 타입이 설비(M)이거나 지시사항(I)이며 첫번째 단편화 패킷인 경우 단편화 Flag 그룹에

FRAG_N Flag가 위치하며 Data Offset를 포함한 6 Byte의 전체 사용자 정의 헤더가 존재하며 후에 payload가 위치한다.

5.4 미들웨어

1) Server-side Middleware

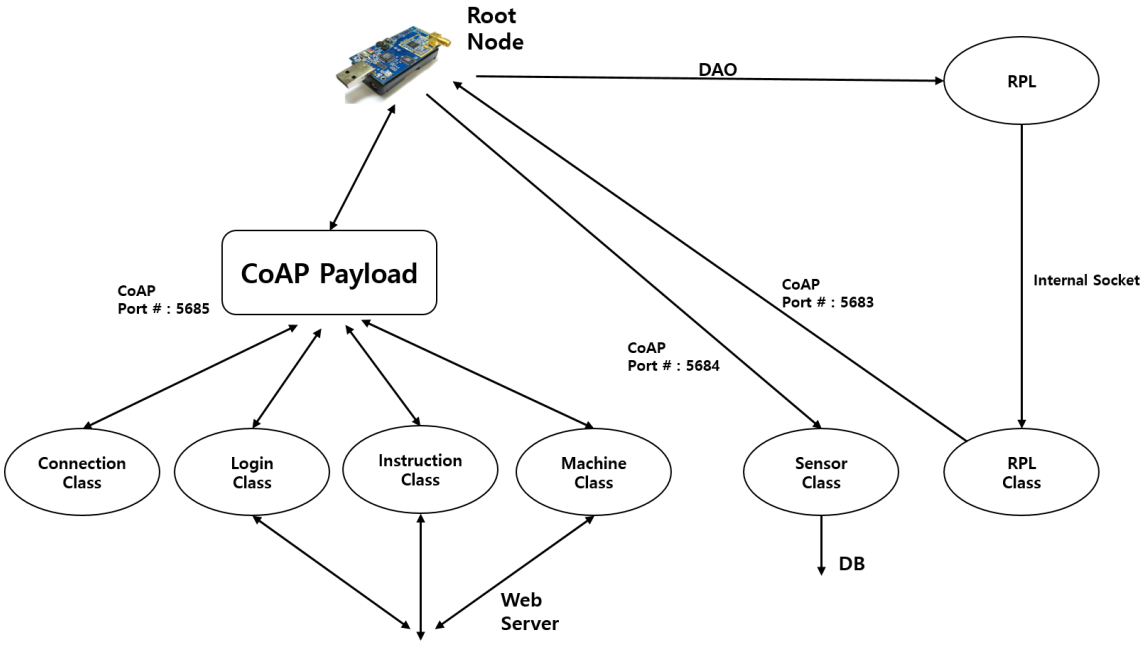


그림. 6 Server-side Middleware 프로그램 구성도

서버에서 동작하는 프로그램으로 작업자 디바이스와의 통신과 설비 노드들 과의 통신을 담당한다. 루트 노드로부터 받은 CoAP 패킷을 사용자 정의 헤더에 따라 처리를 할 클래스를 분할하여 패킷을 처리한다. 클래스의 종류로는 센서, 로그인, 설비, 연결, 지시사항, RPL로 나뉜다.

센서 클래스의 경우 단독적으로 5684번 포트를 이용하여 센서 데이터를 수신 받는다. 수신 받은 데이터는 JSON형식으로 변환되어 DB에 저장된다. 센서 데이터는 각 설비 노드들에서 CoAP PUT 메소드를 이용하여 루트 노드의 주소인 'bbbb::1', 5684번 포트를 목적지로 전송된다.

로그인 클래스의 경우 사용자 정의 헤더 패킷 타입이 'L'인 패킷들을 처리한다. 작업자 디바이스로부터 오는 작업자의 로그인 요청을 통해 ID와 PW로 분리된 데이터를 받는다. 해당 데이터들

을 이용하여 웹서버에 요청을 보내 처리하며 로그인 결과를 다시 작업자 디바이스로 전송한다.

설비 클래스의 경우 사용자 정의 헤더의 패킷 타입이 'M'인 패킷들을 처리한다. 작업자 디바이스로부터 3가지 종류의 요청 패킷을 받고 해당 요청을 처리한다. 3가지 요청들은 설비 정보, 설비 센서 정보, 설비 모터 조작이다.

가) 첫번째로 설비 정보에 대한 요청이다. 현재 작업자가 연결되어 있는 설비 노드의 정보를 서버로부터 전달 받아 해당 데이터의 길이에 따라 단편화를 이용하여 작업자 디바이스로 전송한다.

나) 두번째로 설비 센서 정보에 대한 요청이다. 첫번째와 마찬가지로 작업자가 연결되어 있는 설비 노드에서 측정되고 있는 센서 정보를 서버로부터 전달받는다. 이를 데이터의 길이에 따라 단편화를 이용하여 작업자 디바이스로 전송한다.

다) 마지막으로 설비 모터 조작에 대한 요청이다. 작업자 디바이스로부터 모터 조작에 대한 명령을 받아 해당 설비에 CoAP를 이용하여 조작을 수행한다. 그리고 수행 결과를 다시 작업자 디바이스로 전송하여 준다.

연결 클래스의 경우 작업자 디바이스와 서버측 미들웨어 간의 연결을 관리한다. 사용자 정의 헤더의 패킷 타입이 'C'인 패킷들이 이에 해당하며 이후의 플래그 영역에 SYN(64), FIN(127), ACK(32)의 값을 담고 있다. 연결 클래스의 동작은 크게 연결 설정 부분과 연결 해제 부분으로 나뉜다.

가) 연결 설정의 경우 작업자 디바이스로부터 SYN 플래그를 받으면서부터 시작된다. 이에 SYN+ACK로 답장을 전달하며 다시 작업자 디바이스로부터 ACK를 전달받아 해당 작업자 디바이스와의 연결 상태를 최종적으로 연결 되었다고 정의한다.

나) 연결 해제의 경우 연결 설정의 과정과 비슷하게 동작한다. 작업자 디바이스로부터 FIN 플래그를 받으면 FIN을 담아 답장을 보낸다. 최종적으로 작업자 디바이스로부터 ACK를 전달 받으며 연결 상태를 끊어진 것으로 정의하며 다음 연결 설정 요청에 대비하여 저장하고 있던 변수들을 초기화한다. 또한 작업자 디바이스가 이동하는 것으

로 판단하여 연결되어 있던 설비 노드의 DIO 발신을 RPL 클래스를 통하여 조절한다.

지시사항 클래스의 경우 사용자 정의 헤더의 패킷 타입이 'I'인 패킷들을 처리한다. 작업자 디바이스로부터 지시사항 전송을 요청 받고 해당하는 지시사항을 전송한다. 작업자의 ID와 설비의 ID를 이용하여 해당하는 매뉴얼을 웹 서버로부터 전송 받는다. 해당 데이터를 길이에 따라 단편화를 진행하여 작업자 디바이스로 전송한다.

RPL 클래스의 경우 Openvisualizer로부터 오는 라우팅 정보 또는 연결 클래스의 연결 해제 과정의 요청에 대하여 DIO 송신 주기를 조절하는 패킷을 각 노드에 전송한다. 설비 노드에 대해서는 DIO를 조절하는 패킷을 전송한다. 이때 DIO의 발신 주기를 보고 연결을 대기하는 과정인 경우 설비에 부착된 녹색 LED를 점등 시키며 반대로 연결을 해제하는 과정인 경우는 녹색 LED를 소등한다.

2) Client-side Middleware

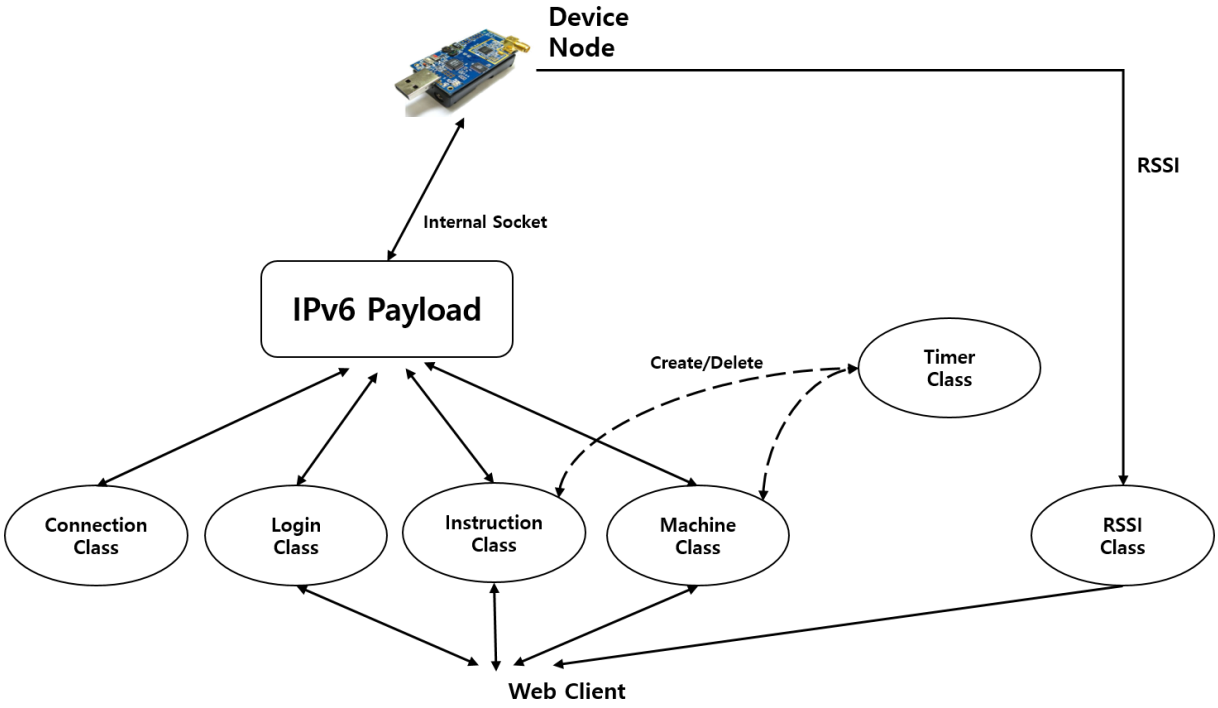


그림. 7 Client-side Middleware 프로그램 구성도

작업자 디바이스에서 동작하는 프로그램으로서 작업자 디바이스의 웹 페이지와 서버의 통신을 담당한다. 웹 클라이언트로부터 받은 요청을 각각의 종류에 따라 클래스 별로 나뉘어 서버가 위치한 루트 노드로 전송한다. 또한 루트 노드로부터 전송되어 오는 패킷을 작업자 디바이스로 전달받은 후 해당 패킷 데이터들을 처리한다. 패킷의 종류로는 로그인, 설비, 연결, 지시사항로 나뉘며 이를 각각의 분리된 클래스로 처리한다. 또한 작업자 디바이스와 설비 노드 간의 신호 세기 감쇠 정보를 지속적으로 업데이트 하는 RSSI 클래스를 가진다.

로그인 클래스의 경우 작업자의 로그인 요청을 받아 처리한다. 웹 클라이언트로부터 아이디와 비밀번호 정보를 받아 이를 서버로 전송한다. 이후 서버로부터 로그인에 대한 결과를 전송 받아 작업자의 아이디와 권한 레벨 정보를 다시 웹 클라이언트에 전달한다.

설비 클래스의 경우 작업자의 설비에 관한 정보 요청이나 제거 요청을 받아 처리한다. 웹 클라이언트로부터 3가지 요청 메시지를 받으며 서버로 해당 요청에 대한 패킷을 전송한다. 3

가지 요청은 설비 정보, 설비 센서 정보, 설비 모터 조작이다. 또한 서버의 처리 결과를 받아 처리하는데 단편화가 되어 오는 경우 다시 패킷을 완전히 조립하여 웹 클라이언트에 전송한다.

연결 클래스의 경우 서버측 미들웨어와 본 프로그램 간의 연결을 관리한다. 웹 클라이언트로부터 연결 요청을 받아 SYN 플래그를 전송하거나 연결 해제 요청을 받아 FIN 플래그를 전송한다. 이를 통해 서버측 미들웨어로부터 상응하는 답장을 받고 ACK를 보내줌으로써 연결 설정 또는 해제를 완료한다.

가) 연결 설정의 경우 웹 클라이언트로부터 오는 연결 설정 요청에 대하여 SYN 메시지를 서버로 보내면서 시작된다. SYN을 전송한 이후 서버로부터 SYN+ACK 메시지를 기다리며 받은 경우 ACK로 답장하며 서버와의 연결 상태를 최종적으로 연결되었다고 정의한다.

나) 연결 해제의 경우 웹 클라이언트로부터 오는 연결 해제 요청에 대하여 FIN 메시지를 서버로 보내면서 시작된다. 연결 설정의 경우와 유사하게 FIN을 전송한 이후 서버로부터 FIN 메시지를 기다리며 받은 경우 ACK로 답장하며 서버와의 연결 상태를 최종적으로 해제되었다고 정의한다. 연결 해제가 정상적으로 이루어진 경우 다음 연결 설정 요청에 대비하여 사용되었던 변수들을 모두 초기화 하는 과정을 거친다.

지시사항 클래스의 경우 웹 클라이언트의 지시사항 요청 정보를 받아 처리한다. 또한 서 서버부터 전송되어 오는 요청 결과들을 처리하는데 단편화가 일어나서 오는 경우 패킷을 완전히 재조립한후 웹 클라이언트로 내용을 전달한다.

RSSI 클래스의 경우 작업자 디바이스와 인접한 설비 노드 간의 신호 수신 감도 정보를 지속적으로 업데이트 한다. 웹 클라이언트는 일정한 주기로 RSSI에 관한 정보를 요청하며 이 요청을 클라이언트 미들웨어는 보유하고 있던 RSSI 데이터를 돌려준다.

타이머 클래스의 경우 단편화가 일어나서 도착하는 패킷들에 대하여 재조립시간에 대한

타임아웃을 담당한다. 타임아웃의 시간은 60초이며 단편화된 패킷의 수신을 담당하는 클래스 내에서 첫 번째 단편화된 패킷을 받으면 타이머를 실행한다. 지정된 60초 내에 패킷이 재조립이 완료되면 타이머를 종료하고 사용된 변수들을 초기화 하며 그렇지 않은 경우에는 이전까지 받은 데이터들을 모두 초기화 하여 웹 클라이언트가 다시 요청을 보낼 경우 정상적으로 처리 할 수 있도록 변수들을 초기화 한다.

5.5 패킷 Header Switching and Forwarding

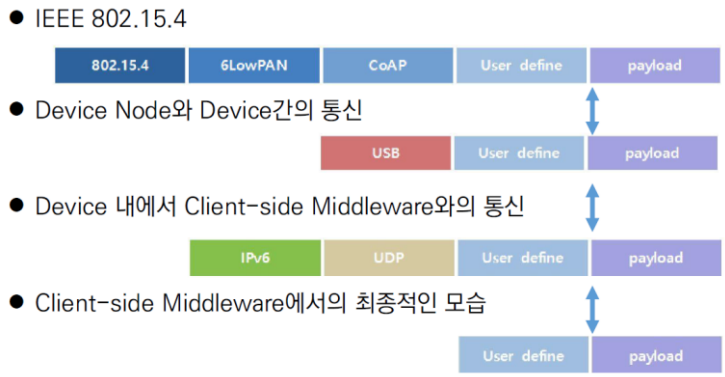


그림. 8 패킷 Header Switching & Forwarding

1) 서버 -> 작업자 디바이스

기본적으로 오픈소스에서 구현된 기능에서 CoAP 중 PUT 메소드를 이용하여 서버로부터 노드에 원하는 데이터를 송신 할 수 있는 기능이 있다. 이를 활용하면 서버측 미들웨어에서 CoAP 메소드를 활용하여 패킷을 전송하면, 모트의 펌웨어의 CoAP 계층에서 수신 받은 패킷 중 payload만을 얻어 낼 수 있다.

이러한 payload에 USB 시리얼 헤더를 붙여 작업자 디바이스에서 구동되고 있는 Openvisualizer로 데이터를 전송한다. 기존에 존재하는 Dispatcher와 Handler는 이러한 방식으로 전달되는 데이터를 처리하는 방식이 구현되어 있지 않으므로 Dispatcher와 Handler를 새로이 추가하여 USB 시리얼 헤더를 구분하여 제거하고 Handler에서 해당 데이터를 처리한다.

Handler를 통하여 받은 payload를 Ipv6 소켓을 이용하여 작업자 디바이스에서 구동되고 있는 Client-side 미들웨어로 전송한다.

최종적으로 Client-side 미들웨어에서 해당 패킷을 처리하여 웹 클라이언트로 데이터를 전달함으로써 서버 측에서 작업자 디바이스로의 송신이 이루어진다.

2) 작업자 디바이스 -> 서버

라즈베리 파이와 노드는 USB 시리얼 통신으로 이어져 있으므로 DODAG에 대한 라우팅 정보를 보유하고 있지 않다. 따라서 라우팅 정보가 존재하는 Leaf 노드로 전송하여 해당 데이터를 최종적으로 서버로 전송하여야 한다.

Client-side 미들웨어는 작업자 디바이스의 웹 어플리케이션으로부터 받은 요청을 사용자 정의 헤더를 붙여 IPv6 소켓을 이용하여 서버의 주소인 bbbb::1로 전송한다. OpenTun의 Dispatcher로 Openvisualizer의 Handler로 전송된다.

Openvisuzlier에서 IPv6 소켓을 이용하여 받은 payload에 Leaf 노드로 전송하기 위하여 USB 시리얼 헤더를 부착한다. 이 패킷은 Leaf 노드 펌웨어에서 수신되며 USB 시리얼 헤더를 제거하여 다시 payload를 얻어낸다. 이를 CoAP의 PUT 메소드를 이용하여 서버로 전송하기 위하여 펌웨어의 CoAP 송신 어플리케이션으로 전달한다.

최종적으로 CoAP 패킷은 서버로 전달되며 서버측 미들웨어가 수신하여 이의 payload를 보고 웹 서버에 요청을 보내어 처리한다.

5.6 WAS(Web Application Server)

1) Node.js

WAS로 Node.js를 이용하며 데이터 교환은 REST API 방식으로 이뤄진다

- /api/auth/register [POST] → (id, password, access level) 사용자 등록

- /api/auth/login [POST] → (id, password) 사용자 로그인
- /api/machines/info[GET] → (workerID) 설비 기본 정보

Request : http://localhost:8088/api/machines/info?workerID=0002

Response

Response(8ms)

```
HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=utf-8
Content-Length: 67
ETag: W/"43-dbeVnaYh9NqbyJlB9EgJlcLNSGM"
Date: Fri, 22 Sep 2017 07:47:52 GMT
Connection: keep-alive

[
  {
    "machineID": "b83c",
    "name": "machine Number 01",
    "accessLevel": "2"
  }
]
```

그림. 9 설비 정보를 가지고 모습

- /api/machines/sensor[GET] → (workerID) 설비 센서 정보

Request : http://localhost:8088/api/machines/sensor?workerID=0002

Response

Response(7ms)

```
1  HTTP/1.1 200 OK
2  X-Powered-By: Express
3  Access-Control-Allow-Origin: *
4  Content-Type: application/json; charset=utf-8
5  Content-Length: 55
6  ETag: W/"37-sT/WpK6Skf+Lwglv3js/g3+L84s"
7  Date: Fri, 22 Sep 2017 07:54:36 GMT
8  Connection: keep-alive
9
10 [
11   {
12     "sensorState": {
13       "photosynthetic": "35",
14       "solar": "370"
15     }
16   }
17 ]
```

그림. 10 설비 센서 정보를 가지고 온 모습

- /api/machines/manual[GET] → (workerID) 설비 지시사항

Request : http://localhost:8088/api/machines/manual?workerID=0002

Response

Response(22ms)

```
HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=utf-8
Content-Length: 177
ETag: W/"b1-XGzAgNmLlOeUa/GpJOBYjOmxtpk"
Date: Fri, 22 Sep 2017 07:58:02 GMT
Connection: keep-alive

[
  {
    "manual": [
      {
        "instruction": [
          "solar가 너무 높다 1",
          "solar가 너무 높다 2",
          "solar가 너무 높다 3",
          "solar가 너무 높다 4",
          "solar가 너무 높다 5"
        ],
        "num": "1"
      }
    ]
  }
]
```

그림. 11 설비 지지사항을 가지고 온 모습

2) MongoDB

A. User Collection (사용자 정보)

- ID 유저 ID
- Password 유저 비밀번호
- accessLevel 접근 권한 수준 (1-10, 숫자가 높을수록 낮은 권한)
- updatedAt 수정 일자

- createdAt 생성 일자

B. Machines Collection (설비 정보)

- machineID 설비 ID
- name 설비 명칭
- accessLevel 설비를 조회,조작할 수 있는 접근 수준
- nearWorkerID 설비에 연결한 작업자 ID
- sensorState 최신 센서 값 Object
- manual 설비에서 사용되는 지시사항 Array

C. sensorDB collection (설비의 센서 값 보관)

- machineID 설비 ID
- sensorRecord 이전까지의 센서 값 Array
  - data 센서 값
  - sensor 센서 이름
  - time 시간

5.7 작업자 애플리케이션

1) 애플리케이션과 서버 간의 통신 과정

작업자 애플리케이션과 서버 간의 데이터 교환은 아래와 같이 이뤄진다.

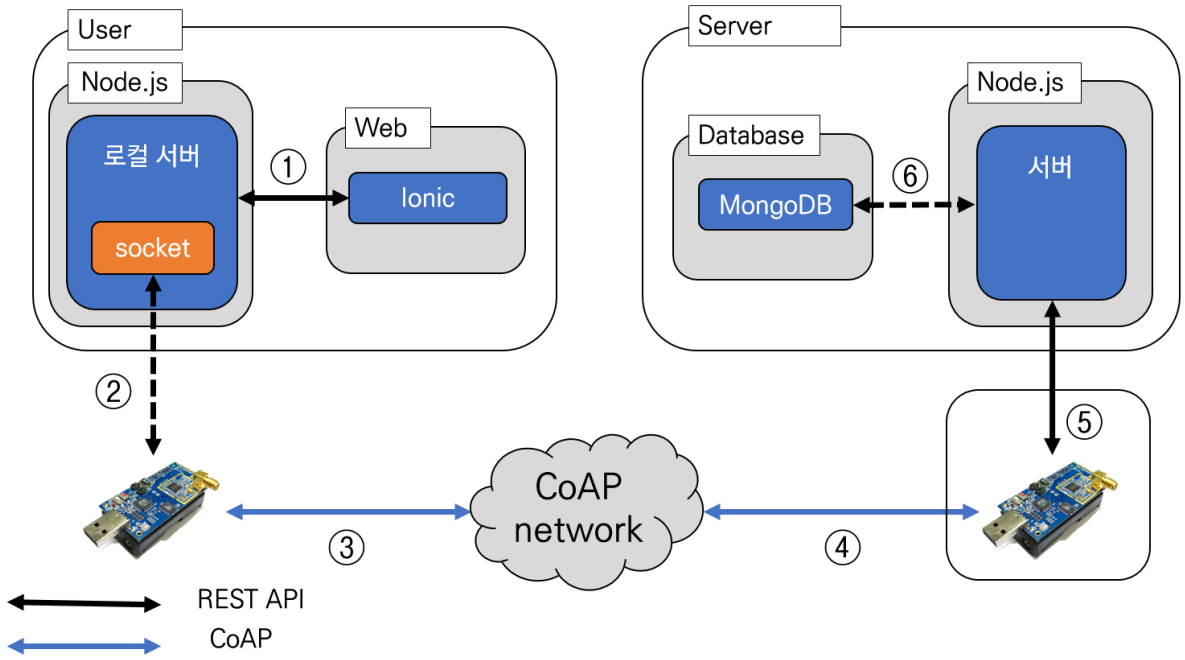


그림. 12 애플리케이션 수준 통신 구조

① 웹 애플리케이션은 작업자의 요청을 REST API로 로컬서버로 전달한다.

② 로컬 서버는 애플리케이션으로부터 받은 REST API를 socket을 이용하여 IoT 네트워크로 전달한다.

③ REST API 요청을 CoAP 패킷으로 서버 측으로 전송한다.

④ 서버 측 TelosB는 패킷을 조립하여 REST API로 생성한다.

⑤ REST API를 이용하여 서버로 요청을 전송한다.

⑥ 서버는 Database에서 필요한 데이터를 추출한다.

⑦ 작업자 REST API에 대한 응답은 요청 작업의 역순으로 진행된다.

6 애로 사항

6.1 Hand-over

자식 노드는 Rank(Tx에 대한 신뢰도)를 이용하여 부모 노드를 판별한다. 우리는 RSSI(Received

Signal Strength Indication) 값에 따라 부모 노드를 판별하길 원했고, 기존의 Rank식에 RSSI값을 반영하도록 식을 재구성했지만, hand-over하는 과정에서 Rank 값이 변경되었음에도 부모를 바꾸지 않거나 아예 작동이 멈춰버리는 등 오작동이 빈번하였다. 따라서 TelosB 버튼을 통해 인터럽트를 발생시켜 preferred-parent를 초기화하고 다시 부모를 찾는 방법으로 변경하였다.

6.2 RAM용량 부족

TelosB의 RAM용량은 10KB로, 우리가 초기에 계획했던 내장 온,습도 센서와 다양한 GIO를 구현하기에는 다소 부족하였다. RAM용량을 확보하기 위해 기존에 구현 되어있는 led 조작, 보드 스펙 리턴 기능을 삭제했고, 패킷 전송할 때 사용하는 오픈 큐와 neighbor 저장 구조체는 해당 시나리오에서 필요한 양으로 줄였다. 또한 모터와 다양한 센서를 연결하는 대신 led 두 개만 연결하여 설비가 동작하는 상황을 가정하기로 하였다.

7 결과

7.1 애플리케이션 결과

1) 연결 관리

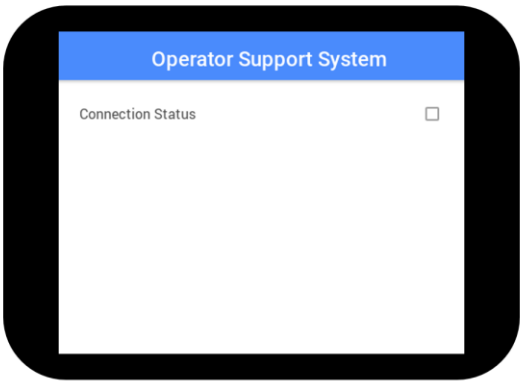


그림. 13 시작 화면

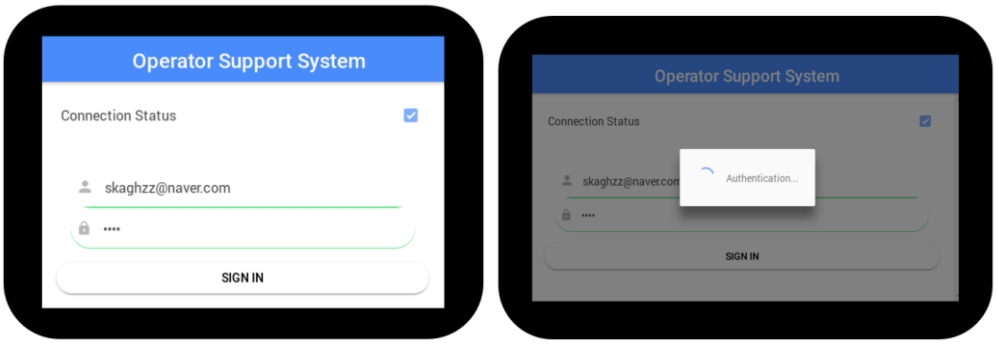


그림. 14 로그인 진행 화면

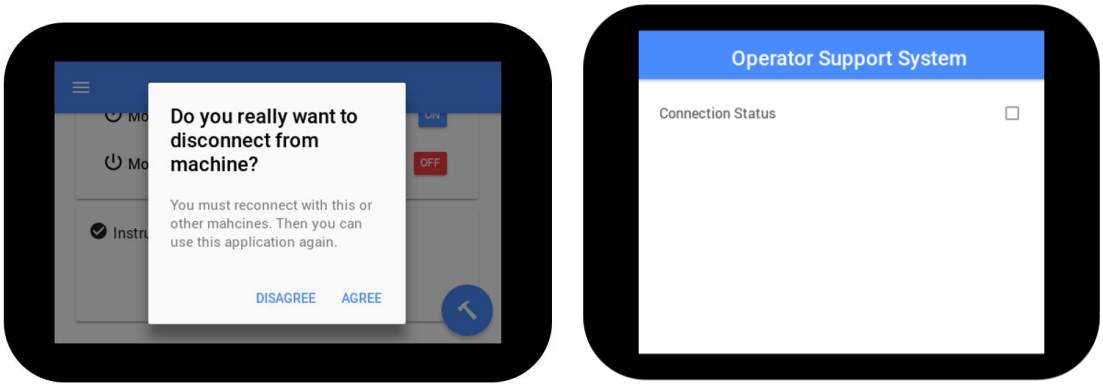


그림. 15 요청 해제 화면

2) 정보 조회

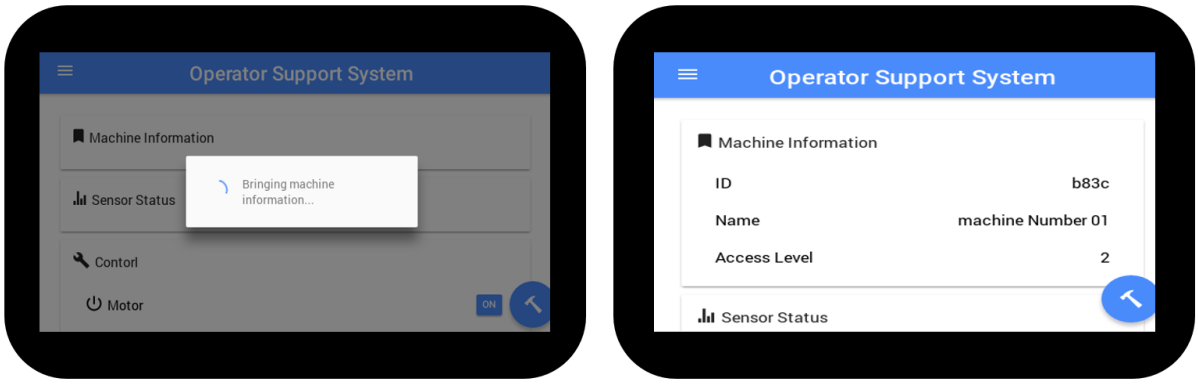


그림. 16 설비 기본 정보

그림. 17 센서 정보 화면

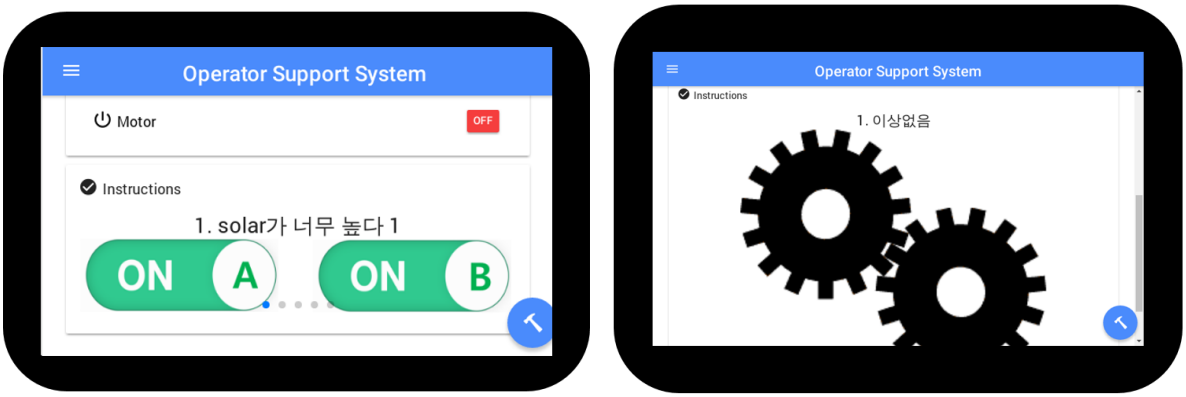


그림. 18 지시사항을 불러온 화면

3) 기기 조작

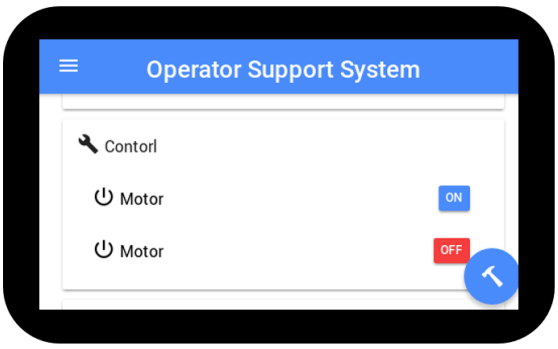


그림. 19 기기 조작 화면

4) 추가 기능

가) RSSI 값 확인해서 연결 해지 주의

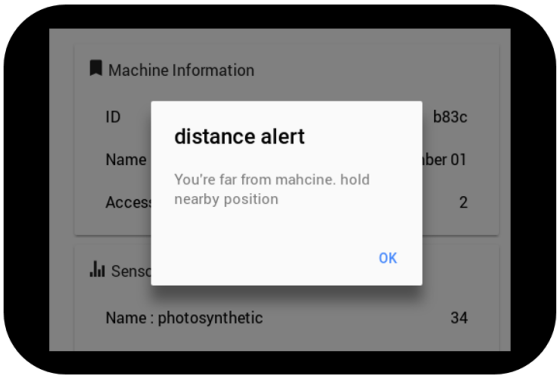


그림. 20 거리값 경고 화면

나) 로그인 시 Access Level에 따른 접근 제어

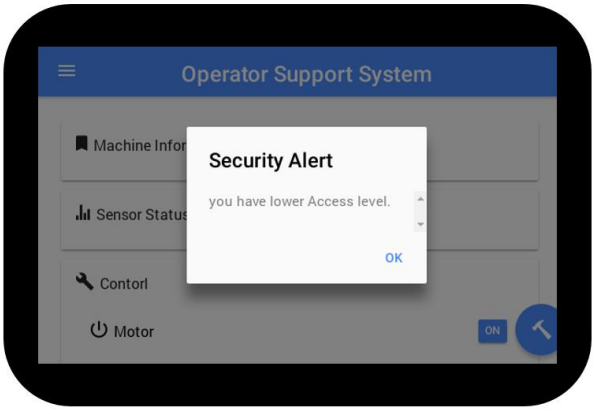


그림. 21 보안 수준화면

8 기대 효과

8.1 非스마트팩토리에 적용하여 설비 오작동 사고 방지

작업자는 공장 순찰시에 단말기를 지니고 다닌다. 단말기로 애플리케이션을 통해 근접한 설비의 센서 현황을 조회할 수 있다. 이를 통해 설비의 현 상태를 파악하여 예상치 못한 상황에 대응할 수 있다. 또한, 시스템은 자동으로 센서 등의 정보에서 문제점을 인식하고 적절한 대응방법을

작업자에게 전송하여 작업자의 수행의 신뢰성을 높여 설비 오작동을 방지한다.

8.2 설비 센서 빅데이터를 이용하여 설비 이상동작 예측 및 지시사항의 고도화 가능

WSN을 통해 수집된 설비정보로 빅데이터를 유지한다. 축적된 데이터로부터 설비 이상동작 탐지와 지시사항의 수준을 고도화 할 수 있다. WSN이라는 네트워크 인프라가 구축되었기 때문에 인공지능, 빅데이터와 같은 다양한 ICT기술을 융합할 수 있다.

9 역할 및 일정

		6월	7월	8월	9월
김남호	서버 및 데이터베이스 구축				
	작업자 애플리케이션 제작				
유동화	서버, 클라이언트 미들웨어 제작				
	Header Switching & Forwarding 구현				
	센서 및 GIO 개발				
김나현	DODAG Topology 구성				
	RPL(DIO 조절)				
공통	최종 테스트 및 발표 준비				