碩士學位論文

작물 생육 환경 모니터링을 위한 비동기 IoT 브로커 설계 및 구현

Design and Implementation of Asynchronous IoT Broker for Monitoring of Crop Growing Environment

仁川大學校 情報技術大學院

컴퓨터 專攻

崔 正 玟

2017年 7月

碩士學位論文

작물 생육 환경 모니터링을 위한 비동기 IoT 브로커 설계 및 구현

Design and Implementation of Asynchronous IoT Broker for Monitoring of Crop Growing Environment

指導敎授　崔 鎭 卓

이 논문을 석사학위 논문으로 제출함

2017年　7月

仁川大學校 情報技術大學院

컴퓨터 專攻

崔 正 玟

崔正玟의 工學碩士學位 論文을 認准함

2017年 7月

審査委員長 (印)

審 査 委 員 (印)

審 査 委 員 (印)

仁川大學校 情報技術大學院

목 차

표 목 차

그 림 목 차

국 문 요 약

작물 생육 환경 모니터링 시스템은 센서 장비를 이용하여 작물의 생육 환경을 수집, 분석하고 이를 바탕으로 생육 단계별 맞춤형 처방을 내릴 수 있도록 지원하기 위해 제안되었다. 이를 위해 작물 생육 환경 데이터를 무선 네트워크 환경에서 동작하는 센서 장비를 이용하여 수집하게 된다. 이때 이러한 센서가 전국에 대량으로 설치되어 데이터를 수집하게 되는 경우 센서 특유의 저속, 저 품질 네트워크 환경과 대량의 커넥션으로 인해 데이터의 손실 또는 처리 지연 현상이 발생되게 된다. 이을 해결하기 위해 신뢰성 있고 안정적으로 데이터를 수집, 처리할 수 있는 수집 시스템이 필요하다.

본 논문에서는 비동기 처리를 지원하는 Vert.x 프레임워크를 기반으로 저속, 저 품질의 네트워크 환경에서 신뢰성 있는 메시지 전송을 위한 MQTT 프로토콜을 사용하여 비동기 IoT 브로커를 설계, 구현하여 수집 시스템을 구축하였다. 또한 구현된 브로커에 대해 아마존 웹 서비스를 이용하여 실험 환경을 구성하고 성능에 대해 실험을 진행하였다. 진행된 실험을 통해 5개의 브로커 인스턴스를 클러스터링하여 2만 커넥션에서 토픽 4개로 나누어 데이터를 전송하고 각 토픽에 대해 각각 처리하였을 때 56000tps로 가장 좋은 성능을 보였다. 이를 통해 작물 생육 환경 모니터링 시스템에서 신속하고 안정적으로 작물 생육 환경을 수집, 처리할 수 있는 수집 시스템으로 본 브로커를 사용 가능하다는 것을 알 수 있었다.

주요어 : MQTT, 대량 커넥션, 비동기 처리

제 1 장 서 론

1.1 연구 배경 및 목적

1.1.1 연구 배경

작물 생육 환경 모니터링 시스템은 작물의 생육 환경을 수집하여 확인, 분석하고, 작물이 건강하게 성장하기 위한 최적의 환경을 유지할 수 있도록 지원하여 작물의 생산성을 확보하고 국내 농가에 경제적 이익을 줌으로써 한국 농촌이 갖고 있는 문제를 해결 할 수 있도록 하고자 제안되었다. 현재 국내 농업은 2016년 총 농업 인구 249만 명 중 132만 명이 60세 이상으로 고령화가 이루어지고 있으며, 국내 농경지는 2008년 175만 ha에서 2016년 167만ha로 지속적으로 하락하고 있다[1,2]. 또한 전체 취업자 중 농림어업 종사자 비중도 10.6%에서 5.7%로 빠르게 감소하고 있다[2]. 이러한 농촌의 고령화, 농업인구감소, 농경지 감소와 더불어 재배 경험이 부족하여 경쟁력 확보가 어려운 귀농, 귀촌인구의 증가와 기후 변화 심화, FTA 확산으로 인한 농업 시장의 작물 수입은 국내 농촌에 안정적인 식량 확보는 물론 경제적인 어려움까지 가져왔다[3,4].

이러한 문제들을 해결하기 위해 제안된 작물 생육 환경 모니터링 시스템은 무선 네트워크 환경에서 동작하는 센서 장비를 농지에 배치하여 농지의 작물의 생육 환경 데이터를 서버로 전송하고 수집된 데이터를 분석, 가공하여 사용자에게 분석된 결과를 사용자에게 제공하게 된다. 이때 저속, 저 품질의 무선 네트워크 환경이라는 점과 센서 장비가 많아질수록 서버와 센서 사이에 발생되는 트래픽으로 인한 병목 현상으로 데이터의 손실 또는 처리 지연 현상이 발생하게 된다. 이러한 문제를 해결하기 위해 대량의 센서에서 수집되는 생육 환경 데이터를 신뢰성 있고 안정적으

로 수집, 처리 할 수 있는 데이터 수집 시스템이 필요하다.

1.1.2 연구 목적

본 연구는 저속 저 품질의 IoT 무선 네트워크 환경에서 신뢰성 있게 서버와 통신하는 기술과 대량의 센서에서 보내오는 작물 생육 환경 데이터에 대해 안정적으로 처리 할 수 있는 기술을 이용하여 브로커를 설계, 구현하고 성능에 대해 확인하므로 작물 생육 환경 모니터링 시스템에서 사용 가능한 데이터 수집 시스템을 개발하는 것을 목적으로 한다.

작물 생육 환경 모니터링 시스템의 데이터 수집 시스템은 센서를 재배지에 설치하고 센서와 서버간의 통신을 통해 생육 환경 데이터를 수집, 분석하여 사용자에게 제공하는 과정으로 진행된다. 이러한 과정에서 센서 기기와 서버가 무선의 저속, 저 품질 네트워크에서 통신해야 하는 것을 감안해야 한다. 또한 대량의 센서를 통해 수집되는 데이터는 빠르게 빅데이터를 발생시키며 이러한 빅데이터는 서버와 센서 사이에서 대량의 트래픽을 발생시켜 데이터의 손실 또는 처리 지연을 불러온다. 따라서 작물 생육 환경 모니터링 시스템은 서버와 센서 간의 통신을 지원하기 위해 신뢰성 있는 메시지 프로토콜을 적용해야 하며, 많은 양의 센서와 통신에 따른 대량 트래픽 환경에서 안정적으로 데이터를 수집, 처리하기 위한 기술이 필요하다.

1.2 연구 방법 및 논문 구성

1.2.1 연구 방법

본 연구는 수집 시스템을 구성하는 비동기 IoT 브로커(AS-Broker)에 대해 설계 및 구현을 진행하고, 성능에 대한 실험을 위해 아마존 웹 서비스(AWS, Amazon Web Services)를 이용하여 서버를 구성한 후 AS-Broker를 실제 구동 후 실험을 진행한다. 이때 실제 대량의 센서 장비로 실험을 진행하기에는 어려움이 있어 센서 장비와 동일한 역할을 하는 가상 센서 장비를 구현하고 AWS를 이용하여 가상 센서 장비를 대량으로 구성하여 실험을 진행한다. 관련연구에서는 사물인터넷 표준 통신 프로토콜과 비동기 처리 기술에 대한 이론적, 기술적 이해를 높이기 위해 관련 참고 서적, 논문 등을 활용하였다.

1.2.2 논문 구성

본 논문은 작물 생육 환경 모니터링을 위한 비동기 IoT 브로커를 설계 및 구현한 논문으로 총 5장으로 구성되어 있으며 내용은 다음과 같다. 제 1장에서는 본 논문의 연구 배경 및 목적과 연구 방법에 대해 설명하며 제 2장에서는 관련 연구로 작물 생육 환경 모니터링 시스템과 AS-Broker의 기반 기술에 대해 조사하여 알아본다. 제 3장에서는 작물 생육 환경 모니터링 시스템에 대한 구성과 수집 시스템에 대해 설명하고 AS-Broker를 설계 및 구현하며 실험 환경을 구축에 관한 설명한다. 제 4장에서는 단일 서버에서 구축된 AS-Broker와 대량의 가상 센서 장비를 이용하여 실험을 진행하고 성능 평가 및 실험 결과에 대해 설명한다. 제 5장에서는 본 논문에 대해 요약하고 결론을 맺는다.

제 2 장 관련 연구

2.1 스마트 팜

스마트 팜은 정보통신기술을 농업에 접목하여 작물의 생육 정보와 환경 정보 등에 대해 정확한 데이터를 기반으로 생육 환경을 점검하고, 적절한 처방으로 생산성과 품질을 높이는 농업을 뜻한다[5].

현재 국내 농업 시장은 국제적 개방과 산업구조의 고도화로 인한 농가 인구의 감소와 고령화, 곡물 자급률의 하락 등의 문제를 해결하고 농업 선진국으로 도약하기 위해 스마트 팜에 대한 논의가 활발하다.

2.1.1 스마트 팜 개요

2016년 농림어업조사 결과에 따르면 농가 고령화 비율이 2016년 12월 기준으로 40.3%로 전년에 비해 1.9% 증가하고 농가인구는 249만 6천 명으로 전년에 비해 2.8% 감소하여 농가의 고령화가 빠르게 진행되고 있으며 농가의 인구는 지속적으로 감소하고 있는 것으로 나타났다[1]. 또한 2016 농림축산식품주요 통계에 의하면 국내 농경지는 2008년 1,759천 ha에서 2015년 1,679천ha로 지속적으로 감소하고 있다[2]. 이러한 변화 실태를 정리하면 [표 2-1]과 같다.

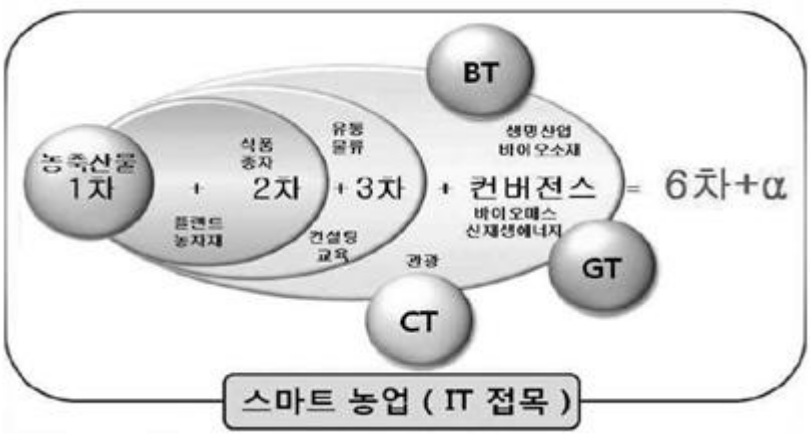
또한 최근 농촌에 귀농, 귀촌 인구가 증가하고 있으나, 기존의 농법으로는 빠르게 경쟁력을 확보하는데 어려움이 있다. 더 나아가 기후 변화의 심화로 인해 작물 생산량이 일정하지 못해 가격이 심하게 등락하고 있어 안정적인 식량 확보에 어려움을 겪고 있다. 이러한 한계를 극복하기 위해 정보통신기술이 접목된 스마트 팜의 보급 필요성이 증가하고 있다[4,5]

[표 2-1] 연도별 농가 인구 및 경작지 면적[1,2]

구분	년 도								
	08	09	10	11	12	13	14	15	16
농가 인구	3,187	3,117	3,063	2,962	2,912	2,847	2,752	2,569	2,496
60세 이상 농가	1,386	1,393	1,279	1,307	1,344	1,361	1,369	1,293	132,5
경작지	1,759	1,737	1,715	1,698	1,730	1,711	1,691	1,679	-

(인구 단위 : 천명 / 경작지 단위 : 천ha)

스마트 팜 기술은 [그림 2-1]과 같이 ICT(정보통신), BT(바이오), ET(환경) 등 첨단 기술과의 융합을 통해 농업 문제 해결 방안으로 빠르게 부상하고 있으며 선진국들의 주도 하에 기존 식량 생산 위주에서 벗어나 1차, 2차, 3차 산업과 결합되어 6차 산업으로 확대되고 있다[3].



[그림 2-1] 스마트 농업의 현황과 발전 방향[3]

스마트 팜은 첨단 기술을 온실, 축사, 과수원 등에 접목해 원격 및 자동으로 작물과 가축의 생육환경을 적절히 제어할 수 있는 농장의 개념으로, 더 넓은 의미로는 노지농업, 시설원예 및 축산분야에서 농산물 생산, 유통, 소비의 전 주기적 과정을 농업 ICT 융합기술 적용을 통한 농촌의 삶의 질 향상을 도모하는 농업 형태까지 포함한다[4].

[표 2-2] 스마트 농업 분야 관련 국내 시장규모 및 전망

구분	2015	2016	2017	2018	2019	2020	CAGR
세계시장	28.1	31.4	35.2	39.3	44.0	49.2	11.8
국내시장	36,051	41,699	44,493	47,474	50,655	54,048	15.5

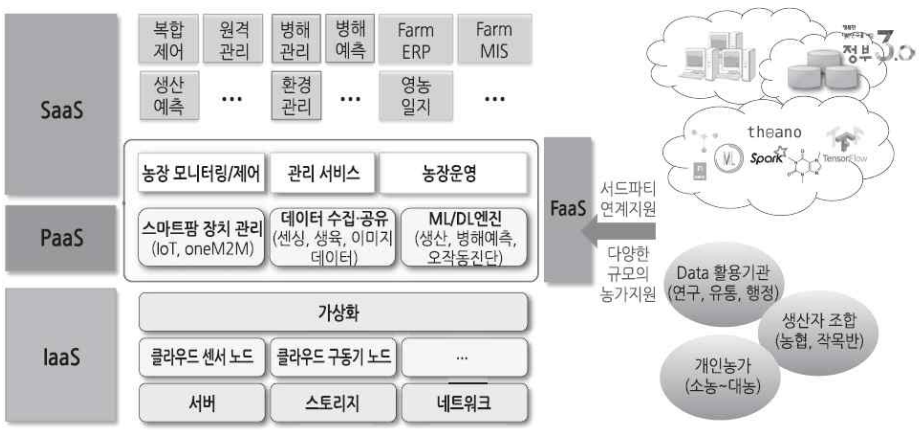
(단위 : 국외: 억 달러, 국내: 억 원, %)

스마트 팜 관련 시장은 꾸준히 증가하고 있는 추세로, [표 2-2]와 같이 세계 스마트 팜 관련 시장 규모는 2015년 28억 달러 수준에서 2020년에는 49.2억 달러 수준까지 11.8% 성장률로 빠르게 증가할 것으로 전망하고 있다. 또한 국내 스마트 팜 생산 관련 시장도 2015년 3조 6000억 원 수준에서 2020년 5조 4000억 원 수준으로 14.5의 성장률로 성장할 것으로 전망하고 있다. 이는 정부에서 농업의 미래 성장산업화 가속화를 위한 경쟁력 제고 및 성장 동력 창출을 위한 스마트 팜 확산 대책을 마련하고 한국형 스마트 팜 보급 및 소프트웨어 수입 대체, 전문 인력 육성, 실습 교육, 사후관리 강화 등의 인프라 강화 등을 위한 지원 정책을 추진하고 있으며, 이러한 정부 차원의 적극적인 지원은 국내 스마트 팜 시장의 주요 성장 요인으로 꼽을 수 있다[3].

2.1.2 클라우드 기반 스마트 팜 서비스

클라우드 기반의 스마트 팜 서비스(FaaS, Farm as a Service)는 클라우드 컴퓨팅 기술을 기반으로 스마트 팜을 관리, 운영하는데 있어 필요한 서비스의 기술적 요구사항과 구성 내용으로 정의된다. 특히, PaaS(Platform as a Service) 기반으로 다양한 형태의 스마트 팜 자원 정보를 가상화하고, 데이터 수집, 제어, 운영, 관리 등을 위한 상위 응용 서비스에 대한 내용을 주 내용으로 한다[6].

클라우드 기반 스마트 팜 서비스는 서버, 스토리지, 미들웨어, 응용소프트웨어 등 IT 인프라 자원을 네트워크를 통해 공유하는 클라우드 기술을 사용한다. 또한, 센서 노드, 구동기 노드 같은 스마트 팜 장치들도 가상화하여 운영한다. 이를 통해, 기존 농가별로 설치 운영되어 이기종 스마트 팜 시스템 및 공급사별로 개별적, 분산적으로 설치, 운영되었던 레거시 시스템을 클라우드 기술을 통해 통합 운영할 수 있으며, 농장 관리 기능을 저가의 서비스 형태로 이용할 수 있다[6].



[그림 2-2] 클라우드 기반의 FaaS 개념도[6]

클라우드 기반 스마트 팜 서비스는 농장에서 작물을 생산하는데 있어 작물의 생육 상태를 모니터링하고, 수동 또는 자동으로 시설 및 장치를 제어하고, PaaS 기반의 서비스로 농장 운영 및 개발 환경을 제공한다. FaaS는 장치 관리 서비스, 데이터관리 서비스, 모델 관리 서비스 등의 관리 서비스와 스마트 팜 모니터링 서비스, 제어서비스, 농장 생산, 경영관리 등을 지원하는 스마트 팜 운영 서비스로 구성된다[6].

2.2 사물인터넷 표준 프로토콜

사물인터넷(IoT, Internet of Things) 기술은 전 세계적으로 많은 관심이 집중되어 있으며, 그에 따라 엄청난 속도로 발전하고 있다. 이러한 사물인터넷의 발전에 따라서, 사람들의 일상 속에서 다양한 정보들을 습득하고 이용할 수 있도록 하는 초 연결 사회가 도래할 것으로 전문가들은 현재 예측하고 있다. 이처럼 사물인터넷의 발전에 따라 다양한 표준화 단체(ITU-T, ETSI, oneM2M 등)에서 사물 인터넷의 표준화를 위해서 플랫폼 표준화를 진행하고 있으며 그에 따른 플랫폼을 제시하고 있는 상황이다[8].

그 중, oneM2M[7]에서는 현재 다양한 표준화 단체에서 제시하고 있는 플랫폼들을 하나로 규합하여, 최상의 플랫폼에 대한 표준화 및 플랫폼 활용 방안을 제시하고 있다. oneM2M은 각 지역을 대표하는 표준기관(TTA(한국), ETSI(유럽), TIA(북미), ATIS(북미), ARIB(일본), TTC(일본), CCSA(중국))들이 범국가적 범지역적 서비스 플랫폼 표준 기술을 개발하는 것에 목적을 두고 2012년 7월에 결성된 파트너십 프로젝트이다[8].

oneM2M은 엔티티 간 메시지(oneM2M Primitive) 교환 프로토콜, 에러처리를 다루는 oneM2M 코어 프로토콜 그리고 코어 프로토콜과 전송계층 프로토콜(CoAP, HTTP, MQTT 등)과의 바인딩을 정의하였다. oneM2M 코어 프로토콜은 Mca, Mcc 그리고 Mcc' 레퍼런스 포인트 상에 정의된 API를 의미한다. oneM2M 코어 프로토콜은 전송계층 프로토콜과 독립적으로 설계되어 있으며 HTTP, CoAP 및 MQTT와 같이 다양한 프로토콜과의 바인딩을 지원한다[9]. 이 중 MQTT[10,11]와 CoAP[12,13]에 대해 알아보도록 한다.

2.2.1 MQTT

MQTT(Message Queue Telemetry Transport)[10,11] 프로토콜은 1999년 IBM에서 발표한 메시지 푸시 프로토콜이다. MQTT 프로토콜은 제한된 성능과 빈약한 네트워크 연결 환경에서 동작하도록 설계된 대용량 메시지 전달 프로토콜로 낮은 대역폭과 긴 지연시간에도 불구하고 신뢰적인 메시지를 전송하도록 설계되었다. 신뢰성과 저 전력이 특징인 MQTT 프로토콜은 2013년 OASIS(Organization for the Advancement of Structured Information Standards)에 의해 IoT를 위한 표준 프로토콜로 선정되었다. 이후 전기, 전자, 통신, 인터넷 등 다양한 분야에 활용될 수 있도록 지속적인 업데이트가 진행되고 있다. 또한 오픈 소스 스택과 컨설팅을 제공하는 많은 상업 회사와 이클립스 커뮤니티 내의 지원을 받고 있다[14].

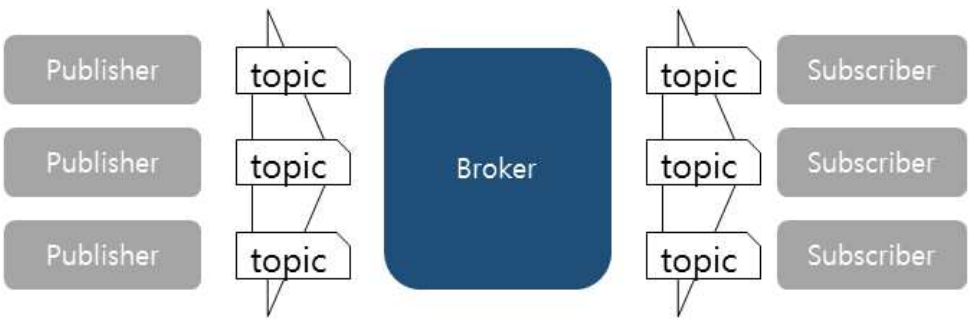
MQTT 프로토콜은 Publish/Subscribe 모델을 사용하며 MQTT 네트워크 노드 간에 메시지를 관리하고 라우팅하기 위해 중앙 MQTT 브로커를 필요로 한다[15]. MQTT 서버라고 하지 않고 브로커라고 하는 이유는 MQTT가 발행인과 구독자가 메시지를 주고받을 수 있도록 다리를 놔주는 역할만을 하기 때문이다. 다른 기능들은 중계를 도와주는 부가 기능일 뿐이다.

MQTT는 IoT 환경에 적합한 통신을 위해 경량화, 유연성, 확장성, 생산성 등의 특징을 지니고 있다. 경량화는 경량 메시지 포맷을 사용하여 80~100kb 수준의 메모리를 사용하므로 구현이 간단하며, 이벤트 방식과 QoS(Quality of Service)를 통해 다수의 사용자와 디바이스를 지원하고, 응용코드의 변경 없이 기능을 확장 할 수 있는 유연성과 확장성을 가지고 있다. 또한 생산성 부분은 탐지, 저장, 전달, Publish/Subscribe 기능을 제공하기 때문에 별도의 응용이 필요 없고, 간단한 개념으로 만들어지므

로 개발자의 학습이 용이하다[14]. MQTT는 TCP를 사용하여 고 신뢰성, 정렬, 에러 검사를 특징으로 하는 트랜스포트 계층을 구현하며 M2M과 IoT에서 사용하는 것을 목적으로 하는 만큼, 낮은 전력으로도 사용할 수 있도록 설계되어있다. 프로토콜의 설계 의도를 정리해보면 다음과 같다[15,16,17].

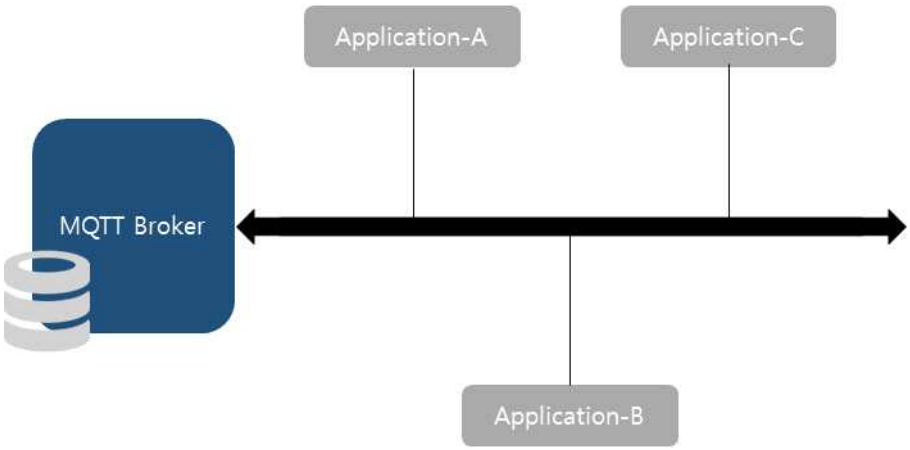
- 프로토콜이 차지하는 모든 면의 자원 점유를 최소화
- 느리고 품질이 낮은 네트워크의 장애와 단절에 대비
- 클라이언트 동작에 자원 활용이 극히 제한적임을 고려
- 다수의 클라이언트 연결에 적합한 Publish/Subscribe 네트워크
- 신뢰성 있는 메시지 전달을 위한 QoS 옵션 제공
- 개방형 표준 메시지 프로토콜 지향

MQTT 프로토콜은 [그림 2-3]과 같이 다자 간 메시지를 서로 공유하기 위해 특정 주제에 대해 구독하는 클라이언트(Subscriber)는 구독을 신청을 하고, 발행하는 클라이언트(Publisher)는 발행하는 구조이다. 이를 위해 특정 토픽(Topic)에 대해 Publisher와 Subscriber, 그리고 중간에서 이들을 중계하는 브로커 서버로 구성된다[15].



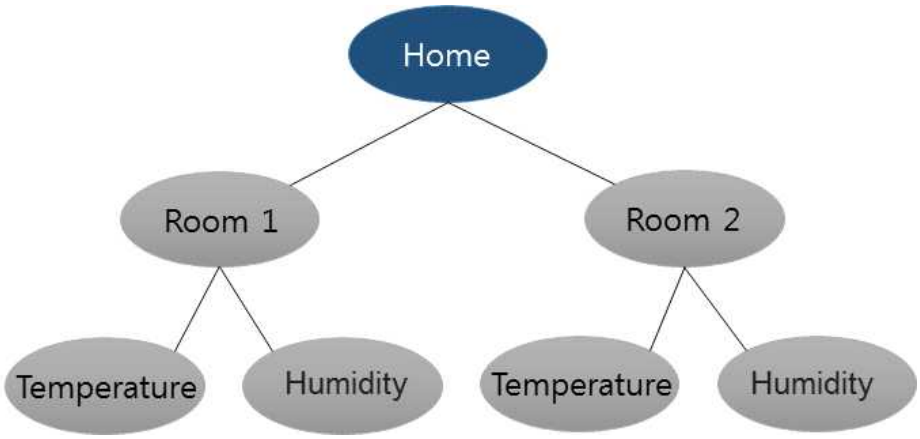
[그림 2-3] MQTT 기본 Publish/Subscribe 구조

이러한 Publish/Subscribe 모델을 지원하기 위해 [그림 2-4]와 같이 MQTT는 메시지 버스 시스템을 가진다. MQTT 브로커가 메시지 버스를 만들고 여기에 메시지를 흘려보내면, 버스에 연결된 Subscriber들이 메시지를 읽어가는 방식이다. 메시지 버스에는 다양한 주제의 메시지들이 흐를 수 있으며 메시지를 구분하기 위해서 토픽을 이름으로 하는 메시지 채널을 만든다. Publisher 또는 Subscriber는 메시지 버스에 연결하고 관심 있는 토픽을 등록해서 메시지를 구독하거나 발행한다.[14, 18]



[그림 2-4] 메시지 버스 흐름도

Publisher와 Subscriber 사이에 전송되는 메시지는 다양한 주제의 하위 주제로 분류되어 송신, 수신될 수 있으며, 각 메시지의 특정 주제를 토픽이라고 한다. 토픽은 하위 토픽을 계층적으로 가질 수 있으며, /를 이용해서 아래 [그림 2-5]과 같이 계층적 구조로 표현할 수 있다[15].



[그림 2-5] MQTT 토픽 구조

또한 토픽은 특정한 토픽과 관련한 다양한 메시지를 받을 수 있도록 와일드카드를 사용할 수 있다. +은 단일 레벨 와일드카드로 하위 단일 레벨에서 발생하는 메시지를 구독할 때 사용되며 #은 다중 레벨 와일드카드로 하위 다중 레벨에서 발생하는 모든 메시지를 구독할 때 사용된다[15]. 와일드카드에 대해 정리하면 다음 [표 2-3]과 같다.

[표 2-3] 와일드카드에 따른 동작 방법

와일드카드	동작 방법
+	단일 레벨 와일드카드, 하위 단일 레벨에서 발생하는 메시지 구독
#	다중 레벨 와일드카드, 하위 다중 레벨에서 발생하는 모든 메시지 구독

MQTT는 디바이스들의 처리 능력, 네트워크 대역폭, 메시지 오버헤드 등 주변 상황에 맞게 시스템이 동작할 수 있도록 3단계의 QoS를 제공한

다. QoS 0 레벨은 메시지의 전달 여부에 대해 확인하지 않고 단순히 전달만 하는 방식이고, QoS 1 레벨은 메시지의 핸드셰이킹 과정을 정확하게 추적하지 않기 때문에 중복으로 전송될 수 있다. QoS 2 레벨은 메시지를 정확히 한 번만 전달하는데 핸드셰이킹 과정을 추적하기 때문에 높은 품질을 보장하지만 성능이 저하되는 문제가 있다. Publisher와 Subscriber 모두 QoS를 지정할 수 있으나 Publisher가 지정한 QoS가 우선시된다[16,17,18]. QoS 레벨에 대해 정리하면 다음 [표 2-4]와 같다.

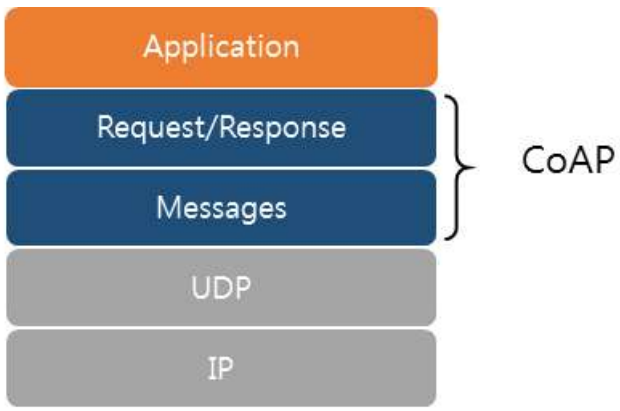
[표 2-4] QoS에 따른 동작

레벨	동작방법
QoS 0	메시지 한번만 절달, 전달 여부 확인 안함. Fire and Forget 타입.
QoS 1	메시지는 반드시 한번 이상 전달. 메시지의 핸드셰이킹 과정을 엄밀하게 추적하지 않기 때문에, 중복 전송 될 수 있음.
QoS 2	메시지는 한번만 전달. 메시지의 핸드셰이킹 과정을 추적. 높은 품질을 보장하지만 성능의 희생이 있음.

MQTT 브로커의 종류는 다양하다. 대표적인 브로커로 Mosquitto, RabbitMQ, ActivceMQ 등이 있다. 이 중 Mosquitto는 MQTT 버전 3.1과 3.1.1을 완전 지원하는 오픈소스 브로커로 Publish/Subscribe 모델을 사용하여 메시지를 전달하는 경량의 브로커다. 또한 저 전력 센서나 휴대전화 등의 모바일 기기, 임베디드 컴퓨터, 마이크로 컨트롤러와 같은 M2M 메시지에 적합하도록 설계되었으며 마이크로소프트의 윈도우즈, 애플의 OS X, 리눅스 계열의 다양한 플랫폼을 지원한다[15].

2.2.3 CoAP

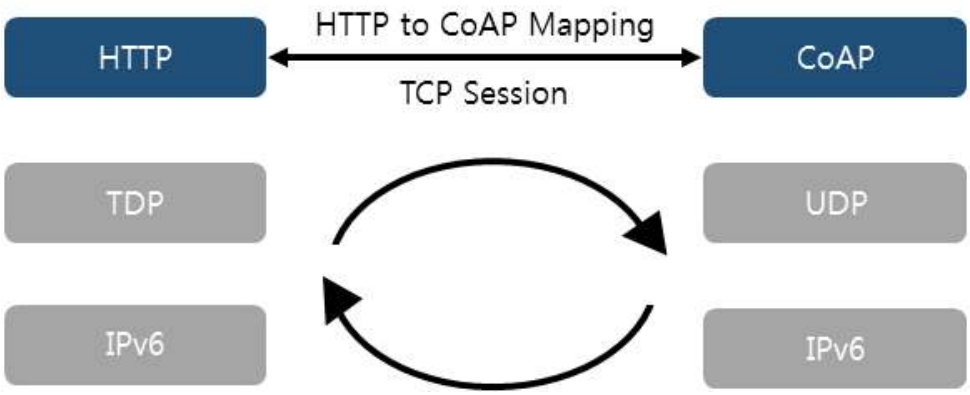
CoAP(Constrained Application Protocol) 프로토콜[12,13]은 인터넷 표준 단체인 IETF(Internet Engineering Task Force)에서 다양한 디바이스가 인터넷에 연결될 것으로 예상하고, 워킹그룹을 만들어 소형 디바이스에 탑재되는 네트워크 표준을 연구하면서 시작되었다. 워킹그룹 중 하나인 CoRE(Constrained RESTful Environments)에서 소 용량, 저 성능 노드와 손실이 높고 전송률이 낮은 네트워크에서 경량화된 방식으로 메시지를 주고받을 수 있는 CoAP 프로토콜을 개발하였다. 이후, IETF에서 표준 프로토콜로 사용하고 있으며, IoT 표준화 단체인 oneM2M, OIC(Open Interconnect Consortium)에서도 CoAP을 기반으로 IoT 환경 구축을 위한 표준화를 진행하고 있다[14].



[그림 2-6] CoAP 프로토콜 계층 구조

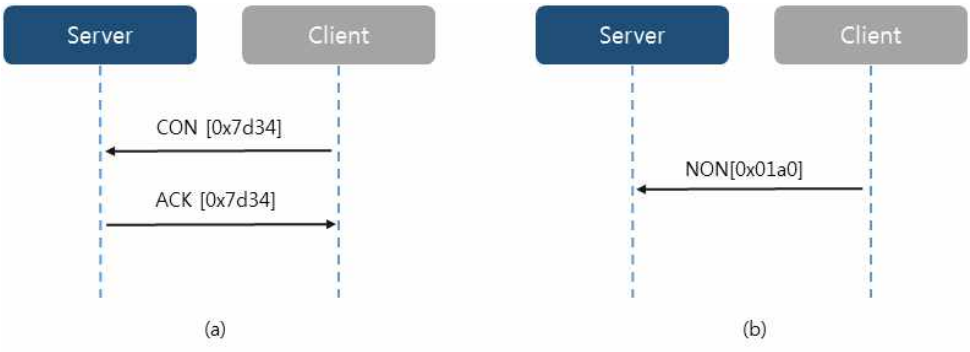
CoAP 프로토콜은 [그림 2-6]과 같이 전송계층은 UDP와 애플리케이션 사이에 위치하고 있으며, UDP 환경 위에서 Request/Response 방식과 Message전송 계층으로 구성되어 있다[19].

CoAP는 기본적으로 RESTful(Representational State Transfer) 기반의 비동기 통신 프로토콜이다. HTTP의 명령어 방식을 따르기 때문에 매핑을 통해 기존의 HTTP 프로토콜과도 쉽게 변환 및 연동이 가능하며 GET, POST, PUT, DELETE 메서드를 이용할 수 있다[18]. [그림 2-7]과 같이 HTTP와 연동 할 수 있다.



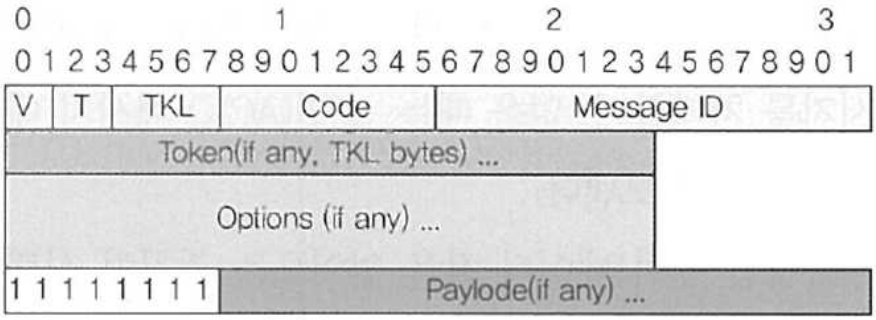
[그림 2-7] HTTP와 CoAP의 연동방식

하지만 CoAP 프로토콜은 UDP 통신 기반으로 TCP 통신에서 제공하는 MQTT 프로토콜에 비해 신뢰성이 다소 부족하다. 이에 신뢰성 있는 데이터를 전달하기 위해 신뢰 메시지 전송과 비 신뢰 메시지 전송을 선택하여 사용할 수 있도록 기능을 제공한다. CoAP의 메시지 전송 방식은 다음 [그림 2-8]와 같다.



[그림 2-8] CoAP (a)신뢰 메시지 전송 / (b)비 신뢰 메시지 전송

이러한 신뢰, 비 신뢰 메시지를 전송하기 위해 확인형(CON, Confirmable), 비확인형(NON, Non-Confirmable), 승인(ACK, Acknowledgement), 리셋(RST, Reset) 총 4가지 종류의 메시지 형태를 지원한다. CON 메시지는 신뢰성 있는 전송을 원할 때 사용하고, CON 메시지에 대한 응답 메시지인 ACK 메시지를 받아야 재전송을 멈춘다. NON 메시지는 요청에 대한 응답 메시지의 도착 여부를 신경 쓰지 않는다. RST 메시지는 에러가 있을 시에 사용한다[20].



[그림 2-9] CoAP 메시지 포맷[21]

[그림 2-9]는 CoAP의 메시지 포맷으로 간단한 바이너리 포맷으로 인코딩 된다. 메시지는 4바이트 고정 헤더를 포함하며 0에서 8바이트 길이의 토큰이 위치하며 그 뒤로 옵션이 온다. 페이로드가 있는 경우, 그 다음부터 데이터그램 끝까지 배치된다[21].

CoAP 프로토콜은 보안 요소를 위해 UDP와 CoAP 계층 중간에 DTLS(Datagram Tran-sport Layer Security)를 사용할 수 있으며 4가지 보안모드를 제공한다. 보안모드 중 기본 모드인 NoSec 모드는 보안 프로토콜이 적용되지 않은 기본모드이며 IPsec이나 기타 하위 계층의 보안 방식이 적용 가능하다. PreSharedKey 모드는 DTLS를 적용하며, 사전 공유키를 활용하여 Node/Key간의 1:1 매칭을 지원한다. RawPublicKey 모드는 DTLS를 적용하며, 비대칭 비밀키 쌍을 지원한다. Certificate 모드는 DTLS를 적용하며, 비대칭 비밀키 쌍과 X.509 인증을 지원한다[19].

CoAP은 UDP 패킷만을 활용함으로써, NAT 환경에서 동작시키기 위해서는 터널링을 하거나, 포트 포워딩과 같은 방법을 사용해야 하는 단점이 있다. 또한 CoAP에서 사용하는 DTLS의 경우 기반 암호기법인 AES(Advanced Encryption Standard)가 IoT 환경에서는 다소 무겁기 때문에 AES보다 경량화 된 암호화 기법을 이용해서 보안성을 개선해야 한다[14].

2.3 고속 처리를 위한 비동기 기술

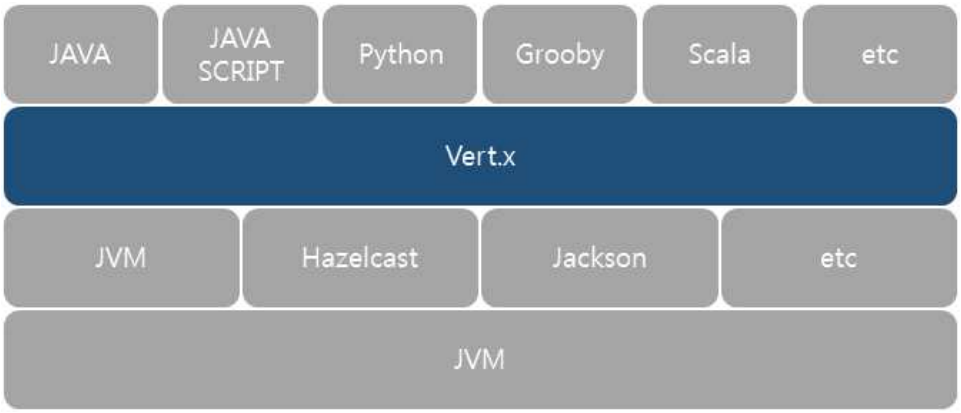
기존 애플리케이션 서버는 동시에 요청한 작업을 다수의 쓰레드를 이용하여 처리하는 멀티쓰레드(Multi-Thread) 방식을 주로 사용하였다. 이와 같은 구조는 작업 요청 수에 비례하여 쓰레드를 생성하기 때문에 I/O 다중화 작업 시 CPU, 메모리 자원소모가 심하고 Context Switching(문맥교환), Deadlock(데드락)이 자주 발생하는 설계상의 한계가 존재 하였다. 이러한 문제를 해결하기 위해 설계된 방식이 비동기 처리 방식이며 오랜 기간 많은 연구 및 성과 등을 통해 발전되었다[22]. 본문에서는 비동기 방식으로 처리하는 프레임워크 중 Vert.x[23]와 Node.js[29]에 대해 알아보도록 한다.

2.3.1 Vert.x

Vert.x[23]는 비동기 병렬 애플리케이션을 위한 차세대 소켓 통신 프레임워크로서 이벤트 기반 프로그래밍 모델을 제공하며 Non-Blacking 애플리케이션을 위한 비동기 프로그래밍 기술을 지원한다. Vert.x는 서버 네트워크 환경을 구축하고 운영하는 서버 프레임워크이며 Node.js로부터 영향을 받아 이벤트 기반 프로그래밍 모델을 이용한 비동기 형태의 API를 제공한다. 그렇기 때문에 Vert.x의 API는 Node.js와 매우 유사하며 둘 다 비동기 형태의 API를 제공한다[24,25,26].

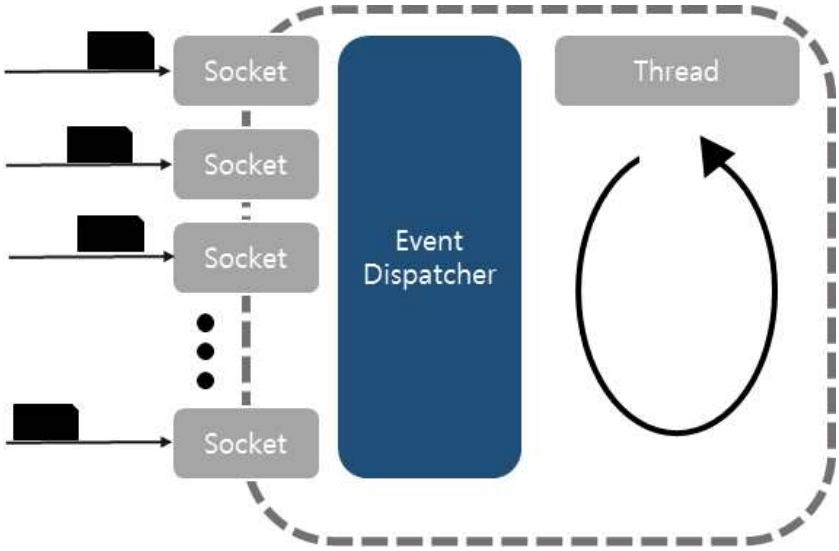
Vert.x는 [그림 2-10]과 같이 자바 가상머신에서 구동되며 다양한 개발 언어를 지원한다. 싱글 쓰레드 프로그램처럼 구현해도 이벤트 기반으로 동작하여 개발이 단순하고 생산성이 높다. 자바 가상머신 위에서 동작하기 때문에 다중코어가 지원되며 Hazelcast[27]가 내장되어있어 IMDG(In Memory Data Grid)를 지원하여 대용량 서버로 사용할 수 있다. 또한 자바 진영의 강력한 네트워크 프레임워크인 Netty가 내장되어

강력하게 네트워크 처리를 할 수 있으며 과부하 방지 처리가 내장되어 있어서 단독으로 사용이 가능할 정도의 성능과 안정성을 가지고 있다. 그 이상의 고성능을 확보하기 위한 방법으로는 클러스터 구성이 가능하기 때문에 여러 대의 서버에 Vert.x를 설치하여 성능을 향상시킬 수 있다[24].



[그림 2-10] Vert.x의 구조

Vert.x의 동작은 [그림 2-11]과 같이 데이터가 수신되면 Event Dispatcher에 Verticle이 생성되고, 이것을 비동기로 수행시킨다. Vert.x는 싱글 쓰레드 상에서 동시 작업을 처리하기 위해 이벤트 기반으로 동작하는데 이는 Reactor 패턴을 기초로 한다. Reactor 패턴은 동시에 요청되는 작업에 대한 이벤트를 적절한 핸들러에게 전달해주는 이벤트 처리 패턴이다. 수행이 완료되면 이벤트를 발생시켜 후처리를 한다[24,26].



[그림 2-11] Vert.x 동작

Vert.x의 주요 특징은 다음과 같다[28].

- 다양한 언어를 사용 가능(JavaScript, Ruby, Java Python 등)
- 간단한 병렬 모델
- JVM을 활용하여 여러 서버와 프로세스 사이의 통신을 직접 처리할 필요 없이 원활하게 이용가능
- 비동기 프로그래밍 모델
- 분산 이벤트 버스를 제공
- 모듈 시스템과 개인 모듈 저장소를 제공

Vert.x의 주요 개념과 사용되는 대표적인 용어를 정리하면 다음과 같다[28].

- Verticle

Vert.x에서 애플리케이션을 구성하는 하나의 컴포넌트 배포 단위를 Verticle이라고 부르며 Javascript, Ruby, Java, Groovy로 작성 가능함. 애플리케이션을 작성할 때 하나의 Verticle로 작성하거나 각각의 Verticle이 이벤트 버스로 통신하는 세트로 구성할 수 있음.

- Verticle 인스턴스

  Vert.x 인스턴스는 JVM 인스턴스 내에서 동작하며, 하나의 Vert.x 인스턴스에서 여러 Verticle을 실행시킬 수 있음. 또한, 각각의 Verticle은 고유의 클래스 로더를 가지며 Verticle 간에 정적 멤버, 전역 변수 등을 통한 직접적인 상호작용을 막을 수 있음. 네트워크 상의 여러 호스트에서 동시에 많은 Vert.x 인스턴스가 실행될 수 있고 이벤트 버스를 형성해 Vert.x 인스턴스 간에 클러스터를 구성할 수 있음.

- 동시성

  Verticle 인스턴스는 항상 동일한 쓰레드에서 실행됨이 보장되며 모든 코드를 단일 쓰레드 동작 형태로 개발할 수 있기 때문에, Vert.x를 사용하는 개발자에게 개발하기 편한 환경을 제공함.

- 이벤트 기반 프로그래밍 모델

  Vert.x는 Node.js 프레임워크와 비슷하게 이벤트 기반 프로그래밍 모델을 제공하기 때문에 서버 프로그래밍을 할 때 개발해야 하는 코드의 대부분은 이벤트 핸들러에 관한 것임.

- 이벤트 루프

  Vert.x 인스턴스는 내부적으로 쓰레드 풀을 관리하며 각각의 쓰레드에서는 이벤트 루프를 실행. 루프를 수행 중 처리해야 할 이벤트가 있다면, 해당 핸들러를 호출하는 방식으로 동작.

- 메시지 전달

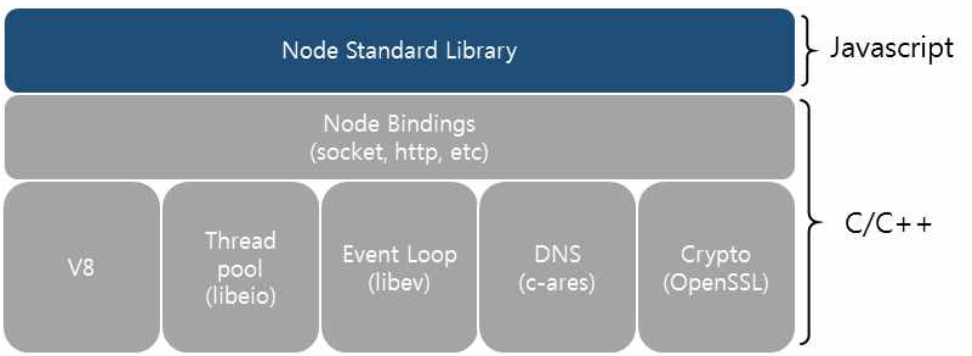
  Verticle 간의 통신은 이벤트 버스를 이용함. Vert.x에서는 많은 Verticle 인스턴스 생성 및 이들 간의 메시지 전달을 통해 Verticle 코드에 대한 멀티 쓰레드 실행이 없이도 사용 가능한 코어에 맞게 시스템 확장이 가능.

- 데이터 공유

  Vert.x는 전역에서 접근할 수 있도록 Shared Map이라는 것을 제공. 그리고 Verticle 사이에서는 오직 불변 데이터만 공유되게 함.

2.3.2 Node.js

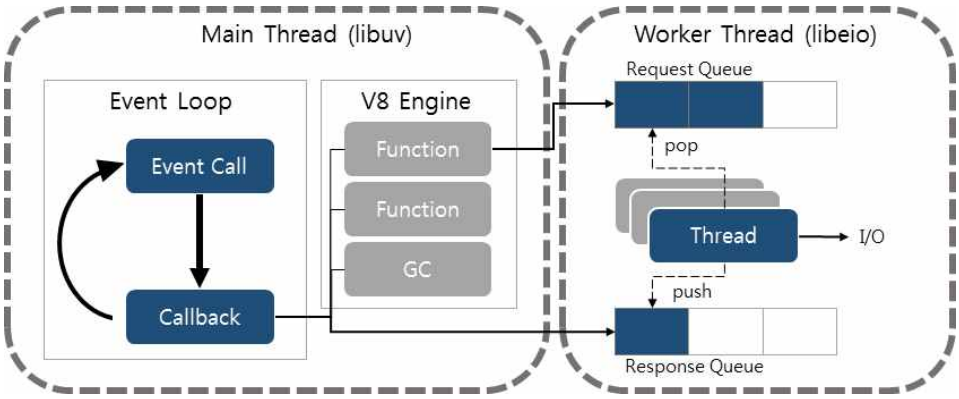
Node.js[29]는 2009년 Ryan LienhartDahl이라는 프로그래머가 JSConf.eu 2009[30]에서 발표하면서 세상에 알려지기 시작했다. MIT 라이선스로 Joyent에서 후원하며 누구나 쓸 수 있는 오픈소스이다. 처음에는 웹 개발을 위해 Web.js라는 이름으로 만들어졌지만 그 기능이 확장되면서 Node.js가 되었다[31].



[그림 2-12] Node.js의 구조

Node.js의 구조는 다음 [그림 2-12]과 같이 Node.js는 고속 자바스크립트 엔진인 V8 위에서 동작하는 이벤트 처리 I/O 프레임워크이며 대부분의 자바스크립트가 웹브라우저에서 실행되는 것과는 달리 Node.js는 서버 측에서 실행된다. Node.js는 일부 CommonJS 명세를 구현하고 있으며 쌍방향 테스트를 위해 REPL(Read-eval-eventloop)환경을 포함하고 있다[31].

또한 Node.js는 싱글 쓰레드 모델이라 다중코어 환경에서도 하나의 코어만 사용한다. 개발이 단순하여 높은 생산성을 보이며 다수의 모듈이 존재하여 개발이 용이하다[24]. 또한 기본적으로 C++기반의 라이브러리들로 구성되었으며 Standard Library에서 V8엔진의 기능에 접근하기 위해 Bindings 통하여 접근 하도록 설계되었다[22].



[그림 2-13] Node.js 구조

[그림 2-13]과 같이 Node.js 프로세스는 이벤트 루프를 실행하는 메인 쓰레드와 I/O 처리를 담당하는 워커 쓰레드로 작동한다. 워커 쓰레드는 쓰레드 풀을 이용하여 관리되기 때문에 이벤트 처리와 I/O 처리는 병행처리되지만 콜백은 모두 메인 쓰레드의 이벤트 루프에서 처리된다. 이벤

트 처리를 위한 이벤트 루프와 I/O 처리를 위한 워커 쓰레드가 완벽히 분리되어 있기 때문에 이벤트 루프에서 I/O 작업에 대한 이벤트만 담당하고 커널의 I/O 요청을 Non-blocking I/O 방식을 통해 병렬적으로 처리하여 I/O 다중화 처리에 적합하다[22].

Ryan Lienhart Dahl의 JSConf.eu 2009년 발표에 의하면 Node.js의 특징은 다음과 같다[30,31].

- 싱글 쓰레드, 이벤트루프 기반
- NPM(Node Package Modules)
- V8 자바스크립트 엔진 사용

Node.js는 싱글 쓰레드, 이벤트루프 기반으로 만들어졌다. 개발자들은 병렬 쓰레드가 싱글 쓰레드에 비해 더 빠르다고 생각을 하는데 병렬 쓰레드로 만들 경우 클라이언트 수가 많아지면 그만큼 메모리 사용량이 늘어나고 서버는 과부화되어 처리를 하지 못하게 된다.

Node.js의 큰 장점 중 하나는 NPM의 존재이다. NPM은 아이작 슈레터(Issac Z, Schluter)가 만든 노드의 모듈 패키지 관리자 도구로 CommonJS의 표준을 준수하여 구성 배포되고 있다. NPM을 통해서 빠른 생태계가 활성화되었고 개발의 편리성을 증대시켰다. Node.js 개발초기에 NPM을 따로 설치해줘야 했지만, 현재는 Node.js를 설치하면 자동으로 NPM이 설치된다.

V8 자바스크립트 엔진은 구글에서 개발된 오픈소스 JIT 가상머신형식의 자바스크립트 엔진이며 구글 크롬브라우저와 안드로이드 브라우저에 탑재되어 있다. V8 자바스크립트 엔진은 ECMAScript(ECMA - 262)3rd Edition 규격의 C++로 작성되었으며, 독립적으로 실행이 가능하다. 또한, C++로 작성된 응용 프로그램의 일부로 작동할 수 있다. V8은 자바스크립트 바이트 코드로 컴파일하거나 인터프리트하는 대신 실행하기 전

직접적인 기계어로 컴파일하여 성능을 향상시켰다. 추가적인 속도 향상을 위해 인라인 캐싱과 같은 최적화 기법을 적용하였다.

제 3 장 비동기 IoT 브로커 설계 및 구현

3.1 작물 생육 모니터링 시스템

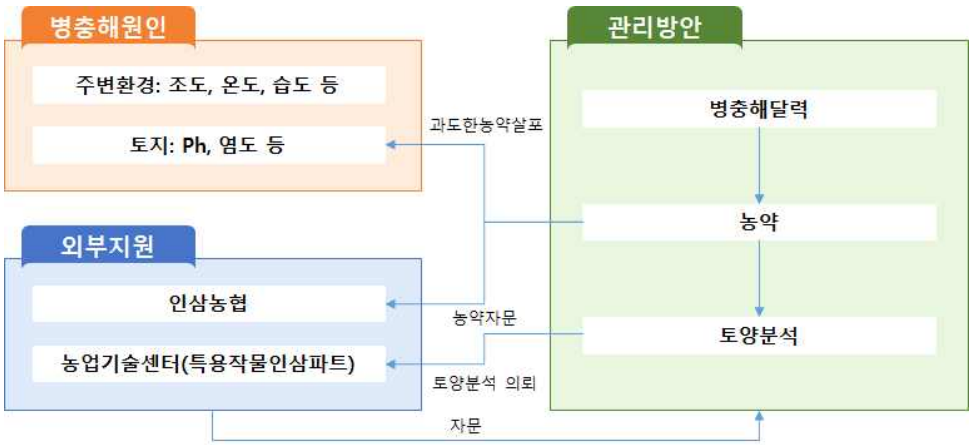
3.1.1 시스템 개요

한국 농촌은 [그림 3-1]과 같이 고령화와 재배 경험이 부족한 귀농인구 증가, FTA로 인한 해외 농산물 수입 등으로 인해 생산성과 경제성에서 많은 문제가 야기되고 있으며, 이를 해결하기 위해 다양한 시도를 하고 있으나 현실적으로 실효를 거두고 있지 못하고 있다.

특히 기존 재배 방법에서 크게 벗어나지 못했고, 토양 환경 변화에 대한 정보 부족으로 인한 병해충 창궐과 비과학적인 재배로 인한 노동력 소모로 인해 이러한 문제를 해결하지 못하고 있다.

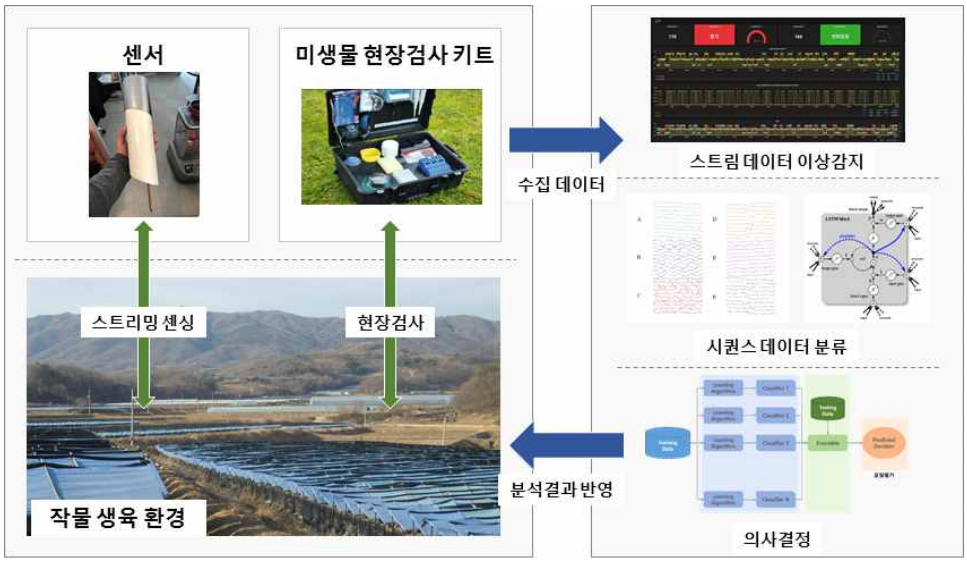
[그림 3-1] 국내 농촌의 문제

또한 [그림 3-2]와 같이 농가는 재배지 관리를 위해 병충해 달력을 기반으로 하는 농약 살포와 농업 관리 상급기관으로 토양 분석 의뢰를 통해 상태를 확인해야 하지만, 토지의 정확한 상태 점검 없이 정해진 규칙에 따라 농약을 살포하게 되어 잔류농약 문제와 토지의 산도, 염도에 변화를 주어 작물의 품질 하락과 또 다른 병충해의 원인이 되기도 한다.



[그림 3-2] 농가의 관리 현황

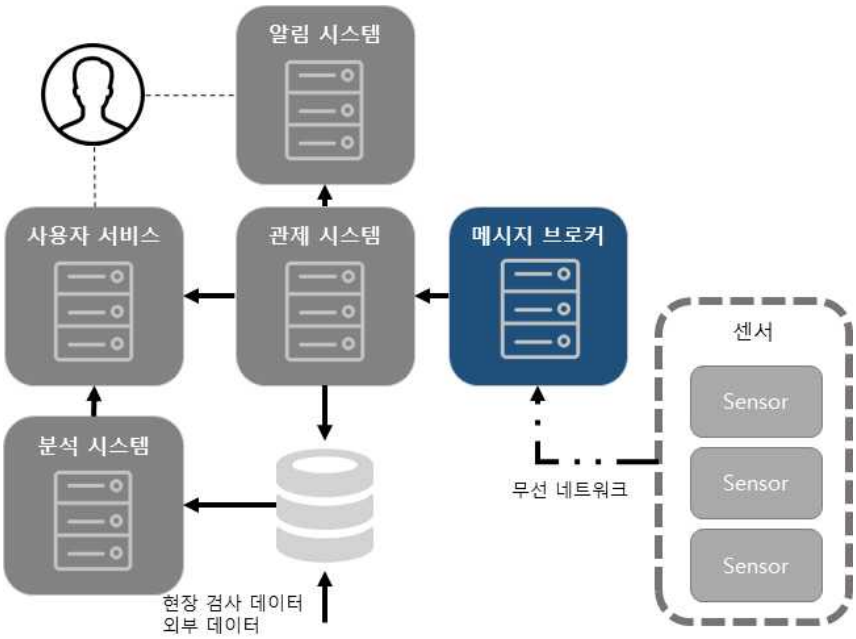
작물 생육 환경 모니터링 시스템은 이러한 문제를 해결하고자 농지에 저 전력 무선 센서 장비를 설치하여 작물의 생육 환경에 대한 모니터링을 통해 이상 상황 발생 시 농가 또는 관련 사용자에게 알림 메시지를 발송하여 최적의 생육 환경을 유지할 수 있도록 지원하며, 수집 된 데이터를 기반으로 생육 환경에 대한 데이터를 분석하여 생육 단계별 맞춤 정보를 제공하는 목적 하에 개발이 되었다.



[그림 3-3] 작물 생육 모니터링 시스템 개념도

[그림 3-3]과 같이 작물 생육 환경 모니터링 시스템은 재배지에 대한 초기 토양 미생물 현황을 분석하고 날씨, 토양에 대한 공공 데이터와 함께 지속적으로 변화하는 작물의 생육 환경에 대한 데이터를 센서를 통해 모니터링하여 현 시점 최적의 상태를 유지할 수 있도록 이상 상태 알림과 각종 병충해 발생 시점 및 조치 방법 알림, 모니터링을 통한 과다한 농약 투입 예방을 지원한다. 시스템의 구성을 보면 [그림 3-4]와 같다.

작물 생육 환경 모니터링 시스템은 클라우드 기반 스마트 팜 구조를 적용하였으며, 작물 생육 환경을 모니터링하기 위해 산도, 염도, 지하온습도, 지상온습도, 조도 데이터를 수집하는 센서장비, 저 전력, 저 품질의 네트워크 통신에서 신뢰성을 보장하며 대량의 데이터에 대해 고속으로 처리가 가능한 고성능 메시지 브로커, 수집된 데이터를 확인하는 모니터링 관제 시스템, 빅데이터 분석 시스템, 알림 시스템, 농가 및 관련 기관에 제공되는 서비스 시스템 등이 필요하다.



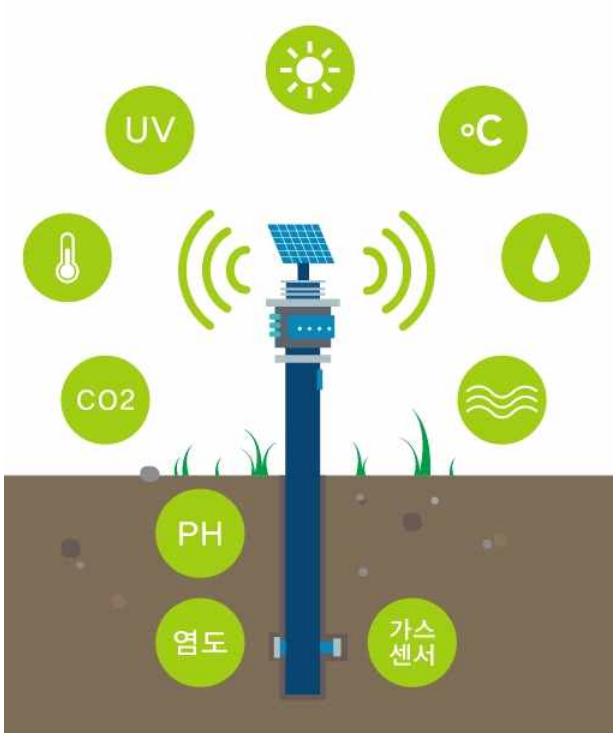
[그림 3-4] 작물 생육 환경 모니터링 시스템 구성도

데이터 수집 센서를 통해 수집된 데이터는 DB에 저장되며 모니터링 관제 시스템은 수집된 데이터를 확인하여 이상이 있는 데이터에 대해 알림 시스템을 통해 농가 또는 기관에 이상 알림 메시지를 발송한다. 이와 별도로 수집된 데이터와 날씨, 토양 미생물 분석 데이터 등을 이용하여 빅데이터 분석하여 생육단계별 필요한 정보와 조치상황을 제공한다.

3.1.2 수집 시스템 구성

작물의 생육 환경을 수집하기 위해 수집 시스템은 센서와 브로커, 모니터링 시스템으로 구성된다. 센서에서 수집된 정보는 브로커로 보내지며 브로커에서 모여진 데이터는 모니터링 시스템으로 전달되어 데이터를 분석, 가공하게 된다.

작물의 생육 환경을 사람의 개입 없이 수집하기 위해서는 센서장비가 필수적이다. 하지만 이러한 센서는 무선 환경에서 동작되기 때문에 지속적으로 전원을 공급하는 방법과 안정적으로 통신하는 기술이 필요하다. [그림 3-5]는 데이터를 수집하기 위한 센서 장비 개요이다.

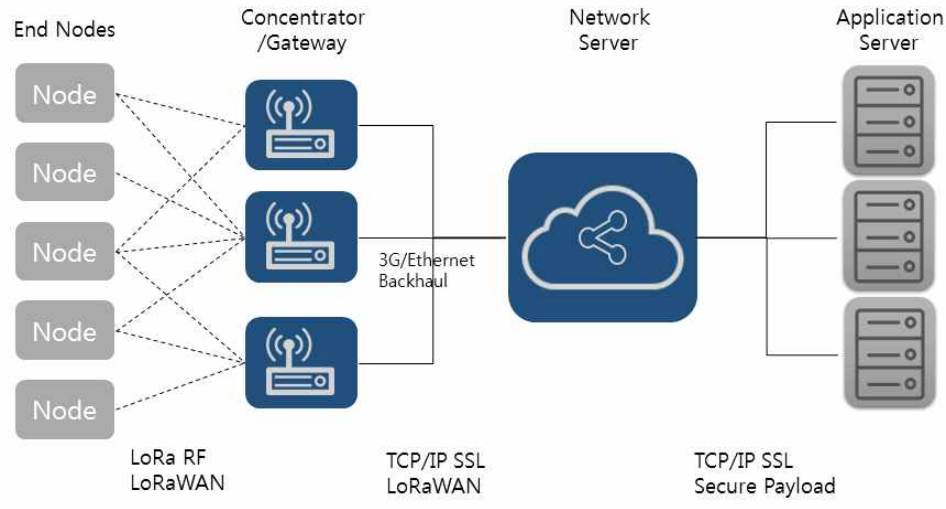


[그림 3-5] 생육 환경 수집 센서 개요

센서장비는 [그림 3-5]와 같은 구성으로 기본적인 작물의 생육 환경 정보 중 토양의 산도, 염도, 온도, 습도와 대기의 온도, 습도, 조도 데이터를 측정한다.

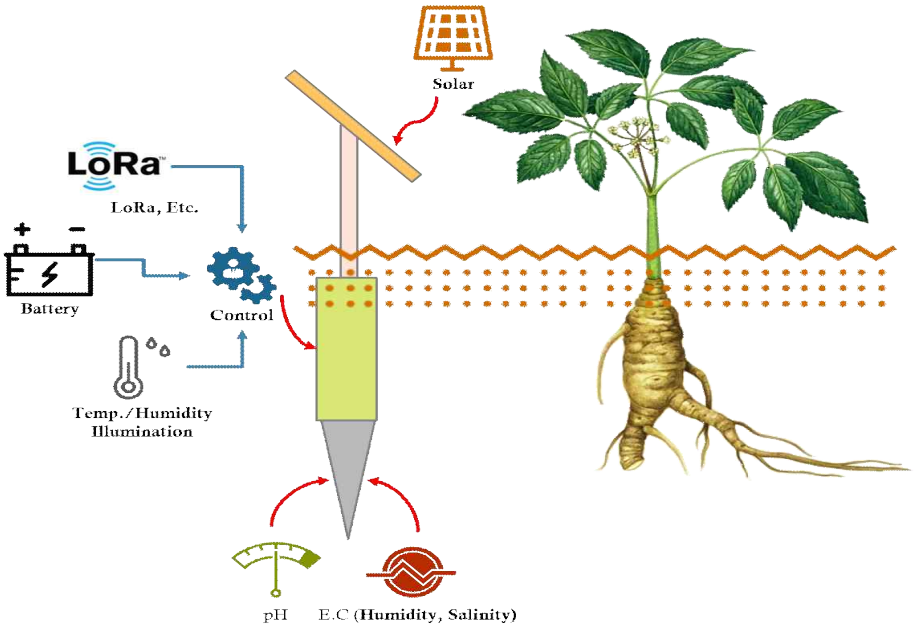
이때 수집된 데이터는 저속, 저 품질의 네트워크 환경을 감안하여 로라(LoRa) 통신을 활용하여 서버에 전달되게 된다. 로라는 다양한 애플리케

이션들을 복수의 네트워크상에서 단일 네트워크 인프라로 연동시키기 위한 기술로, M2M 이동통신 접속을 위한 보완재로써 배터리 기반 센서와 저 전력 애플리케이션에 최적화된 기술이다. LoRaWAN 기술은 수백bps 이하의 매우 낮은 데이터 전송률을 갖고 있는 반면 이동통신 신호 대비 도달 거리가 길며 소규모 기기에서 동작이 가능하다. 또한 매우 낮은 전력 소모로 인해 일부 배터리는 수개월 이상 지속 가능하다. [그림 3-6]은 센서 장비로부터 애플리케이션 서버까지 구성되는 LoRa 네트워크 구조를 나타낸다[32].



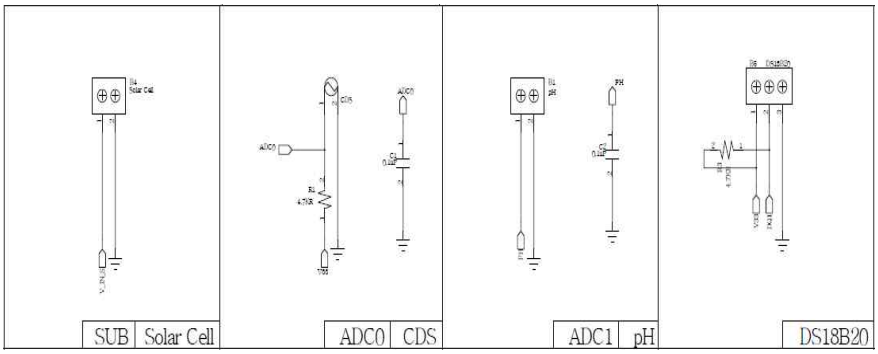
[그림 3-6] 로라 네트워크 구조

또한 센서 장비의 원활한 전원 공급을 통한 운용을 위해 태양 전지를 탑재하여 지속적으로 사용 가능하게 한다. 센서 장비의 전체적인 구성은 [그림 3-7]과 같다.

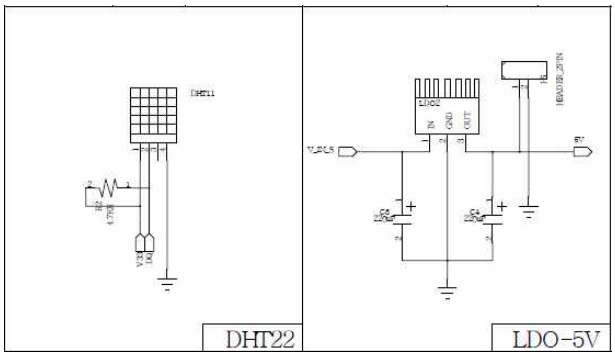


[그림 3-7] 작물 생육 환경 수집 센서 구조

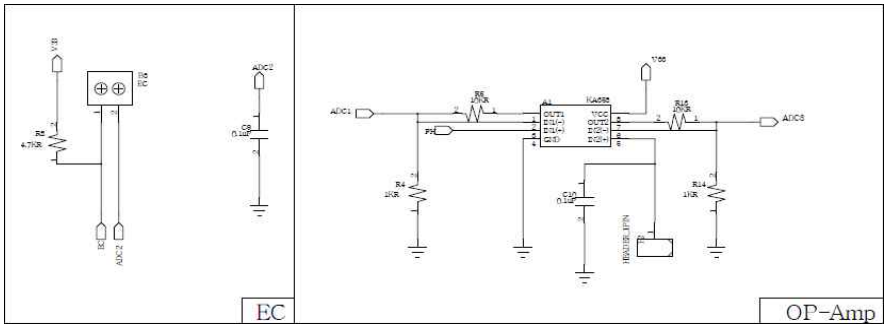
센서 장비의 회로의 구성은 [그림 3-8, 3-9, 3-10]과 같다.



[그림 3-8] 센서 장비 지하부

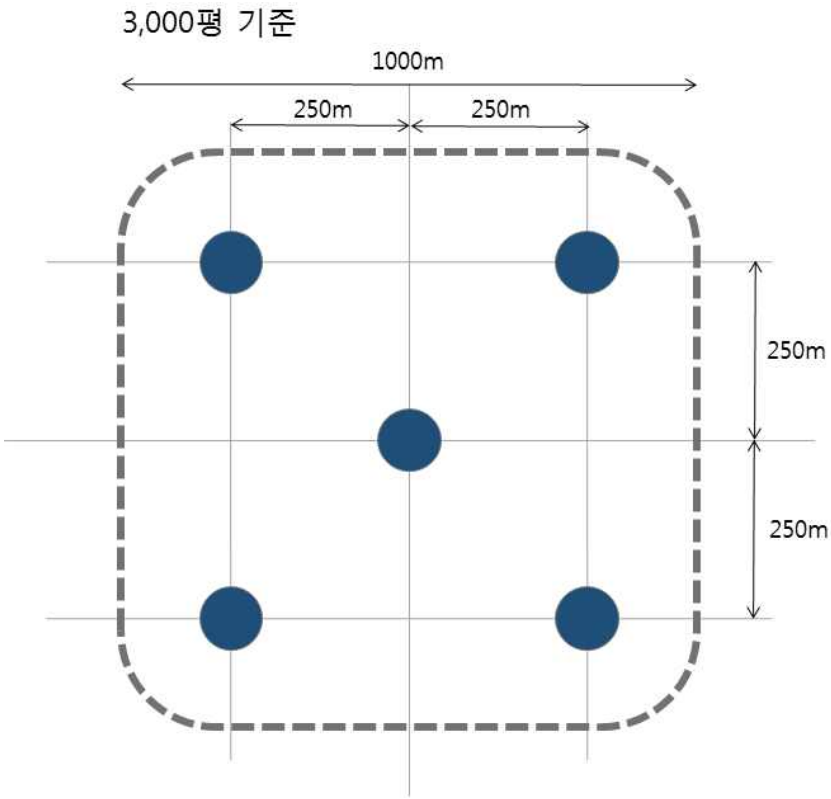


[그림 3-9] 센서 장비 지상부



[그림 3-10] 센서 장비 구동부

이러한 센서 장비를 [그림 3-11]과 같이 농경지 3,000평 기준으로 250m 간격으로 배치하여 위치에 따라 달라지는 토양, 대기 데이터를 수집하도록 한다.

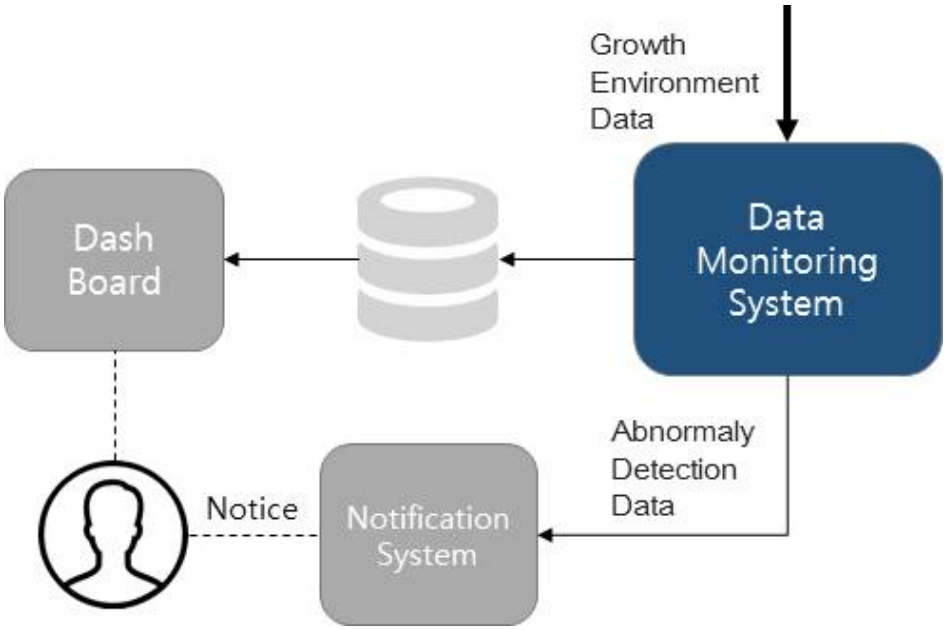


[그림 3-11] 농지 센서 배치 구조

각 센서 장비는 각 상황에 맞게 수집 주기를 설정하고 생육 환경에 대해 수집하여 브로커에 데이터를 전송하게 된다. 또한 서버에서 요청할 경우 센서 장비의 통신주기, 수집주기, 동작여부, 배터리 잔량 등도 전송하게 된다.

브로커로 전송된 수집 데이터는 브로커와 연결되어있는 모니터링 관제 시스템으로 전달된다. 이때 모니터링 관제 시스템은 브로커와 연결되어 두 가지 기능을 제공한다.

◦ [그림 3-12]와 같이 센서로부터 수집된 데이터는 관제 시스템에서 데이터의 이상 유무를 검출하고 이상 알림 시스템을 이용하여 이상 알림을 발송.



[그림 3-12] 생육 환경 데이터 분석 및 이상감지

◦ [그림 3-13]과 같이 재배지에 설치된 센서에 대해 모듈의 상태, 통신주기, 수집주기, 동작 여부, 배터리 잔량 등의 정보를 확인하기 위해서 센서 노드에 상태 정보를 회신하도록 요청하며 센서의 정보를 수정.

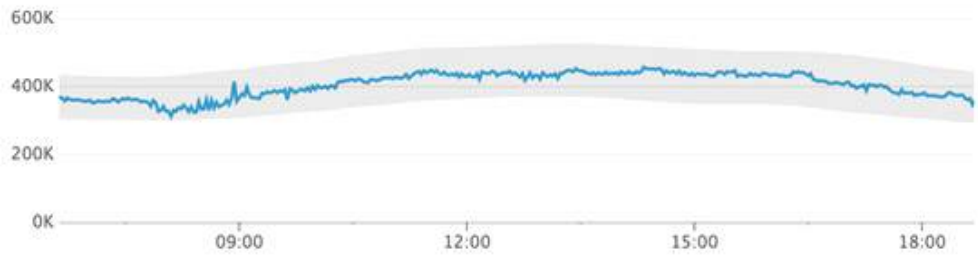
[그림 3-13] 센서 정보 수정 및 요청

생육 환경에 대한 관제를 위한 관제 시스템은 브로커와 지속적으로 연결되어 있어야 하며, 모든 센서의 데이터를 확인 할 수 있어야 한다. 이를 위해 관제 시스템 또한 MQTT 프로토콜을 이용해야 하며 토픽에 # 또는 + 와일드카드를 이용하여 센서에서 Publish 하는 데이터에 대하여 받을 수 있도록 구독을 요청해야 한다. 또한 센서는 센서 정보 수정 및 요청에 응답하기 위해 자신의 토픽에 대해 구독하고 있어야 한다.

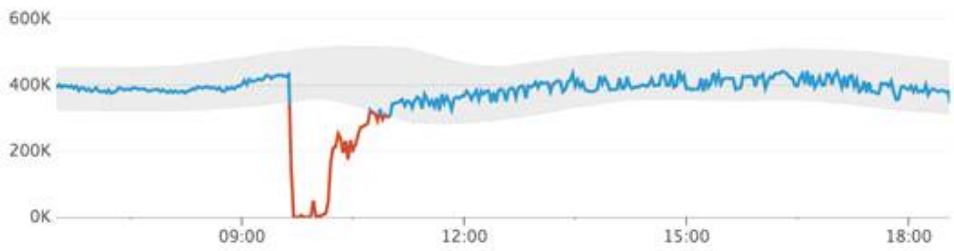
수집된 데이터는 관제 시스템을 통해 이상 유무를 확인하여 알림을 발송한다. 데이터 값의 이상 유무에 대한 판단을 위해 두 가지 방법을 병행한다.

- 특정 범위 이상 값 검사
  수집 데이터 중 미리 정의된 범위 이상의 값이 측정되는 경우이며 일반적인 값의 한계치를 넘는 경우 이상 값으로 판단.
- 변칙 데이터 검사
  변칙적인 데이터를 확인하기 위해 이상 감지(Anomaly Detection) 분석 기법이 사용.

[그림 3-14]와 [그림 3-15]는 이상감지 기법의 예를 나타낸다.



[그림 3-14] 이상감지 기법 - 정상 데이터



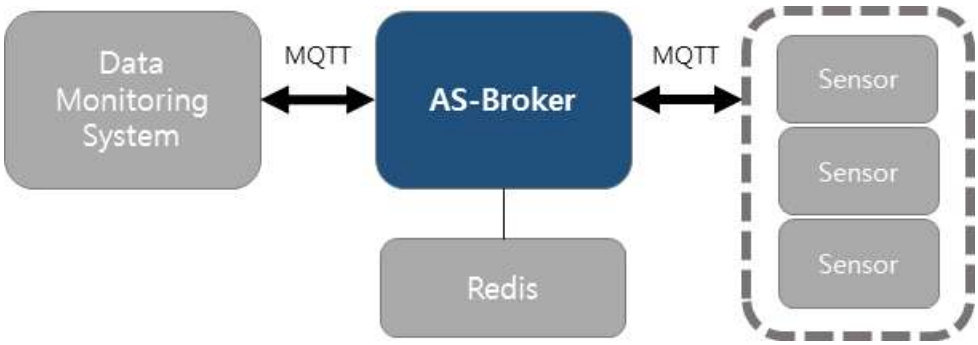
[그림 3-15] 이상감지 기법 - 비정상 데이터

수집된 데이터를 시계열 데이터베이스에 저장하고 실제 데이터 값에 대해 확인 할 수 있도록 가시화를 지원한다. 이를 위해 오픈소스 기술인 Grafana를 이용하여 시계열 데이터에 대한 가시화를 진행한다. [그림 3-16]은 Grafana의 결과 화면이다.

[그림 3-16] 관제 대시보드

3.2 비동기 IoT 브로커 설계 및 구현

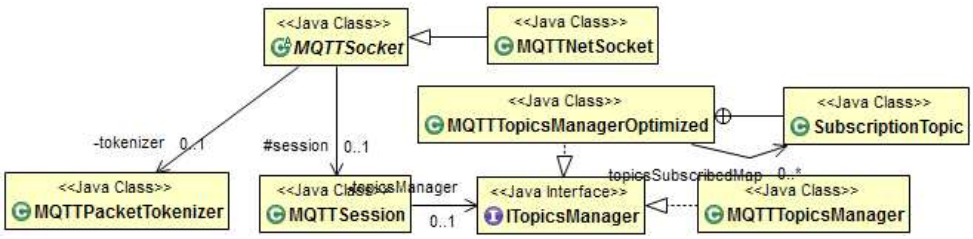
본 수집 시스템에서 사용되는 브로커는 전국 각지에 설치된 대량의 센서를 이용하여 각 농지의 작물 생육 환경을 모니터링하기 때문에 저속, 저 품질의 무선 네트워크를 이용하는 것과 대량의 센서에서 빠르게 수집되는 데이터로 인해 발생된 빅데이터로 인한 병목 현상으로 인해 데이터의 손실 및 처리 지연이 발생될 수 있음을 감안해야 한다. 이를 위해 센서와 브로커 간의 신뢰성 있는 통신을 지원하는 메시지 프로토콜을 적용되어야하며, 대량 트래픽 환경에서 안정적이며 신속한 데이터 처리를 할 수 있는 비동기 처리 기술이 필요하다. 이를 위해 비동기 처리를 지원하는 프레임워크 중 Java 기반의 Vert.x를 사용으로 하였으며, 사물인터넷 표준 규격인 oneM2M에서 지원하는 프로토콜인 CoAP와 MQTT 중 MQTT 프로토콜을 사용하여 구현하였다. [그림 3-17]은 AS-Broker의 블록 다이어그램이다.



[그림 3-17] AS-Broker 블록 다이어그램

센서 또는 관제 시스템에서 AS-Broker로 전달된 Publish 메시지는 Vert.x의 Verticle에 전송되어 서버에 접속 중인 연결에 해당 메시지를

브로드캐스트로 발송한다. 각 연결은 별도의 세션을 생성하여 브로드 캐스팅한 콘텐츠를 확인하여 실제 자신의 Subscribe 토픽에 해당하는지 확인하며 토픽 리스트에 부합하는 메시지이면, Vert.x의 커넥션 Write를 통해 해당 메시지를 발송한다[25]. AS-Broker의 클래스 다이어그램은 [그림 3-18]과 같다.



[그림 3-18] AS-Broker 클래스 다이어그램

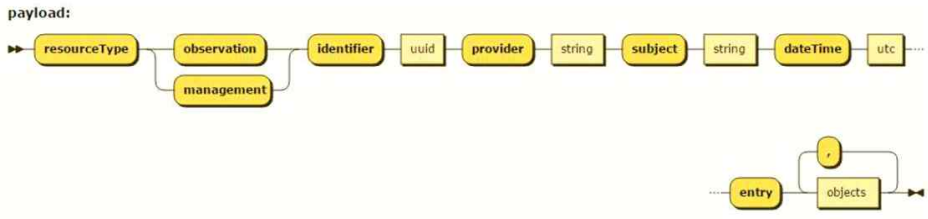
이때 브로커를 이용해 통신한 결과를 기록하기 위해 DB가 사용된다. 빠르게 처리되는 통신 결과를 저장해야 하기 때문에 DB의 성능이 중요하다. 이를 위해 고성능 Key-Value 저장소인 Redis를 함께 사용하였다. Redis의 성능을 확인하기 위해 AWS c4.8xlarge(CPU 36Core, 60G Memory) 서버에서 벤치마크 성능 테스트를 했을 때 [표 3-1]과 같다.

[표 3-1] Redis 벤치마크 성능 테스트 결과

Method	Result
SET	436,681.22 Requests per Second
GET	645,161.31 Requests per Second

AS-Broker는 센서로부터 전달된 데이터를 관제 시스템에게 전달해주

는 역할과 관제 시스템으로부터 센서에 정보를 요청하는 두 가지 역할을 수행한다. 이를 위해 AS-Broker는 Json 형식의 두 가지 Payload 구조를 제공하며 Payload의 기본 구조는 다음 [그림 3-19]와 같다.

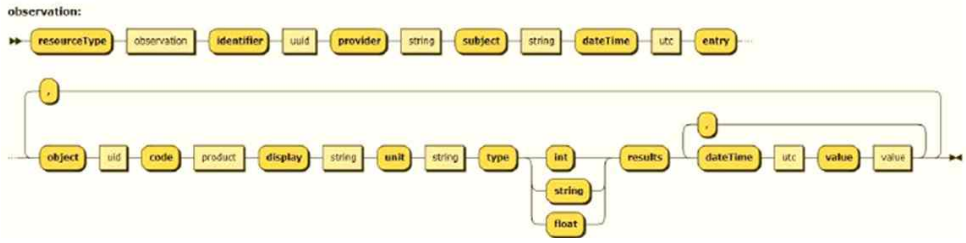


[그림 3-19] AS-Broker 기본 Payload 구조 레일로드 다이어그램

[그림 3-19]와 같이 기본구조는 Payload의 유형을 나타내는 ResourceType, 중복 문서 수신을 방지하기 위한 Identifier, 문서의 발행 기관을 나타내는 Provider, 생성 주체를 나타내는 Subject, 문서의 생성 시각을 나타내는 DateTime, 본문에 해당하는 Entry로 구성된다. 이와 같은 기본 구조를 바탕으로 두 가지 타입의 Payload 구조를 제공한다.

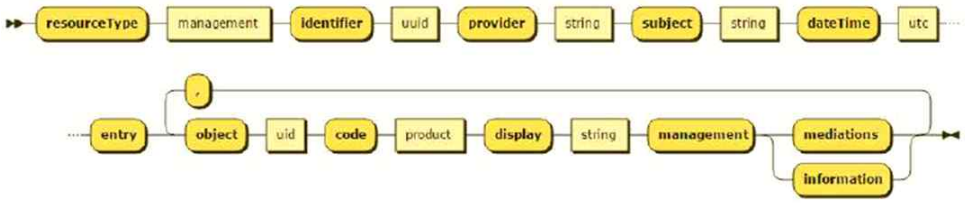
- Observation 타입
- Management 타입

Observation 타입의 Payload로 들어온 데이터는 센서에 대한 관제에 사용되며 데이터를 저장 후 분석을 통한 예측 정보를 산출한다. 또한 모니터링 그래프 출력을 위한 데이터 처리를 수행한다. Observation 타입의 구조는 [그림 3-20]과 같다.



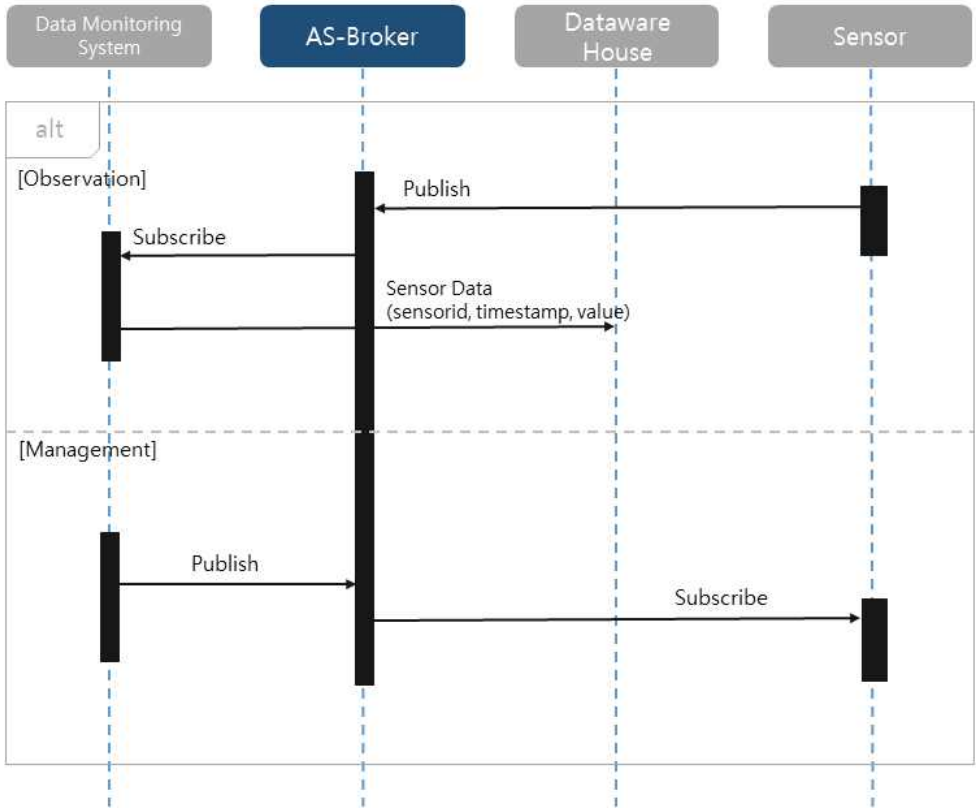
[그림 3-20] Observation 타입 Payload 구조 레일로드 다이어그램

Management 타입은 두 가지 용도로 이용된다. 첫 번째는 원격지에 배치된 센서의 각 모듈의 상태를 관제하기 위해 특정 센서에게 상태 정보를 회신하도록 요청하는 것이다. 두 번째는 센서모듈의 동작에 필요한 파라미터 성격의 데이터 혹은 데이터 수집 스케줄 명령 등을 전송하는 것이다. Management 타입의 구조는 [그림 3-21]과 같다.



[그림 3-21] Management 타입 Payload 구조 레일로드 다이어그램

이러한 Payload를 이용하여 통신할 때 AS-Broker의 동작 순서는 [그림 3-22]와 같이 진행된다.



[그림 3-22] AS-Broker 동작 순서도

[그림 3-22]와 같이 센서에서 수집된 데이터에 대하여 관제 시스템을 통해 이상 유무를 확인하도록 구성한다. 이때 센서로부터 온 데이터의 경우 신뢰성 있는 메시지 전달이 필요하므로 QoS1 또는 QoS2를 지정하여 사용하도록 한다.

3.3 실험 환경 구성

실험을 위해 아마존 웹 서비스(AWS, Amazon Web Service)[33]를 이용하여 가상의 CentOS7 서버를 [표 3-2]와 같이 구성한다.

[표 3-2] 실험 서버 성능

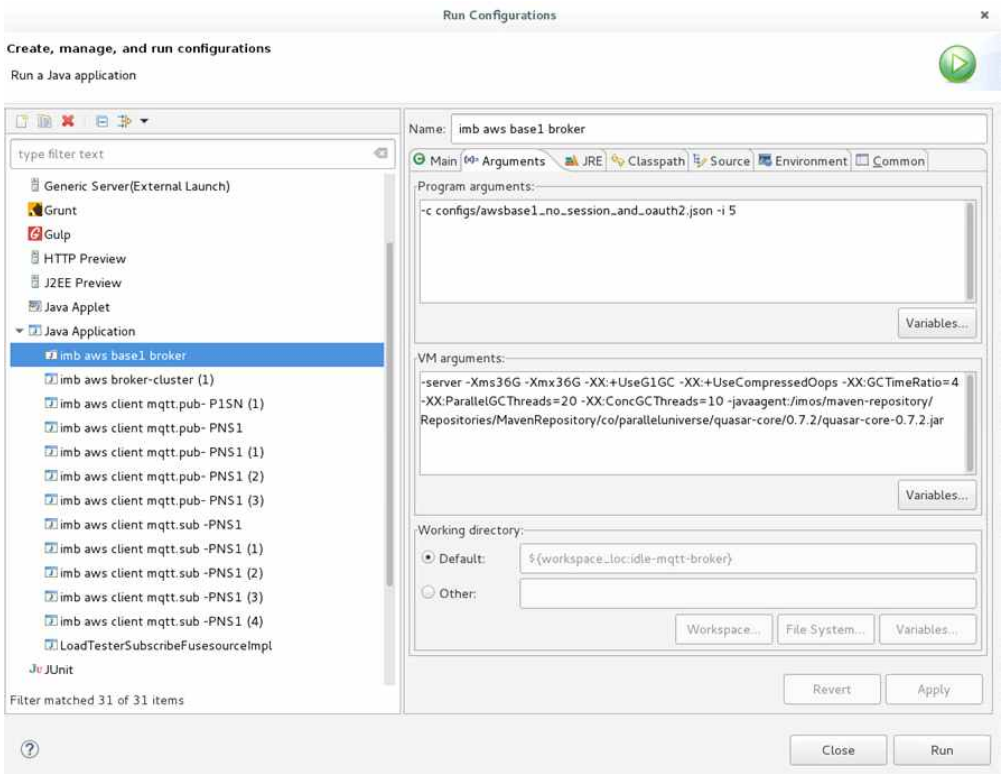
Instance Type	ECUs	vCPUs	Memory	Network Performance	Storage
c4.8xlarge	132	36	60G	10 G	GP2 15G

[표 3-2]와 같은 사양의 인스턴스를 선택 후 고정 IP와 외부에서 접근이 가능하도록 보안 그룹을 설정 후 구동시킨다. 서버 시스템 구성은 다음과 같다.

[그림 3-23] 서버 시스템 구성

[그림 3-23]과 같이 경량 가상머신인 도커(Docker)를 설치하고 도커를 이용하여 브로커에서 처리된 내용을 저장하기 위한 Redis와 서버 성능

모니터링을 위해 Grafana, 시계열 DB인 InpluxDB를 설치한다. 또한 Jdk8을 설치하고 그 위에 AS-Broker를 실행하도록 한다. 이외에 서버에서 대량의 커넥션이 생길 시 리눅스에서 기본적으로 제공되는 File Open 개수로 인해 오류가 생길 수 있기 때문에 이를 방지하기 위하여 /etc/security/limits.conf 파일의 soft stack과 hard stack 부분을 수정하여 준다. 또한 외부에서 접근 가능하도록 방화벽을 열어준다.



[그림 3-24] AS-Broker 동작 설정 화면

서버의 기본 세팅이 끝나면 [그림 3-24]와 같이 AS-Broker를 실행시킨다. 본 실험에서는 원활한 실험 진행을 위해 자바 개발 툴인 이클립

스를 이용하여 실행시켰다.

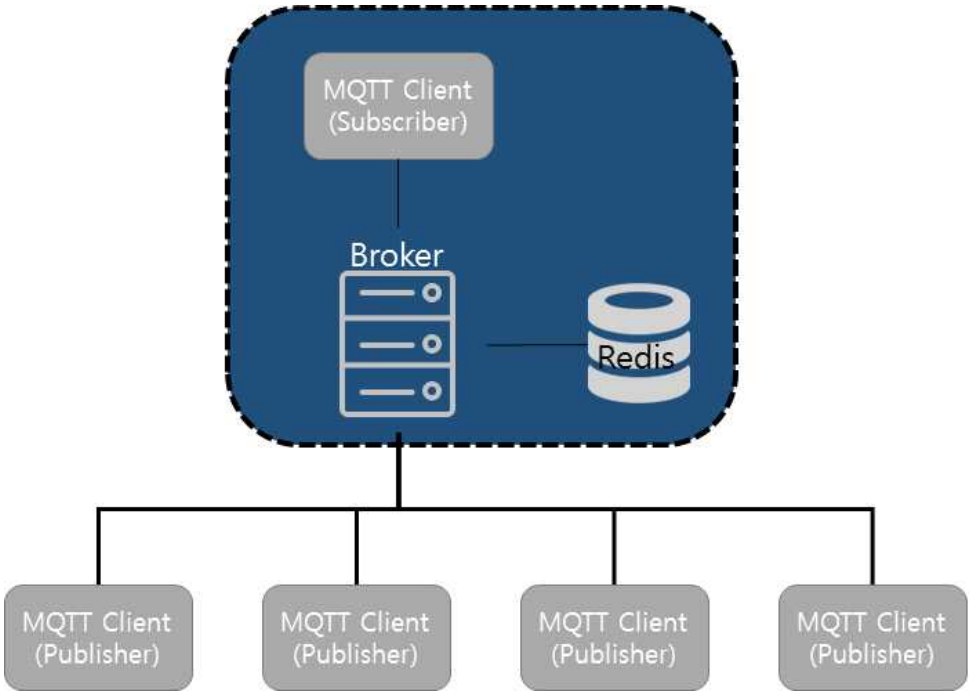
AS-Broker의 성능을 실험하기위해 센서장비를 대량으로 연결시켜야 하지만 대량의 센서장비를 만들어서 실험하기에는 비용과 환경의 제한이 있으므로 센서장비와 같이 MQTT로 통신할 수 있는 MQTT Client를 만들어 서버 장비를 이용하여 대량으로 연결시키도록 한다.

클라이언트를 동작시키기 위해 서버와 마찬가지로 AWS를 이용하여 가상의 CentOS7 서버를 다음 [표 3-3]과 같이 구성한다.

[표 3-3] 실험 클라이언트 성능

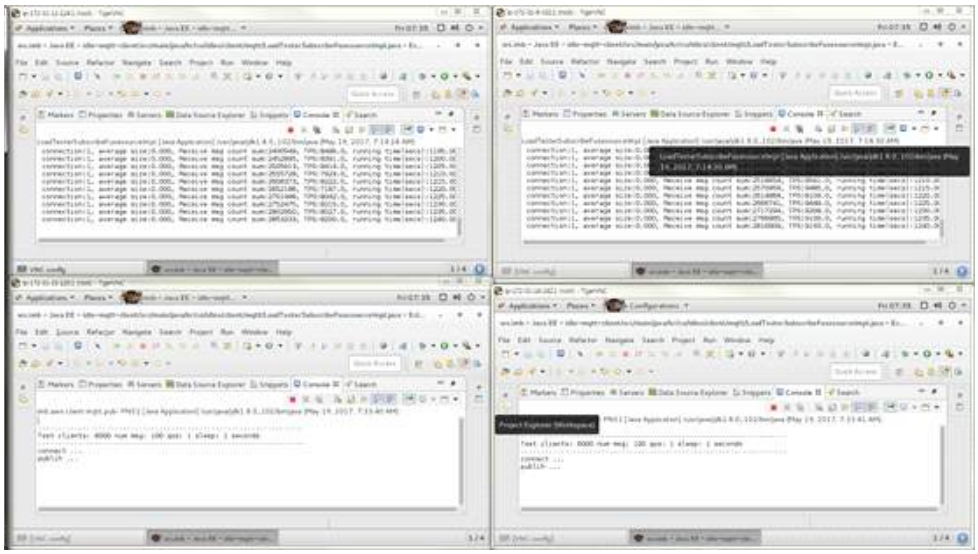
Instance Type	ECUs	vCPUs	Memory	Network Performance	Storage
c4.2xlarge	31	8	15G	High	GP2 15G

MQTT Client는 MQTT를 JAVA 기반의 Paho 라이브러리를 이용하여 구현하였다. Paho 라이브러리는 C, C++, Java, Android, Javascript, Python 등 여러 언어를 지원하며, 본 테스트에서는 Java를 이용하여 구현하였다. 이를 위해 Jdk8을 설치 후 클라이언트 프로그램을 실행 시키도록 한다. 이와 같은 클라이언트 구성 서버를 동일하게 4대 구동 시키도록 한다. 실험 환경 구성은 다음과 같다.



[그림 3-25] 실험 환경 구성

[그림 3-25]와 같이 단일 서버 장비에 AS-Broker와 Subscriber를 실행시키고 총 4대의 클라이언트 장비를 이용하여 각 장비 당 5000개의 Publisher를 실행시킨다. 각 Publisher마다 100Byte Payload를 100개씩 자동으로 생성하여 AS-Broker에 전송하도록 한다. 본 실험에서는 각 클라이언트마다 VNC 서버를 설치하여 화면을 구성하였으며, 브로커 서버와 마찬가지로 원활한 실험 진행을 위해 자바 개발 툴인 이클립스를 이용하여 실행시켰다. 실험을 위해 구성한 화면은 [그림 3-26]과 같다.



[그림 3-26] 클라이언트 구성 화면

제 4 장 실험 결과 및 분석

4.1 실험 결과

본 실험은 앞서 구현된 AS-Broker의 대량 트래픽 환경에서의 초당 요청 처리 능력(tps)에 대해 알아보기 위해 총 4가지 실험이 진행되었다.

- 대량 커넥션 상황에서 AS-Broker의 인스턴스 수에 따른 성능변화
- 단일 Subscriber에서 QoS에 따른 브로커의 성능변화
- 동일 토픽을 여러 Subscriber에서 구독할 경우 브로커의 성능변화
- 여러 토픽을 나눠 Subscriber에서 구독할 경우 브로커의 성능변화

실험 결과는 [표 4-1]부터 [표 4-4]와 같으며 결과 표 항목은 다음과 같다. Instance는 Vert.x를 통해 생성되는 Verticle Instance로 클러스터링된 브로커의 수를 의미한다. Sub은 Subscriber로 가상 관제시스템을 의미하며, Pub은 Publisher로 가상 센서 장비이며 생성된 클라이언트 수를 나타낸다. Sub QoS와 Pub QoS는 Subscriber와 Publisher의 QoS이며 Payload는 Publisher에서 발송되는 메시지의 사이즈와 개수를 나타낸다. TPS(Transaction Per Second)는 초당 요청 처리 능력을 나타낸다.

[표 4-1] 단일 Subscriber에서 QoS에 따른 성능변화

Instance	Sub	Pub	Sub Qos	Pub Qos	Payload	TPS
5	1	20000	0	1	100Byte * 100	25k
5	1	20000	1	1	100Byte * 100	20k

단일 Subscriber 구독할 경우 QoS에 따른 성능변화는 [표4-1]과 같았다. Subscriber의 QoS가 0일 때 처리량에 대한 그래프는 [그림 4-1], QoS가 1일 때 처리량에 대한 그래프는 [그림 4-2]와 같다.

[그림 4-1] 단일 Subscriber QoS 0 경우

[그림 4-2] 단일 Subscriber QoS 1 경우

[그림4-1]은 Subscriber의 QoS가 0일 때 처리량에 대한 그래프로 2만 커넥션에서 생기는 트래픽에 대해 25000tps 정도의 처리 능력을 보이는 것을 확인할 수 있다. [그림 4-2]는 QoS가 1일 때 처리량에 대한 그래프로 2만 커넥션에서 발생되는 트래픽에 대해 20000tps정도의 처리 능력을 보인다.

[표 4-2] 브로커 인스턴스 수에 따른 성능변화

Instance	Sub	Pub	Sub Qos	Pub Qos	Payload	TPS
5	1	20000	0	1	100Byte * 100	25k
3	1	20000	0	1	100Byte * 100	22k
1	1	20000	0	1	100Byte * 100	20k

AS-Broker 인스턴스 수에 따른 성능변화는 [표4-2]과 다음과 같았다. Instance가 5인 경우는 [그림 4-1]의 내용과 같기 때문에 생략하며, Instance가 3인 경우의 처리량에 대한 그래프는 [그림 4-3], Instance가 1인 경우 처리량에 대한 그래프는 [그림 4-4]과 같다

[그림 4-3] 브로커 인스턴스가 3개일 경우

[그림 4-3]은 3개의 브로커 인스턴스를 클러스터링 했을 경우로 22000tps 정도의 성능을 보인다.

[그림 4-4] 브로커 인스턴스가 1개일 경우

[그림 4-4]는 1개의 브로커 인스턴스를 이용하여 성능을 확인한 경우

로 약 20000tps의 성능을 확인하였다.

[표 4-3] 여러 Subscriber에서 동일 토픽 구독에 따른 성능변화

Instance	Sub	Pub	Sub Qos	Pub Qos	Payload	TPS
5	1 (1 topic)	20000	0	1	100Byte * 100	25k
5	10 (1 topic)	20000	0	1	100Byte * 100	52k
5	100 (1 topic)	20000	0	1	100Byte * 100	50k

[표4-3]는 동일 토픽을 여러 Subscriber에서 구독할 경우 Subscriber의 개수에 따른 성능변화를 나타낸다. Subscriber의 개수가 1인 경우는 마찬가지로 [그림 4-1]와 같기 때문에 생략하며, Subscriber의 개수가 10인 경우의 처리량에 대한 그래프는 [그림 4-5], Subscriber의 개수가 100인 경우 처리량에 대한 그래프는 [그림 4-6]과 같다.

[그림 4-5] 동일 토픽을 Subscriber 10개가 구독할 경우

[그림 4-5]는 2만 커넥션에서 생성되는 데이터를 하나의 토픽을 통해 수집해 10개의 Subscriber가 함께 구독하여 처리하는 경우이며 이 경우 약 52000tps 정도의 성능을 보였다. [그림 4-6]에서는 동일한 상황에서 100개의 Subscriber로 함께 구독하여 처리하였으며 약 50000tps의 성능을 볼 수 있다.

[그림 4-6] 동일 토픽을 Subscriber 100개가 구독할 경우

여러 토픽을 여러 Subscriber에서 각각 1개씩 나눠서 구독할 경우 토픽과 Subscriber의 개수에 따른 성능변화는 [표4-4]과 다음과 같았다.

[표 4-4] 여러 Subscriber에서 각기 다른 토픽 구독에 따른 성능변화

Instance	Sub	Pub	Sub Qos	Pub Qos	Payload	TPS
5	1 (1 topic)	20000	0	1	100Byte * 100	25k
5	2 (2 topic)	20000	0	1	100Byte * 100	40k
5	4 (4 topic)	20000	0	1	100Byte * 100	56k

토픽도 1개에 1 개의 Subscriber인 경우는 [그림 4-1]과 같기 때문에 생략하며, Subscriber 2개가 토픽 2개에 대해 각각 처리한 경우의 처리량에 대한 그래프는 [그림 4-7]과 같으며, Subscriber 4개가 토픽 4개에 대해 각각 처리한 경우 처리량에 대한 그래프는 [그림 4-8]과 같다.

[그림 4-7] 각기 다른 토픽을 Subscriber 2개가 구독할 경우

[그림 4-7]는 2만 커넥션에서 생성되는 데이터를 2개의 토픽을 통해 수집해 토픽을 각각 나누어 2개의 Subscriber가 각기 구독하여 처리하는 경우이며 이 경우 약 40000tps 정도의 성능을 보였다.

[그림 4-8] 각기 다른 토픽을 Subscriber 4개가 구독할 경우

[그림 4-8]은 [그림 4-7]과 같은 상황에서 4개의 토픽을 통해 수집해 토픽을 각각 나누어 4개의 Subscriber가 각기 구독하여 처리하는 경우이며 이 경우 약 56000tps 정도의 성능을 보이는 것을 알 수 있었다.

4.2 결과 분석

실험 결과를 분석하면 다음과 같다. QoS 0은 손실 여부 체크 없이 메시지를 최대 한번 발송하는 반면 QoS 1은 적어도 한번 이상 보내는 방식으로 메시지가 잘 도착되었을 때 ACK 메시지를 전송한다. 이러한 동작 방식 때문에 단일 Subscriber에서 QoS에 따른 AS-Broker의 성능변화의 결과 [표 4-1]과 같이 QoS 0으로 발송할 때 약 25000tps인 반면 QoS 1로 발송한 경우 약 20000tps로 5000tps 정도의 성능의 차이를 보이는 것을 알 수 있었다.

또한 AS-Broker의 인스턴스 수에 따른 성능변화의 결과 [표 4-2]와 같이 5개의 인스턴스를 클러스터링 했을 때 25000tps이고 3개의 인스턴스를 클러스터링 했을 때 22000tps, 1개로 동작했을 때 18000tps로 각각 3000tps정도의 차이가 나므로 1개의 인스턴스를 추가로 클러스터링시키는 경우 1500tps에서 2000tps 정도 성능이 향상되는 것을 알 수 있었다.

2만 커넥션에서 1개의 토픽을 통해 보내온 데이터를 여러 Subscriber에서 동시에 구독할 경우 AS-Broker의 성능변화에 대해서는 [표 4-3]와 같았으며 Subscriber 1개일 때 25000tps에서 Subscriber가 10개일 때 52000tps로 증가되었지만 Subscriber가 100개일 때 50000tps로 떨어지게 되는 것을 볼 수 있다. 이를 통해 동일 토픽에 대해 Subscriber가 적정 개수 이상에서는 더 이상 tps가 증가하지 않는 것을 알 수 있다.

마지막으로 2만 커넥션에서 데이터를 보낼 때 토픽을 나눠 보내고 한 토픽만 구독하는 Subscriber를 각각 두는 경우 [표 4-4]와 같다. 1개의 토픽에 데이터를 보내며 Subscriber 1개가 구독 할 때 25000tps, 토픽을 2개로 나눠 각각 2개의 Subscriber로 1만개의 Publisher가 100Byte Payload 100개 씩 메시지를 보냈을 때 40000tps, 토픽을 4개로 나눠서

4개의 Subscriber로 각각 5천개의 Publisher가 메시지를 보냈을 때 56000tps로 나타나는 것을 볼 수 있었다. 해당 결과를 통해 데이터 처리를 4개의 토픽으로 나눠서 하는 경우에 1개일 때 보다 2배 이상 더 좋은 성능을 내는 것을 알 수 있다.

결과를 종합해 봤을 때 QoS 0으로 발송할 때보다 QoS 1로 발송한 경우 약 5000tps 정도의 성능이 증가했으며, 5개의 인스턴스를 클러스터링 했을 때 1개로 동작했을 때보다 6000tps 증가했음을 알 수 있었다. 또한, 동일 토픽에 대해 10개의 Subscriber가 처리하는 경우나 100개의 Subscriber가 처리하는 경우나 성능은 비슷하였으며, 커넥션에서 보내지는 데이터를 4개 토픽으로 나눠 4개의 Subscriber가 각각 처리하는 경우가 1개의 토픽으로 1개의 Subscriber가 처리하는 경우보다 2배 이상 차이가 나는 것을 확인하여 최고의 성능을 보였다.

제 5 장 결론

본 논문을 통해 작물 생육 환경을 모니터링 하기 위한 수집 시스템을 구성하였다. 이를 위해 저속, 저 품질의 네트워크 환경에서 동작하는 대량의 센서를 통해, 발생되는 트래픽 환경에서 안정적으로 데이터를 처리할 수 있는 고성능의 브로커를 설계, 구현하였으며 성능 실험을 진행하였다. 또한 작물 생육 환경 정보를 수집하기 위한 센서와 수집된 데이터에 대해 모니터링 관제하는 시스템에 대해 언급하였다.

본 논문의 중심이 되는 비동기 IoT 브로커(AS-Broker)의 경우, 센서 특유의 네트워크 환경에 신뢰성 있는 메시지 전송을 위해 사물인터넷 표준인 oneM2M에서 지원하는 MQTT 프로토콜이 적용되었으며, 센서 장비의 대량 커넥션에서 발생되는 트래픽 환경에서 빠르게 수집된 데이터를 안정적이고 신속하게 처리 할 수 있도록 비동기 처리 기술인 Vert.x가 적용되었다. 또한 센서 장비와 관제 시스템 간의 통신을 위한 Payload 구조가 정의되었다.

아마존 웹 서비스(AWS)의 c4.8xlarge 타입 인스턴스를 이용하여 AS-Broker의 실험 환경을 구성하고 성능을 테스트 한 결과, 2만 커넥션의 가상 센서 장비(Publisher)에서 발생되는 요청을 가상 관제 시스템(Subscriber)이 처리할 때, 5개의 브로커 인스턴스를 클러스터링 했을 때 1개로 처리 할 때 보다 7000tps 성능이 향상되었으며, 동일 토픽에 대해 Subscriber 1개만 구독할 때보다 10개 구독했을 때 tps가 증가하지만 10개에서 100개로 증가시켰을 때는 더 이상 증가하지 않는 것을 알 수 있었다. 또한 보내지는 데이터를 4개의 토픽으로 나눠 4개의 Subscriber가 각각 처리 하였을 때 56000tps의 성능을 보여 2개의 토픽을 2개가 처리할 때 보다 16000tps 증가하는 것으로 확인되었다.

본 실험을 통해 2만 커넥션에서 5개의 브로커 인스턴스를 클러스터링 하고 보내지는 데이터를 4개의 토픽으로 나눠 4개의 Subscriber가 각각 처리 하였을 때 가장 좋은 성능을 보인 것을 알 수 있었다. 이를 통해 서버의 성능에 맞게 브로커 인스턴스를 추가로 클러스터링 하여 구성하고, 지역별 또는 다른 방식으로 토픽을 나누어 처리할 수 있도록 특정 토픽만 모니터링하는 관제 시스템을 구성하면 작물 생육 환경 모니터링 시스템에서 수집되는 환경 정보 데이터에 대해 신뢰성 있고 안정적으로 처리가 가능한 데이터 수집 시스템으로 사용 가능할 것으로 보인다.

본 논문을 통해 AS-Broker를 이용한 작물 생육환경 모니터링 시스템을 구축하여 생육 환경을 확인, 분석하고, 작물이 건강하게 성장하기 위한 최적의 상태를 유지할 수 있도록 지원하므로 작물 생산성 확보를 통한 국내 농가에 경제적 이익을 줌으로써 국내 농촌이 갖고 있는 문제들을 해결해 나가는데 기여할 수 있다.

또한, 본 논문에서 제안한 AS-Broker는 작물 생육 환경 모니터링 시스템뿐만 아니라 4차 산업혁명의 스마트 팩토리, 스마트 물류, 스마트 시티, 스마트 의료, 스마트 웨어러블 등의 다양한 스마트 관련 사업군에 활용 가능하다.

참 고 문 헌

[1] "농림어업조사 결과보도자료", 통계청, 2016, p1, p32

[2] "농림축산식품주요통계", 농림축산식품부, 2016, p23

[3] “중소·중견기업 기술로드맵 2017-2019, 산업/일반기계시스템”, 중소기업청, 2017 1, pp.25-68

[4] 김철영, “스마트 팜(Smart Farm)산업 – 농업과 ICT의 융합을 통한 고부가가치 6차 산업으로 육성 필요”, 현대able Daily, 2016. 8. 30, pp1-6

[5] 김연중 외 3명, “스마트 팜 운영실태 분석 및 발전방향 연구” 한국농촌경제연구원 보고서, 2016, p11

[6] 정희창, “클라우드 기반 농식품 ICT 융합 표준화 동향”, 한국정보통신 기술협회 TTA 저널 Vol.169, 2017.01, pp98-99

[7] IoT standard, http://www.onem2m.org/

[8] 박기현 외 3명, “oneM2M 통신 프로토콜 기반 사물인터넷 시스템 개발”, Asia-pacific Journal of Multimedia Services Convergent with Art, Humanities, and Sociology Vol.6, No.3, March 2016, pp 42-43

[9] 김기영, “oneM2M 사물인터넷 서비스 플랫폼 표준화 현황”, 한국정보통신 기술협회 TTA 저널 Vol.155, 2014.09, p41

[10] MQ Telemetry Transport, http://mqtt.org

[11] MQTT Version5.0 Working Draft10, https://www.oasis-open.org/committees/download.php/59741/mqtt-v5.0-wd10.html

[12] Constrained Application Protocol, http://coap.technology/

[13] Constrained Application Protocol (CoAP) draft-ietf-core-coap-18, https://tools.ietf.org/html/draft-ietf-core-coap-18

[14] 장영환, 심재성, 박석천, “IoT 기반 저전력·경량 프로토콜 표준화 분석”, 한국정보통신학회논문지 vol.20 no.10, Oct 2016, pp1896-1900

[15] 황현천, "MQTT 프로토콜 기반의 신뢰적인 메시지 전송 시스템의 설계 및 구현", 한국방송통신대학교 대학원 석사학위논문, 2016, pp4-13

[16] MQTT V3.1 Protocol Specification, http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html

[17] 서준호 ,“IoT환경에서 MQTT와 WebSocket을 활용한 실시간 사물제어 시스템 설계 및 구현”, 호남대학교 대학원 박사학위논문 2016, pp50-58

[18] 김현근 "MQTT, CoAP 통신 프로토콜 사용 전력량 비교에 관한 연구", 숭실대학교 대학원 석사학위논문, 2016, pp10-19

[19] 허웅, "Broker를 적용한 보안성이 향상된 NoSec mode CoAP 연구", 건국대학교 대학원 석사학위논문, 2017, pp3-7

[20] 김은정, “웹소켓 기반의 CoAP 구현 연구”, 숭실대학교 대학원 석사학위논문, 2016, pp3-5

[21] 고석갑 외 3명, "IETF CoAP 기반 센서 접속 프로토콜 기술 동향", 전자통신동향분석 제 28권 제6호, Dec 2013, pp133-136

[22] 심명선 ,“I/O 다중화 처리 향상을 위한 Node.js 기반 고가용 어플리케이션 서버 설계 방법 연구”, 숭실대학교 대학원, 석사학위논문, 2014, pp3-4

[23] Vertx framework, http://vertx.io

[24] 최재영, "이벤트기반 서버를 이용한 대외계 서버의 효과적인 구축 방안", 숭실대학교 대학원 석사학위논문, 2016 pp8-11

[25] 전영준, 황희정, “TOS와 Mobile device 간의 폅섭 QoS를 지원하는 대량 커넥션 서비스 브로커 설계”, (사)한국인터넷방송통신학회 논문지 제 16권 제5호, 2016, pp1-6

[26] Vert.x Manual, http://vertx-kor.otofu.me/manual.html#what-is-vertx

[27] Hazelcast: In-Memory Data Grid, https://hazelcast.com/

[28] Jung-Uoong Park, Jong-Cheol Park, Hee-Dong Park, "Vert.x-Based Active Node Management System for IoT Devices", KKITS, 2016

[29] Nodejs, https://nodejs.org/

[30] JSConf.eu 2009, http://www.jsconf.eu/2009/speaker/speakers_selected.html

[31] 이형근, “Node.js를 이용한 실시간 모바일 웹 콘솔 설계 및 구현”, 한양대학교 대학원 석사학위논문, 2014, pp8-16

[32] 송성근, “LoRa 기반 통신 모듈을 적용한 보안등 원격 관제에 관한 연구”, 2016, pp10-11

[33] Amazon Web Service, https://aws.amazon.com/ko/

ABSTRACT

Design and Implementation of Asynchronous IoT Broker for Monitoring of Crop Growing Environment

Choi Jung Min

Department of Computer Engineering
Graduate school of Information and Technology,
Incheon National University, Incheon, Korea

The Crop Growing Environment Monitoring System was proposed to collect and analyze the crop growing environment using sensor equipments, and to be able to make a customized prescription on the basis of it. To this end, the crop growing environment datum are to be collected by using the censor equipment operating in a wireless network environment. At this time, when these sensors installed in large quantities over the country collect datum, due to the low-speed and low-quality network environment unique to the sensor and a large number of connections, the loss of data or the delay of processing occur. In order to solve such problems, the collection system is required, which can reliably and stably collect and process the data.

In this paper, the collecting system was established by designing

and realizing the asynchronous loT broker using the MQTT protocol for the reliable transmission of a message in the low-speed and low-quality network environment base on the Vert.x framework that supports asynchronous processing. In addition, the experimental environment was constituted using the Amazon Web Services for the realized broker, and the test was conducted for the performance. As a result of the experiment, when 5 broker instances were clustered and divided into 4 topics on 20,000 connections to transfer the data and processed by each topic, it showed the best performance at 56,000 tps. Through this, it could be known that this broker can be used as a collection system to be able to collect and process the crop growing environment quickly and stably in the Crop Growing Environment Monitoring System.

Key words : MQTT, Mass Connection, Asynchronous