[UCI]I804:24011-200000267070

2019년 2월
석사학위 논문

IoT 환경에서 MQTT통신의 자원소모를 줄이기 위한 Publisher 송신 주기 조절 방법

조선대학교 산업기술융합대학원

소프트웨어융합공학과

이 연 주

2019년 2월

석사학위논문

IoT 환경에서 MQTT통신의 자원소모를 줄이기 위한 Publisher 송신 주기조절 방법

이 연 주

IoT 환경에서 MQTT통신의 자원소모를 줄이기 위한 Publisher 송신 주기조절 방법

Publisher Transmission Cycle Adjustment Method to Reduce Resource Consumption of MQTT Communication in IoT Device Environment

2019년 2월 25일

조선대학교 산업기술융합대학원

소프트웨어융합공학과

이 연 주

IoT 환경에서 MQTT통신의 자원소모를 줄이기 위한 Publisher 송신 주기조절 방법

지도교수 김 판 구

이 논문을 공학석사학위신청 논문으로 제출함

2018년 12월

조선대학교 산업기술융합대학원

소프트웨어융합공학과

이 연 주

이연주의 석사학위논문을 인준함

위원장 조선대학교 교수 정 일 용 (인)

위 원 조선대학교 교수 김 판 구 (인)

위 원 조선대학교 교수 최 준 호 (인)

2018년 12월

조선대학교 산업기술융합대학원

목 차

표 목 차

그 림 목 차

ABSTRACT

Publisher Transmission Cycle Adjustment Method to Reduce Resource Consumption of MQTT Communication in IoT Device Environment

YeonJu Lee
Advisor : Prof. Pankoo Kim, Ph.D.
Department of Software Convergence Engineering
Graduate School of Industry Technology Convergence, Chosun University

IoT (Internet of Things) is an intelligent infrastructure and service technology that connects all things based on information and communication technology and exchanges information between people, things, things and things. At present, IoT is being actively researched and developed in various fields such as big data analysis through data collection, smart health care, and smart factory. IoT devices applied in these fields are being transmitted and received in a very limited environment in order to be applied to various environments. As the hardware technology developed, IoT devices became smaller and smaller, and the limited computing performance was achieved. As the existing communication method, TCP and HTTP over it, were not suitable, a new communication protocol was needed. In this paper, we propose a new protocol for object-oriented Internet applications such as Message Queue Telemetry (MQTT) and Constrained Application Protocol (COAP). MQTT is

a protocol optimized for IoT devices, focusing on low power. Considering that resource utilization is limited for clients' application operation, a broker which is a mediator of communication is installed in order to minimize all resource occupation occupied by the protocol and a network communication method of a publish / subscribe structure suitable for a plurality of client connections is applied.

However, if the publish / subscribe structure does not go through any additional communication, Publisher will not know the broker's subscription status at all. Therefore, even if there is no subscriber to receive data by requesting from the Broker, continuous transmission continues and unnecessary power consumption occurs. Therefore, when the Publisher transmits, it is necessary to grasp the status of the broker communication in real time and to control the sending and receiving period.

In this paper, to solve this problem, we use QoS, which is basically provided by MQTT, to construct a service node in QoS 0 stage, allowing the Publisher to receive the communication status of the broker, and only in the QoS 1 stage, The PUBACK packet is partially modified to allow the Publisher to know the status of the Broker through the PUBACK packet returned when it is transmitted. The purpose of this method is to reduce unnecessary consumption of power by enabling Publisher to adjust Publisher 's transmission period according to broker' s Subscribe status.

Since the ACK packet does not exist in QoS 0, the service node is implemented using Nodejs because the new communication means was needed, so that the Publisher can receive the data from the Web and know the communication status of the Broker .

In the first stage of QoS, the structure of the PUBACK packet returned when transmitting and receiving is changed, and the information is stored in the new byte space added, allowing the Publisher to recognize the communication status of the broker. The PUBACK packet having the existing 2-byte fixed header is added with a flag indicating the presence or absence of the Topic to be sent and received in a total of 3 bytes.

Direct testing was performed by applying the proposed method according to QoS.

According to the QoS, the transmission cycle of the Publisher is changed from 4 sec to 1.0 sec, 2.0 sec, 3.0 sec and 4.0 sec depending on the presence or absence of the recipient, and the CPU occupancy rate and the consumed current of the corresponding IoT equipment are measured and reduced I could confirm. Also, considering that the proposed method may be less effective than the conventional method, we measured the data transmission delay time, the current consumption and the data loss and compared them, and it was confirmed that there is no big difference to affect the data transmission / reception.

Reducing the unnecessary power consumption of these publishers is expected to help improve the performance of IoT devices with limited computing environments. Future research will focus on communication types that are better suited for limited computing environments by testing communication reliability in extreme hardware environments.

Ⅰ. 서론

A. 연구 배경 및 목적

IoT(Internet of Things)는 정보통신기술을 기반으로 모든 사물을 연결해 사람과 사물, 사물과 사물간의 정보를 교류하고 상호 소통하는 지능형 인프라 및 서비스 기술이다. 현재 IoT는 데이터 수집을 통한 빅데이터 분석, 스마트 헬스케어, 스마트 팩토리 등과 같은 다양한 분야에서 활발히 연구 개발이 진행되고 있다[1]. 이러한 분야에서 적용되는 IoT 디바이스들은 다양한 환경에 적용되기 위해 매우 제한된 환경에서 송수신이 이루어지고 있다. 하드웨어 기술이 점차 발달함에 따라 IoT 디바이스들이 점차 소형화 되면서 제한된 컴퓨팅 성능을 갖게 되었고 기존 통신방식인 TCP와 그 상위의 HTTP는 이에 적합하지 못하였기 때문에 새로운 통신 프로토콜이 필요하게 되었다. 이에 등장한 프로토콜이 MQTT(Message Queue Telemetry)와 COAP(Constrained Application Protocol)같은 저 전력과 송수신 경량화을 목적으로 한 사물인터넷 응용 프로토콜이다[2].

MQTT는 저전력을 중점으로 하여 IoT 디바이스에 최적화된 프로토콜이다. 클라이언트들의 어플리케이션 동작에 자원 활용이 제한적임을 고려하여 프로토콜이 차지하는 모든 리소스 점유를 최소화하기 위해 통신의 중재자인 Broker를 설치하고 다수의 클라이언트 연결에 적합한 Publish/Subscribe구조의 네트워크 통신 방식을 적용하였다.

하지만 Publish/Subscribe구조는 다른 추가적인 통신수단을 통하지 않을 경우 Publisher는 Broker의 Subscribe 상황을 전혀 알지 못한다. 때문에 만약 Broker에 요청하여 데이터를 수신하는 Subscriber가 존재하지 않을 경우에도 계속 전송이 지속되어 불필요한 전력의 소모가 발생한다. 그래서 Publisher가 전송을 할 때 실시간으로 Broker의 통신현황을 파악하고 송수신 주기의 조절이 필요하다.

본 논문에서는 이를 해결하기 위하여 MQTT가 기본적으로 제공하고 있는 기능인 QoS를 이용해 QoS 0단계에서는 서비스 노드를 구축하여 Publisher가 Broker의 통신 상황을 수신 할 수 있도록 하고 QoS 1단계에서는 QoS 1단계에서만 설정되어있는 PUBACK 패킷을 일부 수정하여 Publisher가 전송할 때 반환되는 PUBACK패킷을 통해 Broker의 상황을 알 수 있도록 한다. 이러한 방식을 적용하여 Publisher가 Broker

의 Subscribe 상황에 따라 Publisher의 송신 주기를 조절 할 수 있도록 하면서 불필요한 전력의 소모를 줄이는 것을 목적으로 한다.

B. 연구 내용 및 구성

본 연구는 MQTT의 통신방식인 Publish/Subscribe구조에서 Publisher와 Broker사이에 발생하는 불필요한 전력소모의 방지를 위해 데이터 송신 주기를 조절 할 수 있는 방법을 QoS 단계에 따라 제안한다. 또한 테스트를 통해 기존 MQTT 통신방식과 본 논문에서 제안하는 통신방식을 데이터 송수신 지연과 손실 측면에서 비교분석하고 Publisher 송신 주기에 따른 IoT디바이스의 CPU사용률과 전력소모량의 변화량을 측정하여 성능 평가를 기술한다.

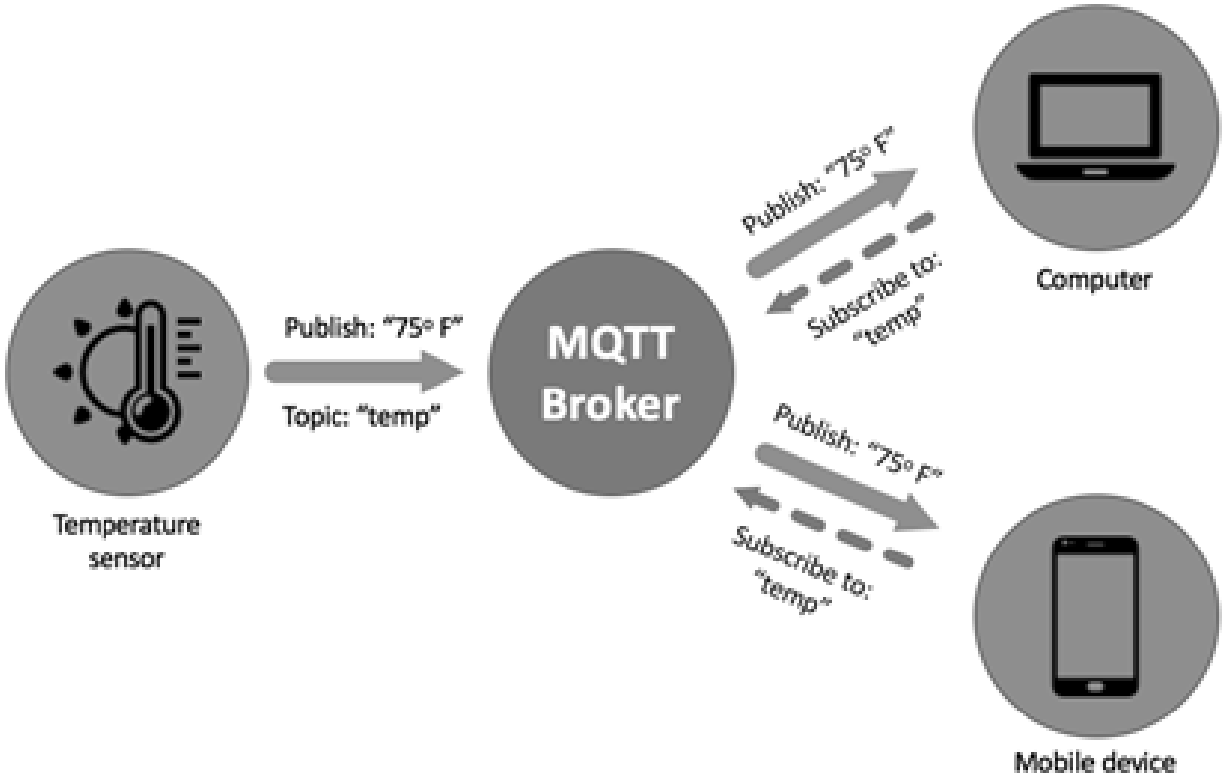
본 논문의 구성은 다음과 같다. 2장에서는 MQTT의 기본원리와 Publish/Subscribe 구조 및 MQTT 패킷 분석을 설명하고 3장에서는 기존 통신방식에 대한 테스트를 통한 문제인식과 QoS 단계에 따른 Publisher의 주기조절 방법에 대해 설명한다. 4장에서는 실험 및 평가를 통해 결과를 도출하고 마지막으로 5장에서는 결론과 향후 연구에 관해 서술하며, 마무리 한다.

Ⅱ. 관련연구

A. MQTT

1. MQTT 개요

MQTT란 Message Queuing Telemetry Transport의 약자로서, IoT분야에서 응용에 용이한 초경량 통신 프로토콜이다. 초기 버전이 1999년에 등장하였고 IoT 시장의 성장에 따라 새로운 통신프로토콜의 필요성이 대두되었고 MQTT v3.1이 2013년 IBM에 의해 OASIS에 상정되었다.



[그림 2-1] MQTT의 Publish/Subscribe 구조

MQTT는 저전력을 중점으로 하여 IoT 디바이스에 최적화된 프로토콜이다. 클라이언트들의 어플리케이션 동작에 자원 활용이 제한적임을 고려하여 프로토콜이 차지하는

모든 리소스 점유를 최소화하기 위해 [그림 2-1]과 같이 통신의 중재자인 Broker를 설치하고 다수의 클라이언트 연결에 적합한 Publish/Subscribe구조의 네트워크 통신 방식을 적용하였다. 이러한 방식은 IoT디바이스의 클라이언트 어플리케이션 동작에 자원이 제한적임을 고려할 때 아주 효율적인 방법이다. 그리고 송수신시 주고받는 데이터 패킷의 고정헤더를 2byte로 설정하여 오버헤드를 줄이면서 프로토콜이 차지하는 리소스 점유를 최소화 할 수 있게 되었다.

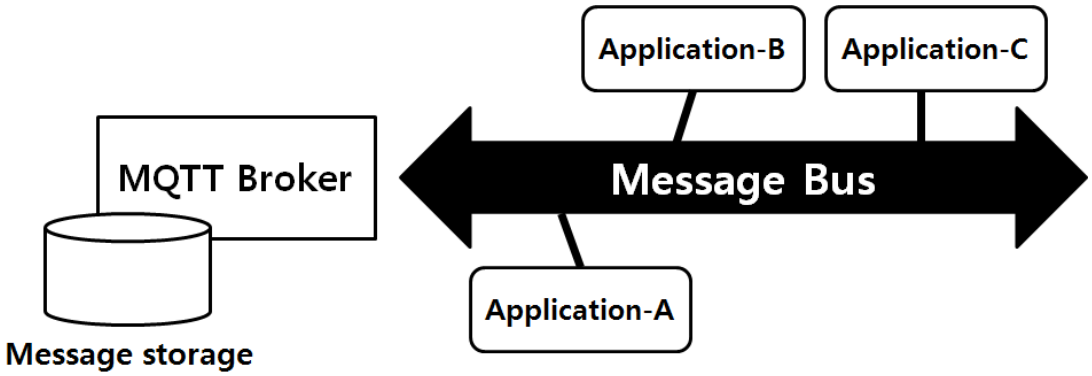
MQTT는 특히 저전력, 신뢰할 수 없는 네트워크, No TCP/IP 기반에서 운용할 수 있다는 장점이 있다. 가전기기, 빌딩, 도시, 산업, 개인 등 다양한 영역에서의 센서 정보를 수집할 수 있고 네트워크 영역으로 보자면 LAN(가정/소규모 오피스), PAN(개인 네트워크), BAN(빌딩 네트워크), MAN(도시영역 네트워크)등에서 사용할 수 있다. 개방형 표준 메시징 프로토콜을 지향하기 때문에 제 3자 기기 제조업체와 소프트웨어 개발업체의 용이한 도입과 적용을 유도 할 수 있다.

하지만 제한된 컴퓨팅 환경에서 사용할 목적으로 나왔기 때문에 데이터양이 대폭 증가 할 경우 데이터 손실이나 전송지연이 발생할 수 있는 단점은 존재한다[3][4]. 현재 MQTT의 점유율이 점차 증가 하면서 사용자들의 요구사항이 많아짐에 따라 버전의 업데이트와 함께 단순 송수신기능이 뿐만 아니라 WebSocket 지원과 SSL연동을 통한 보안기능 등 다양한 형태로 발전해 나가고 있다.

현재에는 Facebook에서 외국 그룹메시징 서비스의 하나인 Beluga에 도입된 MQTT 프로토콜 방식을 가져와 Facebook Messenger에 도입하여 실제 사용되고 있으며 IBM의 Blue projects에서 아바타를 통해 실시간으로 음성을 수화로 번역하는 프로그램이나 장애인을 위한 실시간 위치 인식 메시징 어플리케이션에 쓰이는 등 다양한 분야에 적용되고 있다.

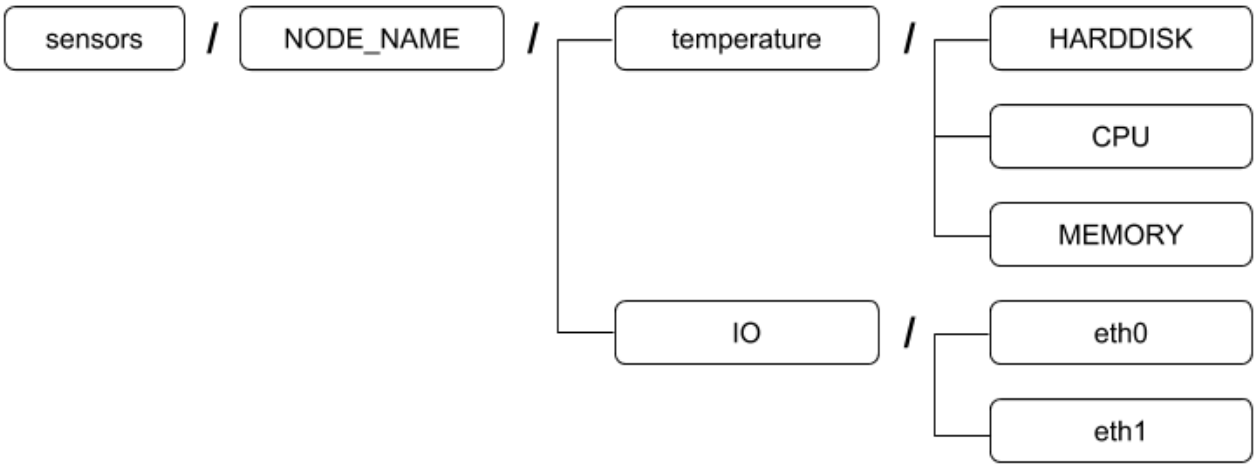
2. Topic

Publisher와 Publisher는 통신을 중재하는 역할을 담당하는 Broker에 대한 클라이언트로 작동하며, Publisher는 데이터를 전송하기 위한 목적으로, Subscriber는 데이터를 수신하기 위한 목적으로 Broker 서버에 연결한다. [그림 2-2]과 같이 메시지 버스 시스템으로 작동한다고 볼 수 있다. Publisher로부터 데이터를 수신 받은 Broker는 메시지버스를 만들고 메시지를 흘려보내면 버스에 연결된 Subscriber의 어플리케이션들이 메시지를 받아가는 방식이다[5].



[그림 2-2] MQTT의 메시지 버스 방식

이러한 버스에는 다양한 메시지들이 전송되는데 이를 구분하기 위해 topic을 사용하여 메시지 채널을 만든다. Publisher가 topic을 발행하게 되면 Subscriber가 받고자 하는 데이터의 topic을 등록하여 구독하는 구조이다. 하나 이상의 Publisher과 Subscriber가 Broker에 연결하여 topic을 발행하거나 구독할 수 있다. 또한 다수의 클라이언트가 하나의 topic를 구독할 수도 있다. Topic은 슬래시(/)를 이용해서 계층적으로 구성할 수 있어서 대량의 센서 기기들을 효율적으로 관리 할 수 있다. 예를 들어 컴퓨터의 다양한 상태를 측정하는 센서가 있다고 가정한다면 [그림 2-3]과 같이 구성 할 수 있다. 문자열을 비교하고, 문자열을 구로 나누어 Topic의 검색의 정확도와 속도를 향상 시킬 수 있다.



[그림 2-3] Topic의 계층적 구조 예시

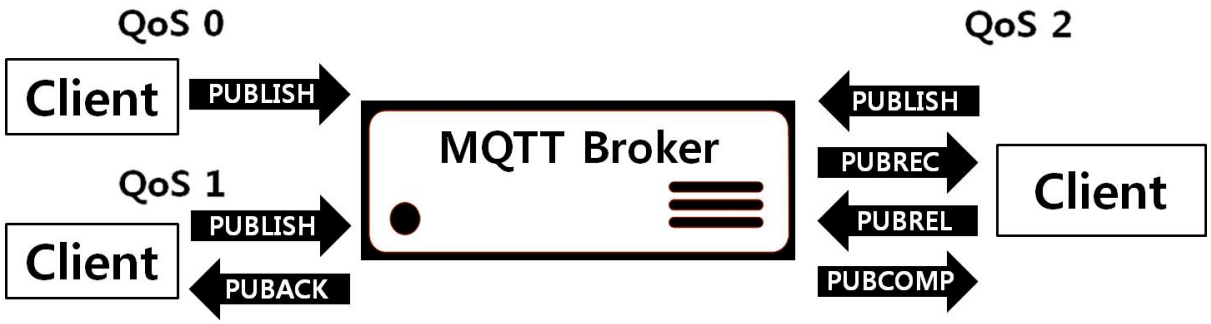
3. QoS(Quality of service)

MQTT는 [그림 2-4]와 같이 메시지 송수신의 신뢰성을 위해 3단계의 QoS 레벨을 제공한다. 이는 반드시 전달되어야 하는 중요메시지에 대한 전달을 보장한다[6].

QoS 0단계는 기존 소켓통신의 UDP방식처럼 신뢰성을 보장하지 않고 데이터를 전송하는 방법과 유사하다. 최대 한번 만 메시지를 전송하고 전달 여부는 확인하지 않는다. 이는 신뢰성을 보장 하지 않지만 데이터 전송속도와 리소스 점유율 측면에서 유리한 방법이기 때문에 가장 많이 사용되고 있다.

QoS 1단계는 적어도 한번 이상을 전달하고 전달여부를 확인하는 방법이다. 수신자가 성공적으로 데이터를 받았을 경우 그림 3의 PUBACK 패킷을 송신자에게 전송하여 데이터의 전달여부를 알린다. 송신자가 수신확인을 수신하지 않은 경우에는 수신 될 때까지 DUP플래그가 설정되어 다시 송신된다.

QoS 2단계는 1단계의 과정을 TCP/IP의 4-Way Handshaking 방식으로 발전시켜 데이터를 정확하게 1번만 전송하도록 하여 신뢰성을 확보하였다. 하지만 Handshaking과정 중에 데이터 전송속도가 느려지고 리소스 점유율이 높아진다.



[그림 2-4] MQTT Quality of service

4. MQTT 데이터 패킷 분석

Bit	7	6	5	4	3	2	1	0
Byte1	MQTT Control Packet type (4)				Reserved			
Byte2	Remaining Length							

[그림 2-5] MQTT의 고정헤더

	7	6	5	4	3	2	1	0
Byte 1	Message Type				DUP	QoS Level		RETAIN
Byte 2	Remaining Length (1 - 4 bytes)							
Byte 3 ... Byte n	Optional: Variable Length Header							
Byte n+1 ... Byte m	Olptional: Variable Length Message Payload							

[그림 2-6] MQTT Control Packet 전체 구조

[그림 2-5]는 MQTT 통신패킷의 고정헤더를 나타내고 있다. 사물인터넷 환경의 특성 때문에 리소스를 줄이기 위하여 용량이 작은 매우 간단한 헤더를 가지고 있다. 패킷의 종류는 1바이트의 0비트~3비트에서 MQTT Control Packet type를 통해 구별하도록 설계되어있다.

[그림 2-6]은 MQTT Control Packet 전체 구조이다. 2Bytes의 Fixed header, 옵션에 따른 Variable Header, Packet의 마지막 부분인 Payload로 이루어져 있다. Fixed header에서 Message Type 부분은0 ~ 3 bit에 위치한 부분으로 메시지 제어 타입을 정

의하며 4 비트의 공간인 2^4 개의 데이터 타입을 정의할 수 있다. 비트 DUP는 PUBLISH 제어 패킷의 중복 전달을 나타내고 비트 QoS는 PUBLISH 서비스 품질을 나타내며 비트 RETAIN PUBLISH 플래그 유지를 나타낸다. Remaining Length 부분은 2 바이트에서 시작하며 variable header 및 payload의 데이터를 포함하여 현재 패킷 내에 남아있는 바이트 수 즉 전체 메시지의 크기를 계산하기 위해서 사용한다. Variable Length header는 CONNECT Packet의 변수 헤더로 프로토콜 이름, 프로토콜 레벨, 연결 플래그 및 연결 유지 순으로 네 개의 필드로 구성되어있다[7].

[그림 2-7]는 MQTT Control Packet type의 값에 따른 메시지 제어타입을 보여준다. 기본적인 연결시도 및 QoS에 따라 전송하는 패킷의 유형 등 다양한 제어타입을 확인 할 수 있다.

Name	Value	Direction of flow	Description
Reserved	0	Forbidden	Reserved
CONNECT	1	Client to Server	Client request to connect to Server
CONNACK	2	Server to Client	Connect acknowledgment
PUBLISH	3	Client to Server or Server to Client	Publish message
PUBACK	4	Client to Server or Server to Client	Publish acknowledgment
PUBREC	5	Client to Server or Server to Client	Publish received (assured delivery part 1)
PUBREL	6	Client to Server or Server to Client	Publish release (assured delivery part 2)
PUBCOMP	7	Client to Server or Server to Client	Publish complete (assured delivery part 3)
SUBSCRIBE	8	Client to Server	Client subscribe request
SUBACK	9	Server to Client	Subscribe acknowledgment
UNSUBSCRIBE	10	Client to Server	Unsubscribe request
UNSUBACK	11	Server to Client	Unsubscribe acknowledgment
PINGREQ	12	Client to Server	PING request
PINGRESP	13	Server to Client	PING response
DISCONNECT	14	Client to Server	Client is disconnecting
Reserved	15	Forbidden	Reserved

[그림 2-7] MQTT Control type 리스트

B. WebSocket

1. WebSocket 기본 원리

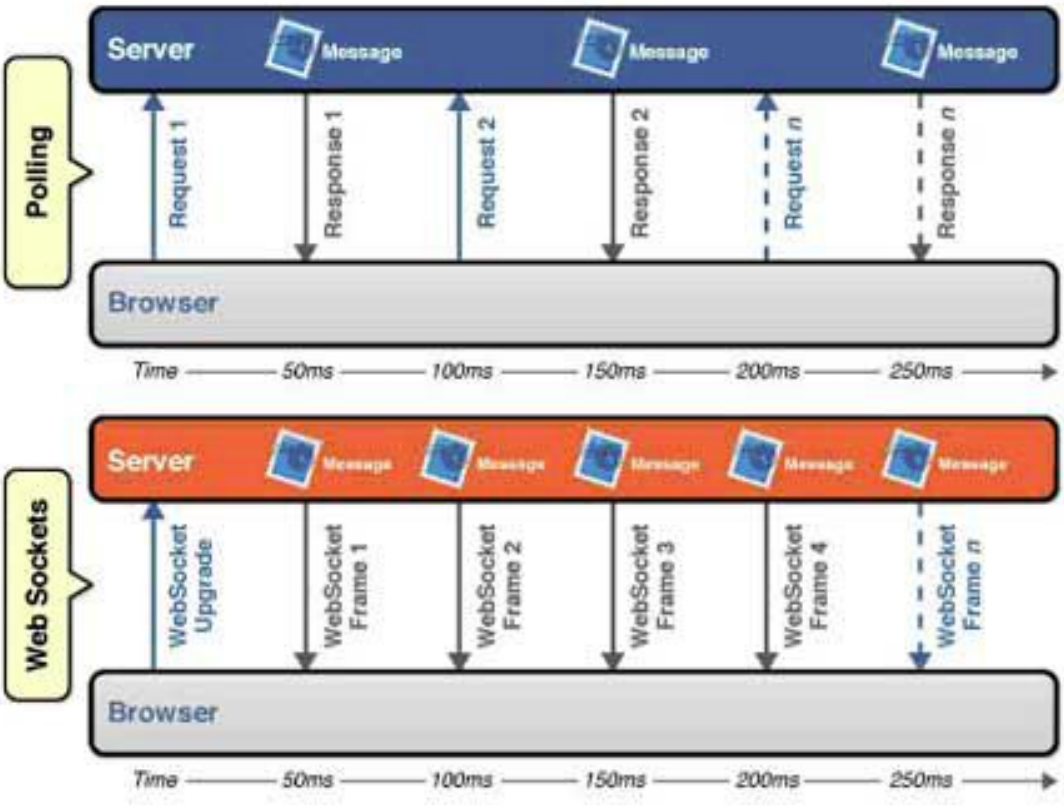
전형적인 브라우저 렌더링 방식은 HTTP Request에 대한 HTTP Response을 받아서 브라우저의 화면을 깨끗하게 지우고 받은 내용을 새로 표시하는 방식이다. 내용을 지우고 다시 그리면 브라우저의 깜빡임이 생기게 된다. 이러한 깜빡임 없이 원하는 부분만 다시 그리며 실시간으로 사용자와 상호작용하는 방식이 나타나고 사용자와 상호작용하는 웹 서비스를 선호하는 사용자가 증가하면서 RIA(Rich Internet Application) 기술의 발달이 촉진되었다.

상호작용하는 웹 서비스를 위해 기존에는 숨겨진 프레임(Hidden Frame)을 이용한 방법이나 Long Polling, Stream 등 다양한 방법을 사용했다. 브라우저가 HTTP 요청을 보내고 웹 서버가 이 요청에 대한 HTTP 응답을 보내는 단방향 메시지 교환규칙을 변경하지 않고 구현한 방식이다. 그렇기 때문에 상호작용하는 웹 페이지를 복잡하고 어려운 코드로 구현해야 했다. 보다 쉽게 상호작용하는 웹 페이지를 만들려면 브라우저와 웹 서버 사이에 더 자유로운 양방향 메시지 송수신(bi-directional full-duplex communication)이 필요하다. 그래서 HTML5 표준안의 일부로 WebSocket API(이후 WebSocket)가 등장했다[8].

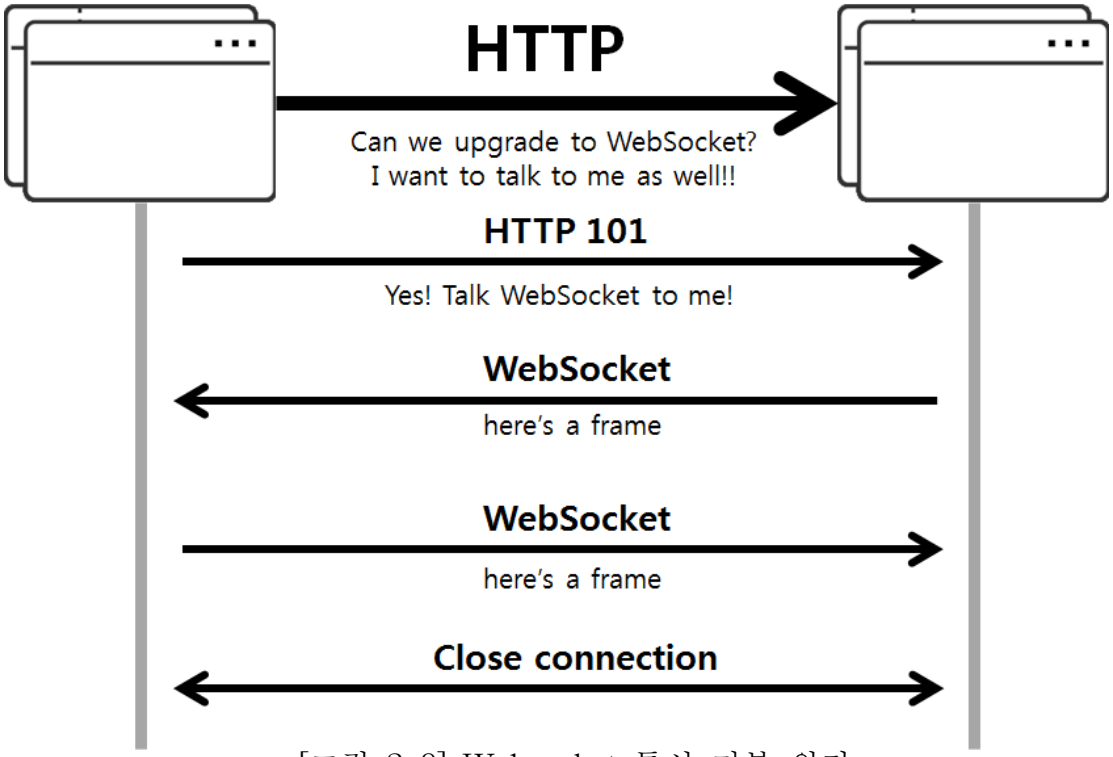
WebSocket은 서버와 클라이언트 간의 효율적인 양방향 통신을 실현하기 위한 구조이다. WebSocket은 그 이름에서 알 수 있듯이 소켓을 이용하여 자유롭게 데이터를 주고받을 수 있다. 즉 기존의 요청-응답 관계 방식보다 더 쉽게 데이터를 교환할 수 있다. 데이터의 실시간 특성을 중시하는 웹 애플리케이션들이 많이 등장하게 되면서 많은 주목을 받고 있다[9].

자바 스크립트의 처리 성능이 크게 향상 되었지만 웹 애플리케이션의 네트워크 통신 부분에서 병목현상이 나타나기 때문에 웹 소켓은 실시간 웹을 구현하기 위한 핵심기술로 기대 받고 있다. 일반적인 TCP소켓과 같이 연결지향 양방향 전이중 통신이 가능하다. 이와 같은 특징으로 웹에서도 채팅이나 게임, 실시간 주식 차트와 같은 실시간이 요구되는 응용프로그램의 개발을 한층 효과적으로 구현할 수 있게 되었다. 결정적인 차이는 프로토콜에 있다. WebSocket 프로토콜은 [그림 2-8]과 같이 접속 확립에

HTTP를 사용하지만, 그 후의 통신은 WebSocket 독자의 프로토콜로 이루어진다. 또한, header가 상당히 작아 overhead가 적은 특징이 있다. 장시간 접속을 전제로 하기 때문에, 접속한 상태라면 클라이언트나 서버로부터 데이터 송신이 가능하다. 더불어 데이터의 송신과 수신에 각각 커넥션을 맺을 필요가 없어, 하나의 커넥션으로 데이터를 송수신 할 수 있다. [그림 2-9]은 WebSocket과 일반적인으로 널리 사용하고 있는 Ajax Long Polling 방식을 비교한 것이다. 속도와 데이터 전송량 면에서 우위에 있다는 것을 볼 수 있다[10].



[그림 2-8] 기존 Ajax Polling과 Websocket 비교



[그림 2-9] Websocket 통신 기본 원리

C. MQTT 전송에 관한 기존 연구

J. H. Kim의 연구에서는 MQTT의 Read/Write 과정에 활용되는 메시지가 없다면 불필요한 송수신 루프가 진행되는 것을 해결하기 위하여 강화학습 도구인 MARL (Multi-Agent-Reinforcement Learning)를 활용하여 디바이스들의 사용 패턴을 기계학습으로 예측하여 메시지의 양을 조절하는 방안을 제안하였다[11]. 또한 H. C. Jo의 연구에서는 MQTT에서 종단 단말이 갖는 주기적 메시지 전송특성을 고려하기 위해 QoS레벨, 메시지 사이즈, 노드의 개수, 통신 매체 등과 함께 메시지 전송주기를 변화시켜 수신율을 측정하였다. 이를 토대로 데이터 전송주기를 결정하기 위해 고려해야할 사항들을 제시하였다[12].

S. H. Kim의 연구에서는 안정적인 사물인터넷의 플랫폼을 위해 MQTT기반으로 데이터 수집 솔루션을 제안하였다. 기존 HTTP 프로토콜을 지원하는 시스템에서의 데이터 헤더의 크기 때문에 전송 효율을 문제가 보였고 이는 시스템 과부하의 문제점을 발생시켰다. 이에 경량의 표준 사물인터넷 프로토콜인 MQTT와 Web Socket을 활용하여 전송효율을 개선하였고 안정성 있는 사물인터넷 플랫폼 수집 솔루션을 제안하였다[13].

S. J. Kim의 연구에서는 제한된 네트워크 환경에서의 디바이스 통신을 충족시키기 위하여 MQTT를 사용하는 추세이지만 데이터 전송에 대한 우선순위 처리방식을 지원하지 않아 특정한 IoT서비스에서의 Quality of Service를 제공 할 수 없다는 단점을 제시하였다. MQTT에서 다양한 IoT서비스의 우선순위를 고려하기 위하여 노드에서 MQTT의 고정헤더에 우선순위 플래그를 추가하여 메시지를 전송하며, Broker에서는 해당 메시지에 대한 우선순위를 확인하여 우선적으로 처리하는 방식으로 노드 간 종단 간 지연의 감소에 대한 해결방안을 제시하였다[14].

S. W. Kang의 연구에서는 라즈베리파이 환경에서 자주 사용되는 MQTT 프로토콜에 대해서 IoT의 특성상 전력소모에 대해 민감하기 때문에 이에 대한 에너지 효율성운 매우 중요한 이슈라고 언급하였다. 이에 사물인터넷 하드웨어 플랫폼으로 널리 사용되는 라즈베리파이에서 MQTT의 데이터 송수신의 다양한 송수신 상황을 제시하고 이에 따른 전력소모를 측정하여 분석하였다[15].

IoT 센서의 데이터 처리 부담을 줄이기 위한 방안으로 S. J. Lee의 연구에서는 경량 프로토콜인 MQTT를 이용하여 센서 노드들이 보내는 데이터를 최소화하여 트래픽 효

율성을 높이는 Broker캐싱을 제안하였다. MQTT에서 센서 노드가 보낸 데이터를 Broker가 캐싱해두고, 센서 노드로부터 중복 처리형 패킷을 받았을 때 캐싱한 데이터를 Subscriber에게 전송하도록 한 방식이다[16].

MQTT가 단일 어플리케이션으로 전송될경우에 발생하는 데이터 부하로 인한 손실이나 지연 등을 해결하기 위하여 Yue. Ma의 연구에서는 인터넷 트래픽을 분산시키기 위하여 RocketMQ에 MQTT를 접목하여 높은 신뢰성과 실시간 능력의 특성을 가진 MQTT 프로토콜 메시지 푸시 서버를 구현하였다[17].

사물인터넷 응용 프로토콜 시장은 MQTT와 CoaP가 양분하고 있다. 통신의 구조도 매우 유사하기 때문에 Thangavel, Dinesh의 연구에서는 WSN 환경에서 MQTT와 CoAP를 동시에 지원하도록 공통 미들웨어를 구현하였고 테스트를 통해 두 프로토콜의 송수신 성능을 연구하는 실험을 수행하였다[18]. 테스트 결과 MQTT가 Coap보다 전송 지연이 적으며 데이터 손실률이 높을수록 이러한 경향이 뚜렷하다.

MQTT의 QoS는 매우 중요한 기능을 담당한다. 데이터의 신뢰성을 보장하기 위하여 0~2단계로 나누어 기능을 제공한다. S. H. Lee의 연구에서는 이러한 QoS의 단계에 따른 MQTT 손실 및 지연에 대해 상관분석을 연구하였다. 다양한 페이로드를 사용하여 3가지 수준의 QoS를 통해 메시지를 전송함으로써 엔드 투 엔드 지연 및 메시지 손실을 분석하기 위해 패킷을 캡처하는 방식으로 진행하였다. QoS 0단계는 일방향 통신이기 때문에 메시지 손실은 가장 높지만 전송 지연이 가장 낮았고 QoS 2단계는 핸드셰이킹 과정으로 송수신이 진행되기 때문에 메시지 손실은 가장 낮지만 전송 지연이 가장 높았다[19].

인터넷환경은 HTTP프로토콜이 지배하고 있지만 IoT시장이 점차 커지면서 이에 적합한 새로운 프로토콜에 대한 논의가 이루어지고 있다. 기존 IP형태는 IoT의 적용할 때 문제가 발생한다. 이에 Tetsuya Yokotani의 연구에서는 HTTP가 IoT 환경에 적용될 때 성능저하 우려가 있기 때문에 이를 실제적인 수치상으로 알아내기 위하여 기존의 HTTP프로토콜와 MQTT프로토콜을 네트워크 자원의 측면에서 비교분석을 하였다. 동일한 IoT 디바이스 측정 환경에서 MQTT가 HTTP보다 평균 30%가량 리소스 자원을 덜 소모한다는 것을 분석하였다[20].

Ⅲ. QoS에 따른 Publish 주기 조절 방법

A. Publisher와 Broker의 송수신 상황

MQTT의 송신구조는 Publisher가 Broker에게 지속적으로 데이터를 일 방향으로 송신하기 때문에 수신할 대상인 Subscriber가 존재하지 않을 경우에도 송신이 지속되어 불필요한 전력소모를 발생시킨다는 것을 언급하였다. 본 장에서는 실제 통신테스트를 통해 MQTT의 송수신 방식을 파악하고 또한 어떠한 형태로 문제점이 발생하는지 확인해 보았다.

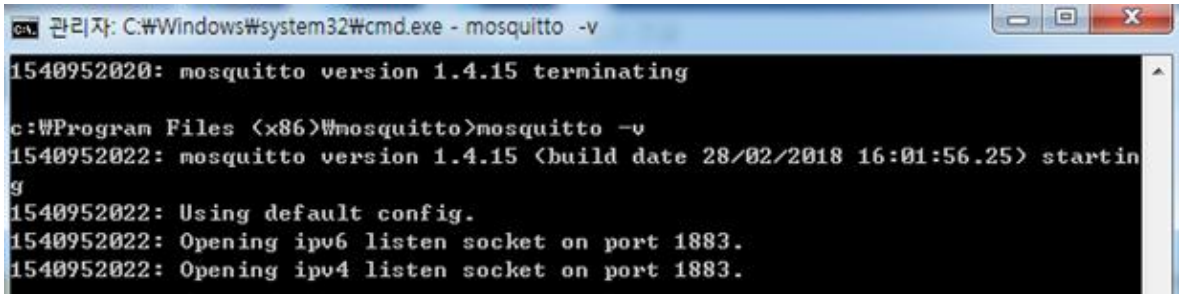
우선 토대가 되는 MQTT Broker가 운영되어야한다. [그림 3-1]은 Broker를 1883번 포트를 사용하여 실행하는 것을 확인 할 수 있다. 다음은 Broker로부터 데이터를 받을 Subscriber을 등록해야한다. [그림 3-2]는 Subscriber가 Broker로 데이터 수신을 요청하는 상황이며 [그림 3-3]을 통해 요청이 수신되어 mosqsub:17856-chosun-PC라는 ID와 hello라는 Topic을 사용한다는 정보의 등록이 이루어 졌음을 알 수 있다.

수신자가 등록이 되었으니 Publisher가 데이터 전송을 해야 하는 차례이다. [그림 3-4]는 Publisher가 Broker로 Hello라는 Topic과 hi라는 메시지를 담아서 전송하는 화면이다. 전송을 받은 Broker는 곧바로 Topic이 일치하는 Subscriber들에게 데이터를 넘겨준다. 이는 [그림 3-5]에서 Received PUBLISH와 Sending PUBLISH가 동시에 발생하는 상황으로 알 수 있다. mosqpub:17720-chosun-PC라는 ID를 가진 Publisher로부터 데이터가 왔으며 전에 등록되었던 Subscriber인 mosqsub:17856-chosun-PC로 송신이 진행되었음을 확인 할 수 있다. Broker로 수신을 요청한 Subscriber는 Topic이 본인과 일치 할 경우 데이터를 수신하게 되고 [그림3-6]처럼 hi라는 메시지를 수신 할 수 있다.

이러한 MQTT의 송신구조는 전력을 적게 소모하면서도 데이터 송수신의 실시간성을 보장하기 때문에 IoT 분야에서 효율적으로 적용된다. 하지만 다음에 제시할 상황을 통해서 불필요한 송신으로 인해 전력의 소모가 발생하는 것을 파악 할 수 있다. [그림 3-7]을 보면 Subscriber가 수신 연결을 중단하여 Broker서버에 'socker error on client mosqsub:17856-chosun-PC, disconnecting.'이 메시지가 출력되었다. 현재 broekr에 수신 받을 대상이 없다는 표현이다. 이 때 [그림 3-8]에서 등록된 Subscriber가 없는

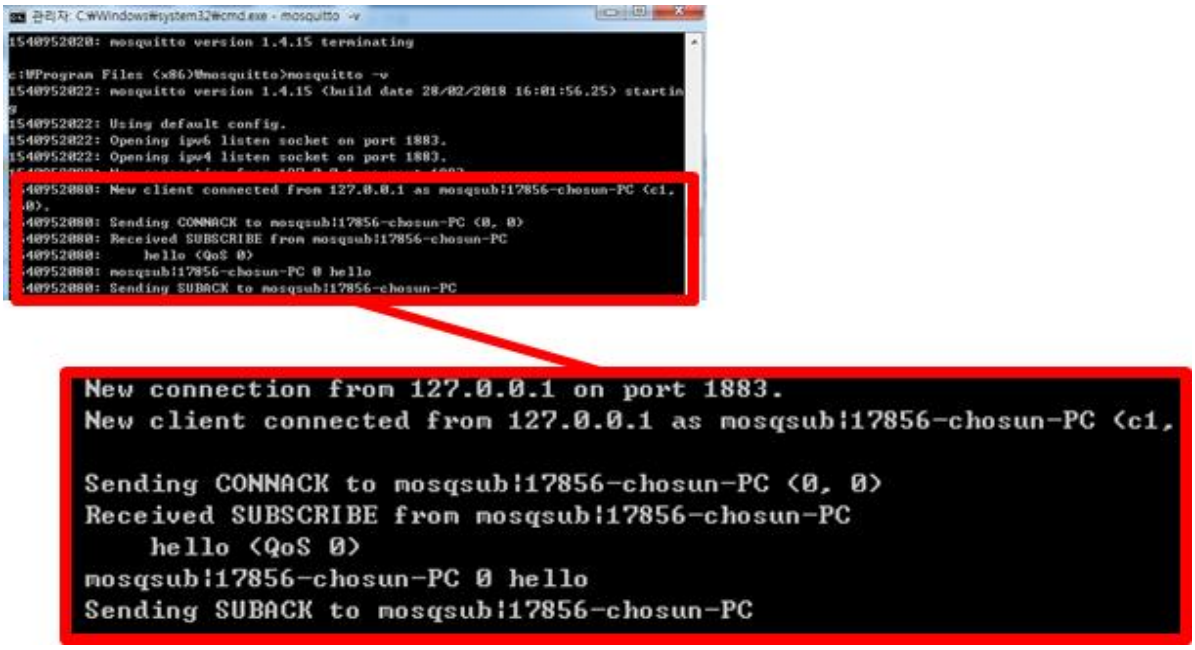
Broker에 Publisher가 데이터를 전송하게 될 경우 아무런 이상 없이 전송이 지속 되는 것을 확인 할 수 있다. 목적지에 대한 정보도 출력되지 않았지만 Broker로 전송이 이루어 졌다. 이는 수신자가 없는 상황에서도 송신자는 Broker의 상황을 인지하지 못하기 때문에 의미 없는 데이터를 계속 보내는 상황을 만든다. 제한된 컴퓨팅에서 주로 사용하는 MQTT의 특성상 불필요한 송수신은 어플리케이션 전체의 성능의 저하와 전력소모를 야기 시킬 수 있다.

본 논문에서는 이러한 MQTT의 불필요한 성능저하와 전력소모를 방지하지 위해 유동적인 주기조절 방법을 제안하고자 한다. Publisher가 Broker에 접속하고 있는 Subscriber에 따라 송신 주기를 조절하기 위해서는 Broker로부터 Subscriber정보를 직접 전송 받거나 데이터를 참조 할 수 있는 공통적인 추가 통신수단이 있어야한다. 그래서 ACK패킷이 따로 존재하지 않는 QoS 0에서는 서비스 노드를 추가하여 통신상황을 수신하는 방식으로 진행하였고 QoS 1에서는 PUBACK 패킷을 수정하는 방법을 사용하였다.



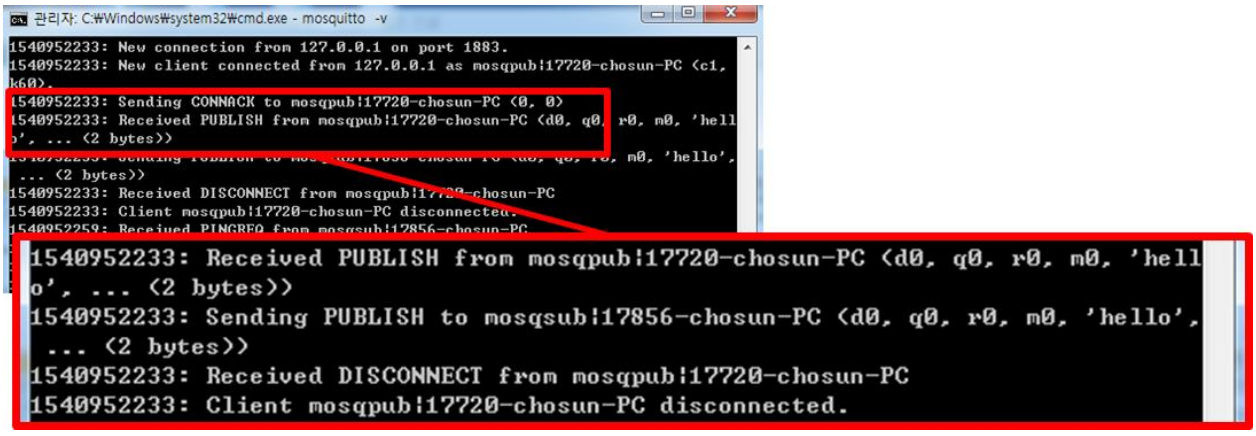
[그림 3-1] MQTT Broker서버 1883번 포트 실행

[그림 3-2] Subscriber가 MQTT Broker에 구독 요청

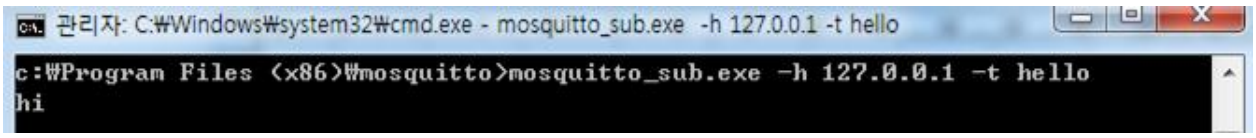


[그림 3-3] MQTT Broker가 Subscriber 요청 확인

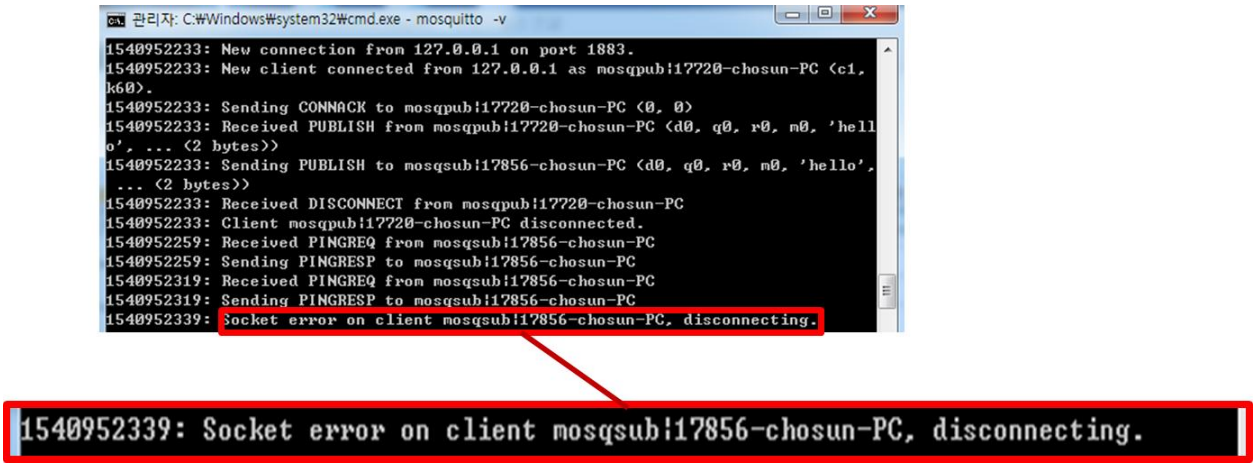
[그림 3-4] Publisher가 MQTT Broker로 데이터 전송



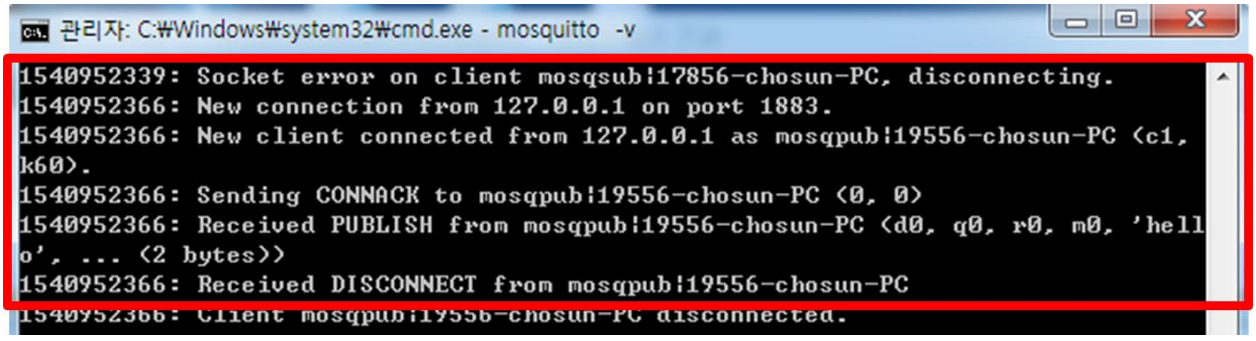
[그림 3-5] MQTT Broker의 데이터 수신 및 전송



[그림 3-6] Subscriber 데이터 수신



[그림 3-7] MQTT Broker에서 Subscriber 데이터 수신 연결 중단 확인

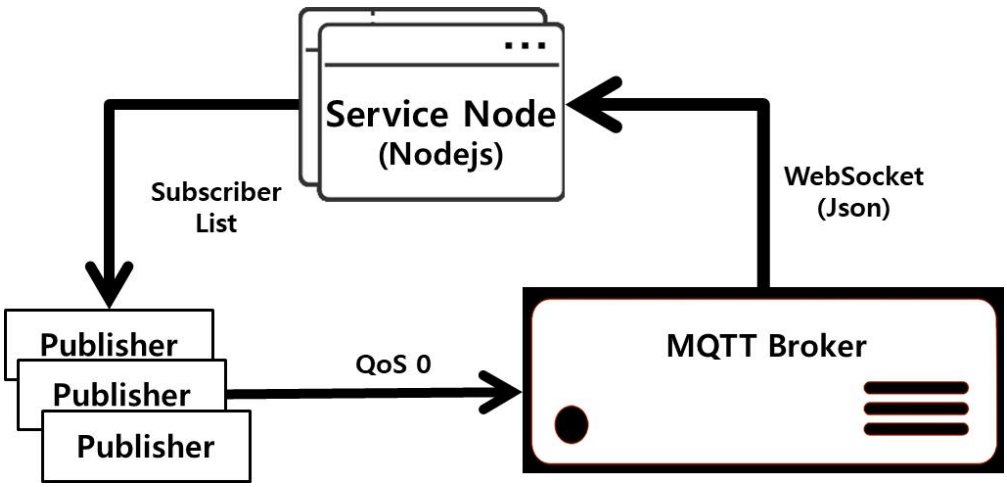


[그림 3-8] Subscriber의 수신 중단 후에도 데이터 수신

B. QoS 0단계에서 서비스 노드를 이용한 Publisher 주기 조절 방법

QoS 0단계 에서는 데이터를 UDP 프로토콜과 유사하게 목적지에 일 방향으로 전송한다. 이러한 방식은 Publisher의 데이터 전송 시 발생하는 부하를 줄일 수 있어서 주로 사용되는 방식이지만 ACK패킷이 존재하지 않아 데이터의 신뢰성은 낮다는 특징이 있다. 이러한 구조에서 Broker와의 정보를 주고받기 위해서는 추가적인 통신수단의 도입이 필수이기 때문에 본 논문에서는 일종의 서비스 노드를 구축하여 Publisher가 Broker로부터 통신 상황을 받을 수 있도록 한다.

[그림 3-9]는 QoS 0단계에서 Publisher가 Broker로부터 정보를 받는 것을 대략적으로 표현한 그림이다. 서비스 노드를 구축하고 Broker는 WebSocket을 이용하여 Json형태로 Subscriber들의 Topic과 아이피주소 목록을 주기적으로 전송하고 서비스 노드는 데이터화 시켜 저장한다. 서비스 노드에 전송되어 저장한 Subscriber 리스트들은 Publisher가 전송할 때마다 HTML 파싱을 통해 정보를 가져와서 수신자 상황에 따라 전송을 중단하거나 주기를 변경한다. [그림 3-10]는 Broker에서 서비스 노드로 Json형태로 전송된 정보를 루프백 아이피주소로 간단하게 출력시킨 웹페이지 예시이다. Json 구조로 topic과 Subscriber의 아이피를 확인 할 수 있다. Publisher는 이러한 정보를 HTML parser를 이용하여 서비스노드에 있는 데이터를 가져 올 수 있도록 한다.



[그림 3-9] QoS 0단계에서의 전송 주기 조절 방법

[그림 3-10] Json 데이터 출력 화면

C. QoS 1단계에서 PUBACK패킷을 이용한 Publisher 주기 조절 방법

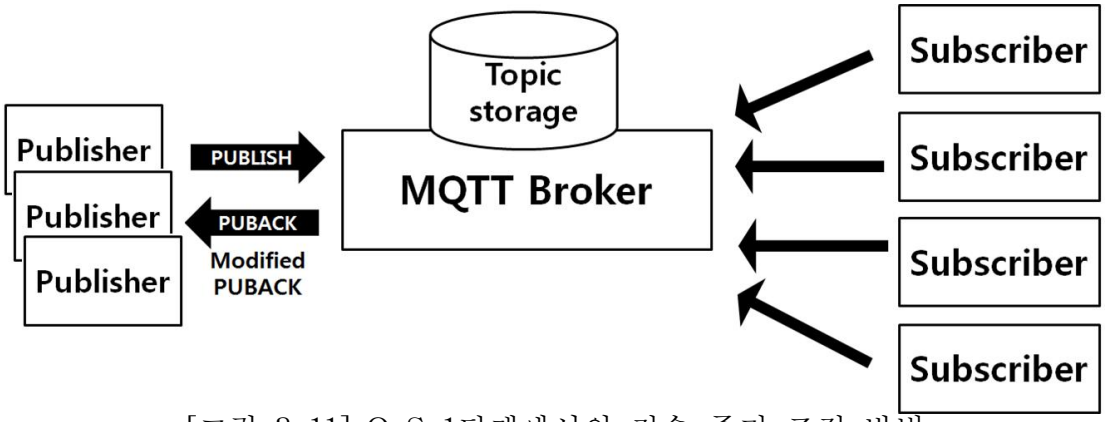
Qos 1단계는 0단계와 달리 데이터 전송 시 PUBACK라는 패킷을 받아 데이터 전송 유무를 확인 할 수 있게 하면서 신뢰성을 높인 방식이다. 이는 Publisher와 Broker가 정보를 교류 할 수 있는 수단으로 사용 될 수 있기 때문에 PUBACK 패킷을 이용하여 Broker로부터 정보를 받을 수 있도록 하였다.

[그림 3-11]는 제안하는 방법의 전체적인 구조도이다. Broker는 Subscriber들이 접속할 때 저장 공간을 만들어 Topic과 Subscriber 리스트를 생성하고 데이터가 수신됐을 때 목록에 대한 결과를 패킷에 추가된 1바이트의 공간에 정보를 담아 반환한다. Publisher는 데이터를 전송하고 반환되는 PUBACK패킷의 3번째 바이트의 정보를 추출하여 현재 리스트 유무를 인지하고 전송 주기를 조절하게 된다. 본 논문에서는 3바이트의 Topic Presence Message 플래그는 Boolean형으로 구성하여 True 혹은 False 형태로 정보를 전송하고 Publisher가 전송하는 Topic을 구독하는 Subscriber들이 있는지 알려주는 역할을 하도록 설정하였다.

[표 3-1]과 [표 3-2]는 기존의 PUBACK패킷에 세 번째 바이트 공간에 플래그를 추가하기 전과 후를 표현한 것이다. Qos 1단계의 PUBACK패킷에 플래그를 추가하여 Publisher가 Broker로부터 Subscriber정보를 받을 수 있도록 하였다.

[그림 3-12]는 수정된 PUBACK패킷의 송수신이 어떤 방식으로 작동하는지 대략적으로 표현한 흐름도이다. Broker는 Publisher가 송신한 데이터에서 Topic을 추출하고 해당 Topic에 대한 메시지를 수신 받을 Subscriber가 접속해있을 경우에 PUBACK패킷에 True값을 Boolean형으로 전송하고 접속이 없을 때에는 false값을 설정하여 패킷을 전송한다. 반환되는 패킷을 받은 Publisher는 사전에 Boolean값에 따라서 변화할

주기를 설정하고 수신된 PUBACK패킷의 Boolean값에 따라 설정한 주기 값으로 송신을 조절한다.



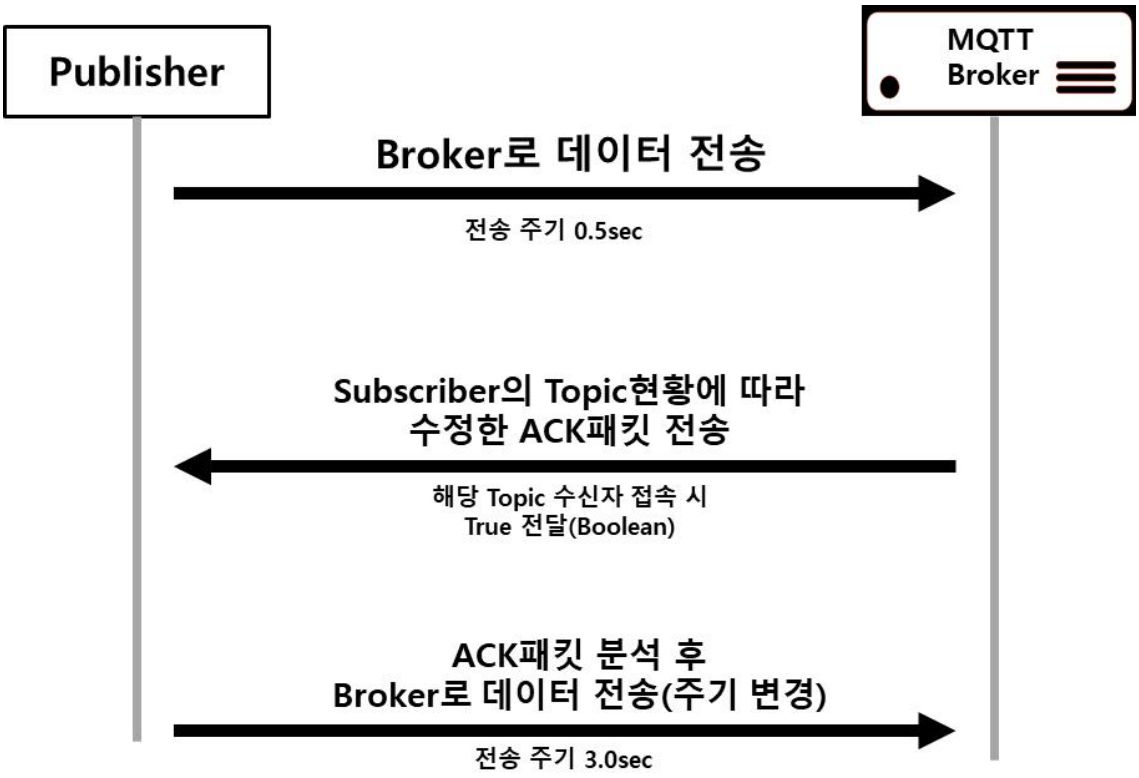
[그림 3-11] QoS 1단계에서의 전송 주기 조절 방법

[표 3-1] 일반적인 PUBACK 패킷

Bit	7	6	5	4	3	2	1	0
Byte1	MQTT Control Packet type (4)				Reserved			
Byte2	Remaining Length							

[표 3-2] 수정한 PUBACK 패킷

Bit	7	6	5	4	3	2	1	0
Byte1	MQTT Control Packet type (4)				Reserved			
Byte2	Remaining Length							
Byte3	Topic Presence Message (boolean)							



[그림 3-12] 수정된 패킷을 통한 Publisher 송수신 원리

Ⅳ. 실험 및 평가

A. 실험 환경

본 장에서는 Publisher의 주기에 따른 CPU점유율과 IoT디바이스의 소모전류량을 측정한다. 실험 도구는 [표 4-1]과 같이 서비스 노드는 Nodejs를 사용하였고 HTML 파싱은 Jsoup 라이브러리를 사용하였다. MQTT Broker는 Eclipse Jee Photon환경에서 JAVA기반 오픈소스 경량화 Broker인 Mosquette를 이용해 내부구조를 수정하였고[21], MQTT Client는 paho MQTT client을 통해 구현하였다[22]. 성능 평가를 위한 IoT디바이스는 RASPBIAN 4.14버젼의 Raspberry Pi 3 Model B를 사용하고 RCPU를 이용해 CPU점유율과 멀티미터기로 소모전류량을 측정하였다.

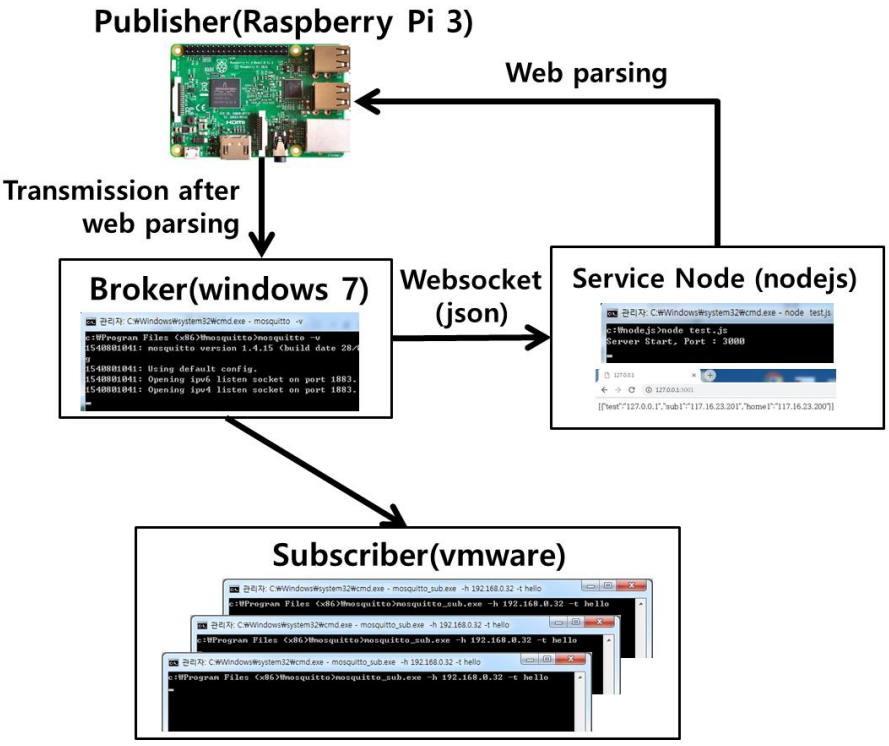
실험에 사용된 모든 PC들은 Windows 7 Enterprise 64비트 운영체제와 Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz 프로세서 환경에서 진행되었다.

[표 4-1] 테스트 도구

	Test tool
MQTT Broker	Mosquette
MQTT Clinet	Paho MQTT Client
IoT Device	Raspberry Pi 3 Model B (RASPBIAN 4.14)
Measurement Tools	RCPU
Development Tools	Eclipse Jee Photon
Service node	Nodejs
Java HTML parser	Jsoup

B. QoS 0단계의 테스트 구조

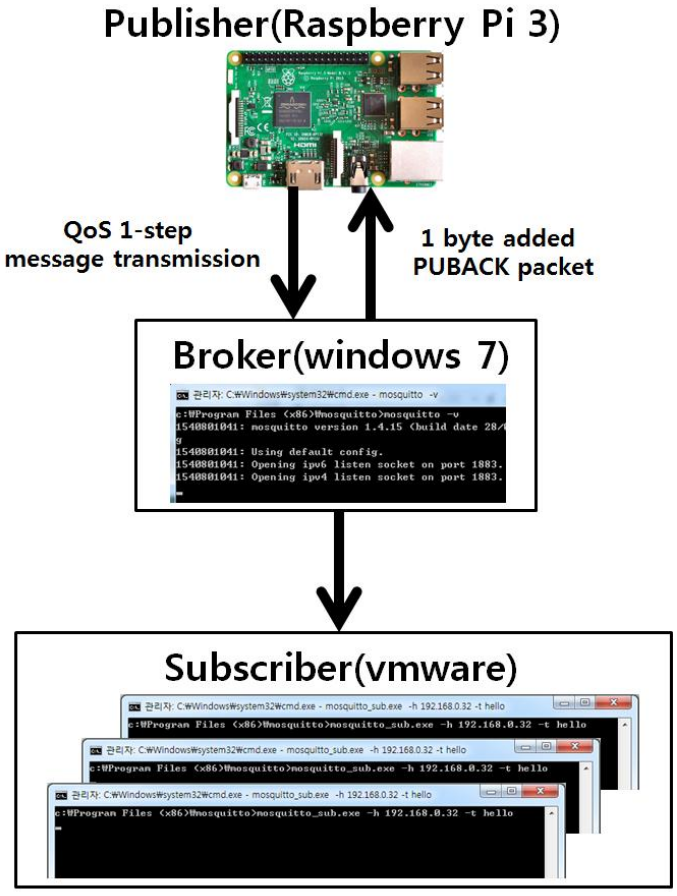
[그림 4-1]과 같이 QoS 0단계에서는 데이터의 전송유무를 확인하는 ACK패킷이 따로 존재하지 않기 때문에 Publisher가 Broker와 연결되어있는 서비스 노드를 통해 통신상황을 수신하는 방식으로 테스트를 진행하였다. Publisher가 송신 시 HTML 파싱을 통해 서비스 노드로부터 데이터를 가져오기 때문에 새로운 통신수단이 추가된 형태이다. 본래 단일 Publisher가 아닌 다수의 Publisher가 Broker에 송신을 진행하는데 성능측정을 위해 단일 Publisher를 통해 테스트를 진행하였다. 서비스 노드는 Nodejs로 구현하였고 WebSocket을 통해 데이터를 수신받기 위한 WebSocket API를 사용하였다. 전송 테스트의 전체적인 구조는 Broker서버가 기본적으로 서비스 노드에 WebSocket을 통해 Subscriber의 아이피와 topic정보를 json형태로 지속적으로 전송하여 업데이트 시킨다. Publisher 기능을 하는 Raspberry Pi가 전송하기 전 Web Page부터 수신자 정보를 HTML parsing을 통해 가져오고 결과 값에 따라 Broker로 전송할 주기를 결정 후 전송한다. 전송된 데이터는 그대로 Subscriber에 이어서 전송된다.



[그림 4-1] QoS 0단계에서의 테스트 구조

C. QoS 1단계의 테스트 구조

QoS 1단계는 기존의 PUBACK 패킷에 1바이트 용량의 정보를 추가하였다. 추가적인 통신 수단이 없기 때문에 구조는 [그림 4-2]와 같이 비교적 간단하다. 전체 테스트 구조는 Publisher 기능을 하는 Raspberry Pi가 Broker서버로 데이터를 전송하고 PUBACK 패킷을 수신 할 준비를 한다. QoS 1단계 패킷을 받은 Broker는 현재 Subscriber의 리스트를 확인하고 결과에 따라 수정한 PUBACK 패킷에 정보를 담아 반환시킨다. 결과를 받은 Publisher는 주기를 알맞게 조정하게 된다. 이 방법 또한 데이터 송수신 비교분석을 위해 단일 Publisher를 통해 송신 테스트를 진행하였다.



[그림 4-2] QoS 1단계에서의 테스트 구조

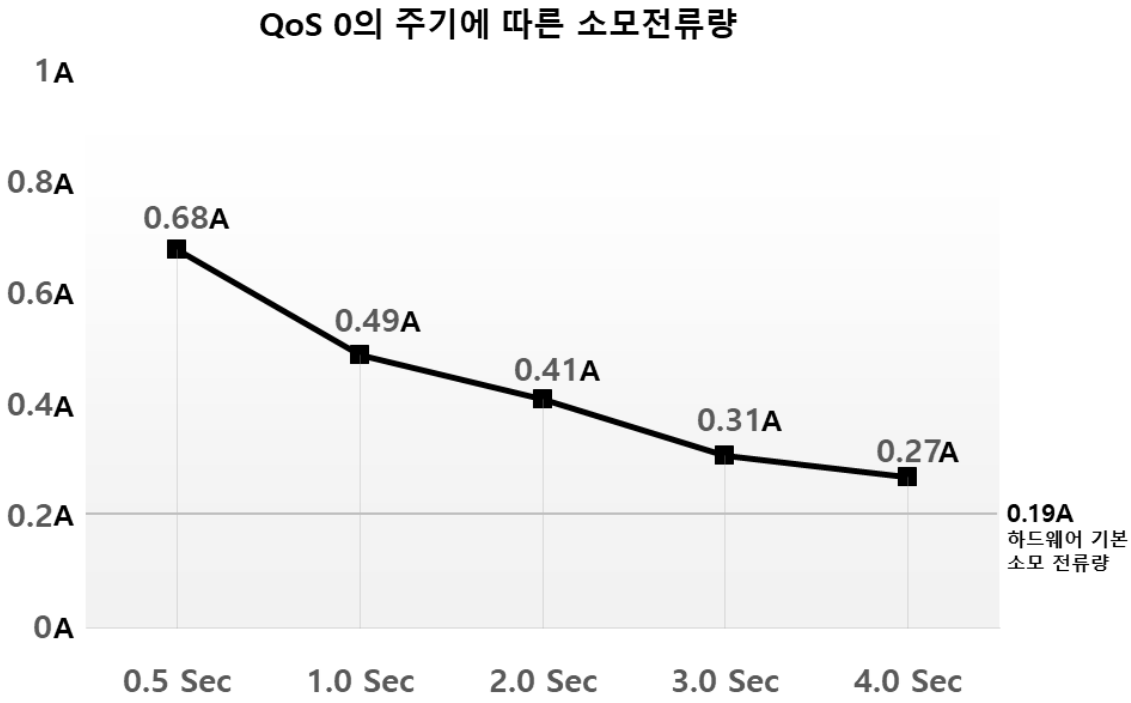
D. 성능 평가

1. QoS 0단계에서 전송 주기에 따른 CPU사용률 및 전류소모량

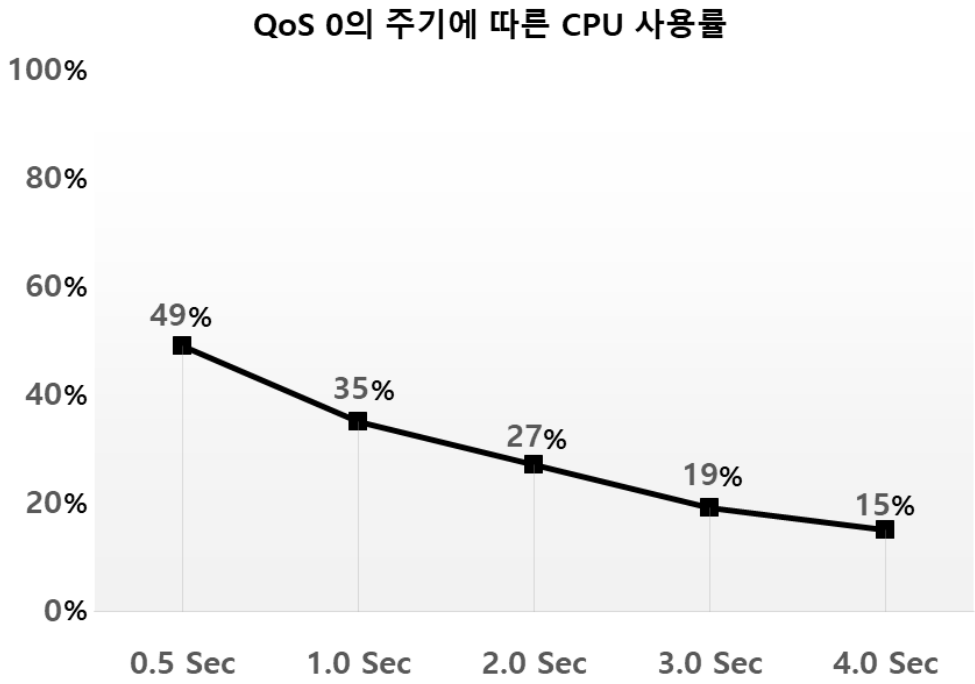
제안한 MQTT 통신방식이 실제로 전력소모량 개선에 효과가 있는지 확인하기 위해 테스트를 하였다. [표 4-2]는 테스트하기 위한 MQTT 통신 설정이다. 전송은 총 1000회 실시하였고 기본 전송 주기는 0.5sec로 설정하여 500회를 전송하고 서비스노드로부터 수신자가 없다는 데이터를 전송받으면 주기를 변경하여 500회를 전송하였다. 변경하는 주기는 1.0sec, 2.0sec, 3.0sec, 4.0sec 총 4가지로 설정하고 테스트를 진행하였다. [그림 4-3]와 [그림 4-4]은 Publisher의 주기에 따른 CPU점유율과 전력소모량을 측정한 그래프이다. [그림 4-3]의 0.19A는 아무런 동작을 하지 않은 상태에서 하드웨어의 구동에 필요한 기본 전류소모량을 표기하였다. 전송 주기가 변경 되면서 CPU점유율과 전력소모량이 줄어드는 것을 확인 할 수 있었다. 이러한 측정치는 제한된 컴퓨팅 자원을 가진 IoT 디바이스에서 중요하게 고려해야 할 사항이다.

[표 4-2] QoS 0단계 CPU사용률 및 전류소모량 측정 설정표

	Value
Topic	“Cycle”
Number of transmissions (Before cycle change)	500 time
Number of transmissions (After cycle change)	500 time
Transmission Message	“hello Sensor” (12byte)
Transmission Frequency (Before / After)	0.5sec / 1.0sec, 2.0sec, 3.0sec, 4.0sec
Parsing Data	{”test : 192.168.0.1“, “test1 : 192.168.0.2”, “test2 : 192.168.0.3”}



[그림 4-3] QoS 0단계 Publisher의 주기에 따른 전력 소모량 변화



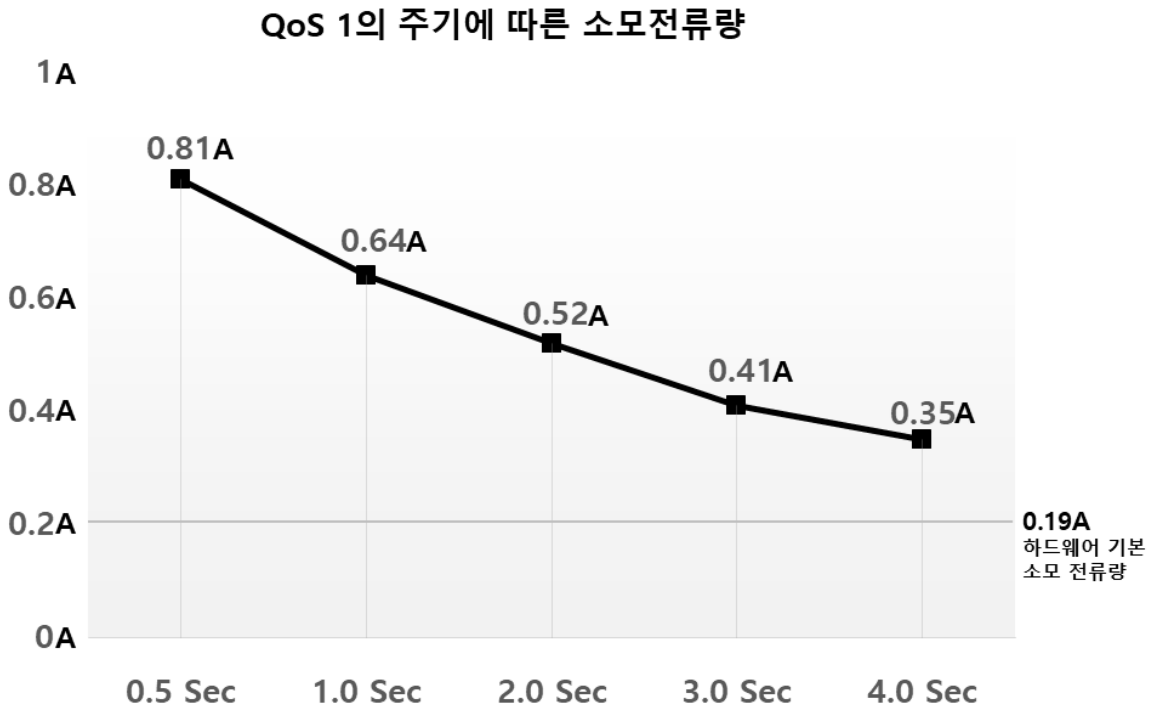
[그림 4-4] QoS 0단계 Publisher의 주기에 따른 CPU 사용률 변화

2. QoS 1단계에서 전송 주기에 따른 CPU사용률 및 전류소모량

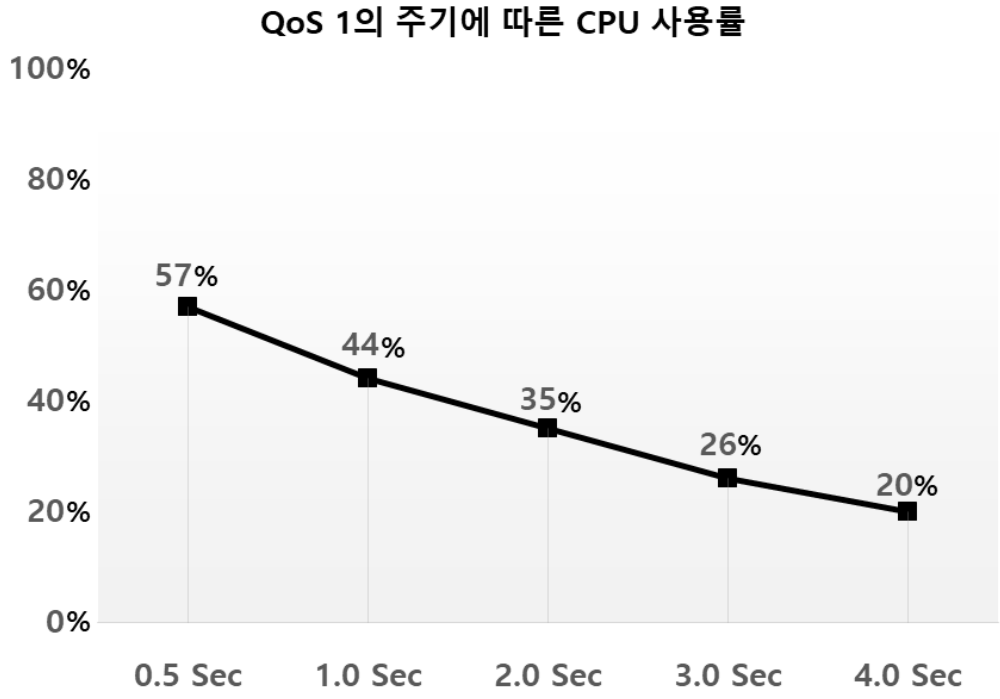
QoS 1단계에서도 전과 같은 방식으로 CPU사용률과 전류소모량을 측정하였다. [표 4-3]와 같이 QoS 1단계에서도 총 1000회를 전송하였으며 주기 변경전후로 각 500회씩 전송하였다. [그림 4-5]와 [그림 4-6]은 Publisher의 주기에 따른 CPU점유율과 전력소모량을 측정한 그래프이다. QoS 1단계에서도 주기가 변경 되면서 CPU점유율과 전력소모량이 줄어드는 것을 확인 할 수 있었다.

[표 4-3] QoS 1단계 CPU사용률 및 전류소모량 측정 설정표

	Value
Topic	“Cycle”
Number of transmissions (Before cycle change)	500 time
Number of transmissions (After cycle change)	500 time
Transmission Message	“hello Sensor” (12byte)
Transmission Frequency (Before / After)	0.5sec / 1.0sec, 2.0sec, 3.0sec, 4.0sec
Added packet data	Boolean (1byte)



[그림 4-5] QoS 1단계 Publisher의 주기에 따른 전력 소모량 변화



[그림 4-6] QoS 1단계 Publisher의 주기에 따른 CPU 사용률 변화

3. QoS 0단계의 데이터 송수신 테스트

[표 4-4] QoS 0단계 송수신 테스트 설정표

	Value
Topic	“SensorTest”
Number of transmissions	3000 times
Transmission Message	“hello Sensor” (12byte)
Transmission Frequency	1.0 sec
Parsing Data	{”test : 192.168.0.1“, “test1 : 192.168.0.2”, “test2 : 192.168.0.3”}

Qos 0단계는 Publisher가 Broker로부터 데이터를 수신 할 수 있는 수단이 없기 때문에 서비스 노드를 구현하여 웹 파싱을 통해 수신을 진행하도록 하였다. 새로운 통신 수단의 추가는 프로세스를 생성하기 때문에 CPU가 처리해야할 테스크가 증가한다. 그래서 MQTT의 기존 QoS 0단계 송수신과 제안하는 방식을 적용한 QoS 0단계 송수신을 CPU사용량과 전력 소모량 측면에서 비교하고 추가적인 통신수단의 추가가 IoT디바이스에 영향을 끼치는지에 대한 테스트를 진행하였다.

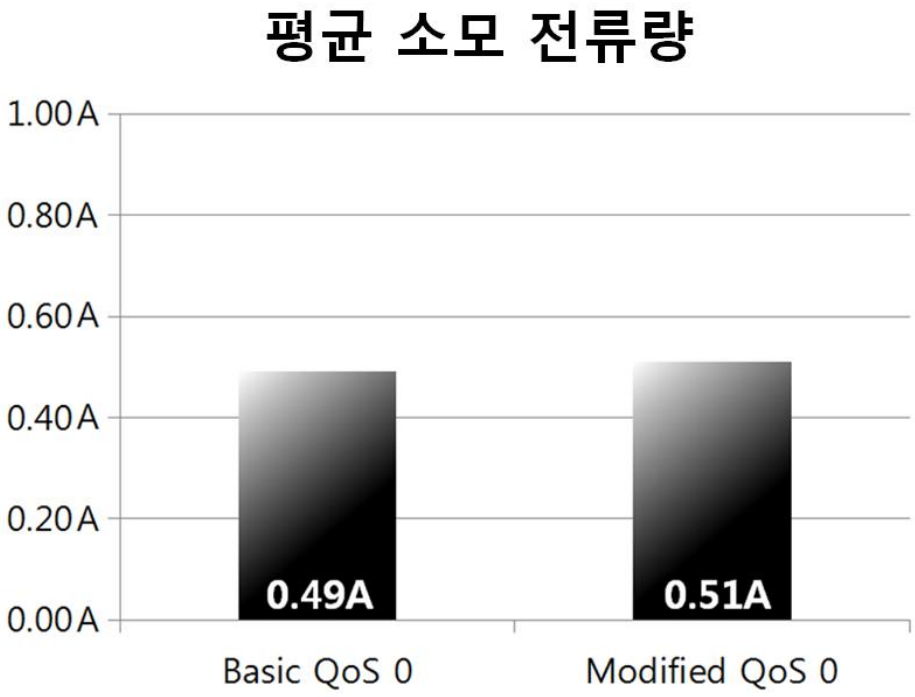
[표 4-4]는 테스트에 사용된 토픽 및 메시지, 전송주기 그리고 파싱하는 데이터의 내용에 대한 설명이다. 데이터는 Broker가 Json형 데이터를 서비스 노드에 전송하였고 Publisher가 파싱을 통해 데이터를 수신하여 Broker의 상황을 인지하도록 진행하였다.

[그림 4-7]은 제안한 방식을 적용하고 측정한 전류소모량을 그래프로 나타낸 것이다. QoS 0단계에서는 서비스 노드로부터 파싱을 통해 데이터를 추가적으로 수신하기 때문에 새로운 통신수단의 추가가 디바이스에 영향을 끼칠 것으로 예상하여 기존 전송방식과의 비교를 통해 큰 차이가 있는지 확인하였다. 기존 방식에서의 평균 전력소모량은 0.49A로 측정되었고 제안한 방식은 0.51A로 측정되었다. 이러한 차이는 전송을 시작하는 초기단계

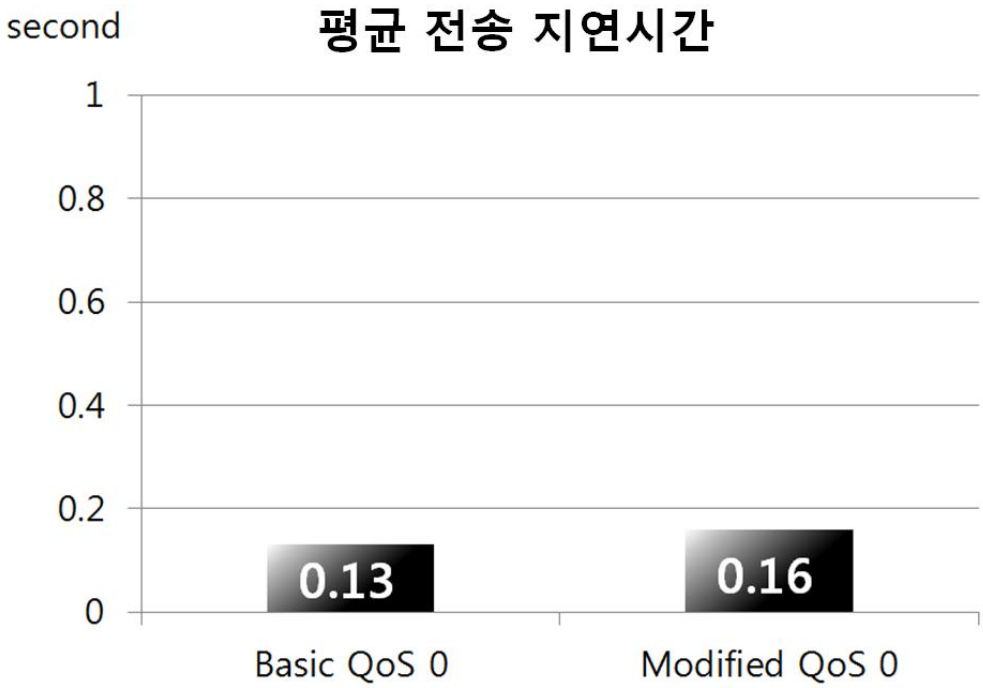
에서 서비스 노드에 연결되면서 발생하는 버퍼링 현상 때문에 제안한 방식의 전류 소모량이 조금 더 높게 측정되었다. 하지만 이러한 차이는 주기에 따른 전류소모량에 영향을 끼칠 정도의 큰 수치가 아니다.

데이터의 송수신은 데이터 속도나 하드웨어의 부하도 중요하지만 데이터가 제 시간에 전송되는지에 대한 사항도 중요하다. [그림 4-8]은 데이터 전송 시 발생하는 지연시간을 기존 방식과 제안한 방식을 비교 측정한 것이다. 기존 방식은 전송 도착까지 0.13sec가 걸렸고 제안한 방식은 0.16sec가 걸렸다. 이는 소모전류량에서 설명하였던 전송 초기에서 발생하는 서비스 노드의 초기 접속 버퍼링이 평균 수치에 영향을 주었다. 주로 초기 전송설정 단계에서만 지연이 발생하고 시간이 지남에 따라 기존방식과의 속도차이가 없기 때문에 전체적인 성능에는 영향을 주지 않는다. 센서 데이터를 수집하고 PUSH서비스에서 사용되는 MQTT 프로토콜에서 0.1sec 단위의 차이는 성능측면에서 큰 영향을 끼치지 않으므로 제안한 방식이 데이터 전송지연 측면에서 문제가 없다는 것을 알 수 있다.

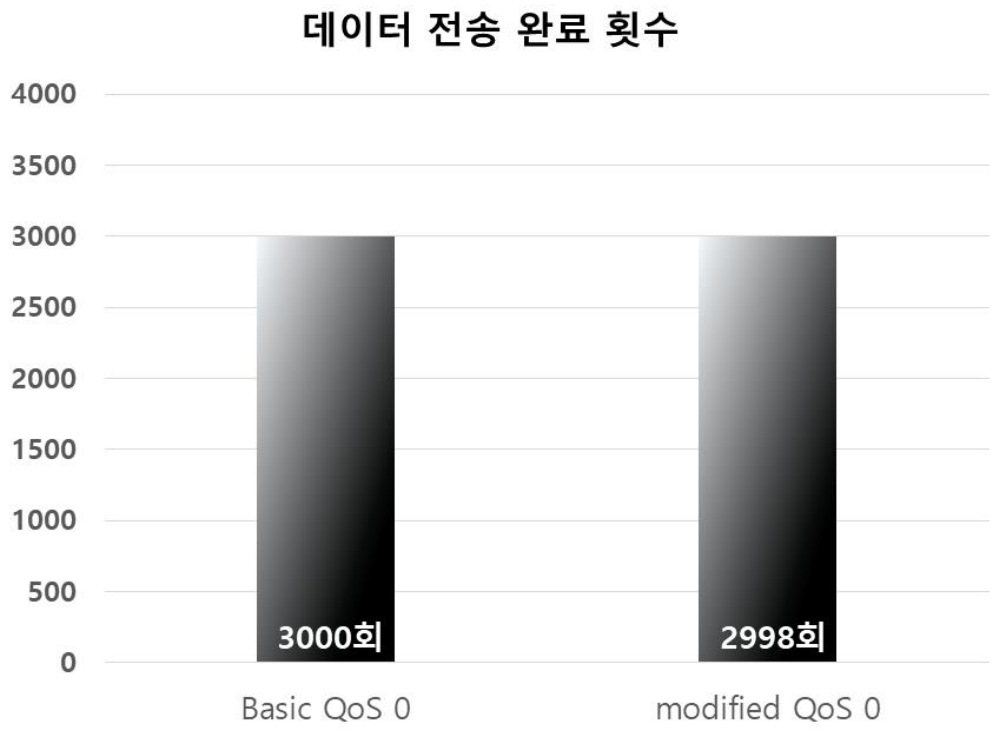
전송시간의 지연도 중요하지만 데이터의 손실 없이 전송할 수 있는지도 중요하다. [그림 4-9]는 제안한 QoS 0단계에서 데이터 손실이 발생하는지 알아보기 위하여 송신과 수신시 이루어질 때마다 횟수를 측정하였다. 총 3000회의 송신 중에서 기존방식은 0건이 발생하였고 제안한 방식은 2건의 손실이 나타났다. 이는 전에 언급한 서비스 노드와 초기접속에서 발생하는 버퍼링이 데이터 전송에 영향을 준 것이다.



[그림 4-7] QoS 0의 평균 전류소모량



[그림 4-8] QoS 0의 평균 데이터전송 지연시간



[그림 4-9] QoS 0의 데이터 전송완료 횟수

4. QoS 1단계의 데이터 송수신 테스트

[표 4-5] QoS 1단계 송수신 테스트 설정표

	Value
Topic	“SensorTest”
Number of transmissions	3000 times
Transmission Message	“hello Sensor” (12byte)
Transmission Frequency	1.0 sec
Added packet data	Boolean (1byte)

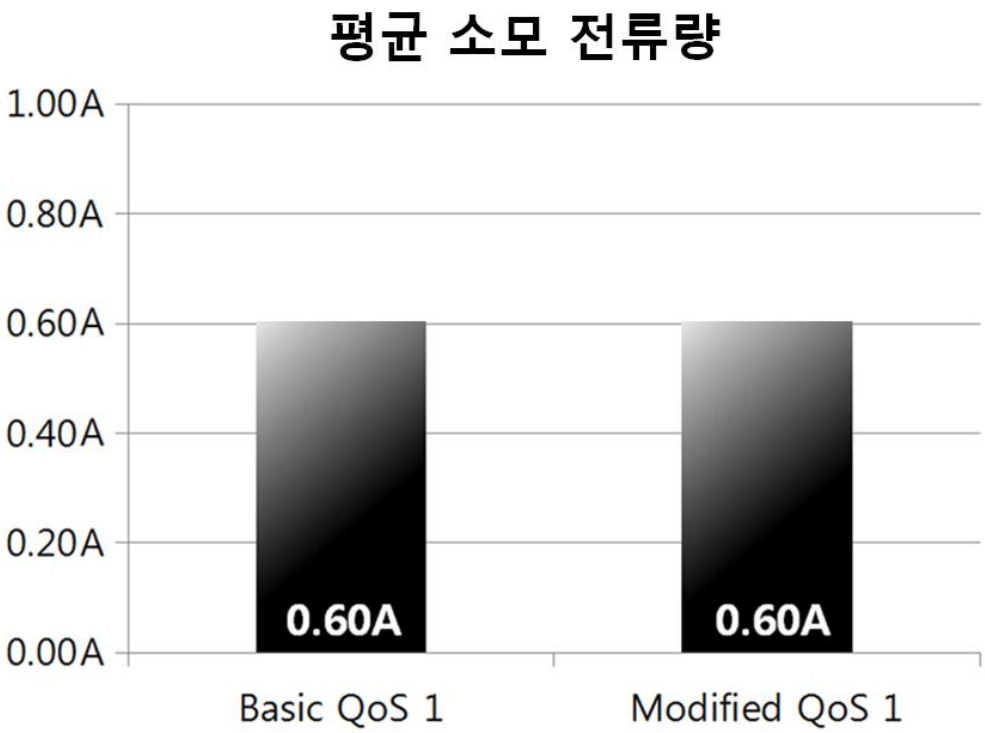
Qos 1단계는 데이터 전송 시 반환되는 PUBACK패킷이 존재하기 때문에 따로 추가적인 통신수단의 추가 없이 패킷의 수정을 통해 Broker로부터 데이터를 수신하도록 하였다. 하지만 패킷의 추가도 용량의 증가가 발생하기 때문에 어떠한 정보를 담는가에 따라 데이터 패킷의 손실이나 지연이 발생 할 수 있다. 이에 따라 패킷의 수정되기 전과 후의 데이터 손실과 지연을 비교하고 CPU사용량의 변화가 있는데 테스트를 진행하였다. [표 4-5]는 테스트에 사용된 토픽 및 메시지와 전송주기에 대한 설명이다. QoS 0단계에서 진행하였던 [표 4-4]의 조건과 파싱 데이터를 제외하면 동일하다.

[그림 4-10]은 QoS 1단계에서의 기존 통방식과 제안한 방식의 전류 소모량을 비교한 그래프이다. QoS 1단계는 기본적으로 패킷을 주고받기 때문에 QoS 0단계보다 전류소모량이 조금 더 높은 경향이 있다. 제안한 방식이 기존방식과 전류소모량의 차이가 없는 것을 확인할 수 있는데 이는 PUBACK 패킷에 1byte의 용량만 추가하였기 때문에 전체적인 송수신에는 영향을 끼치지 않는 것으로 파악된다.

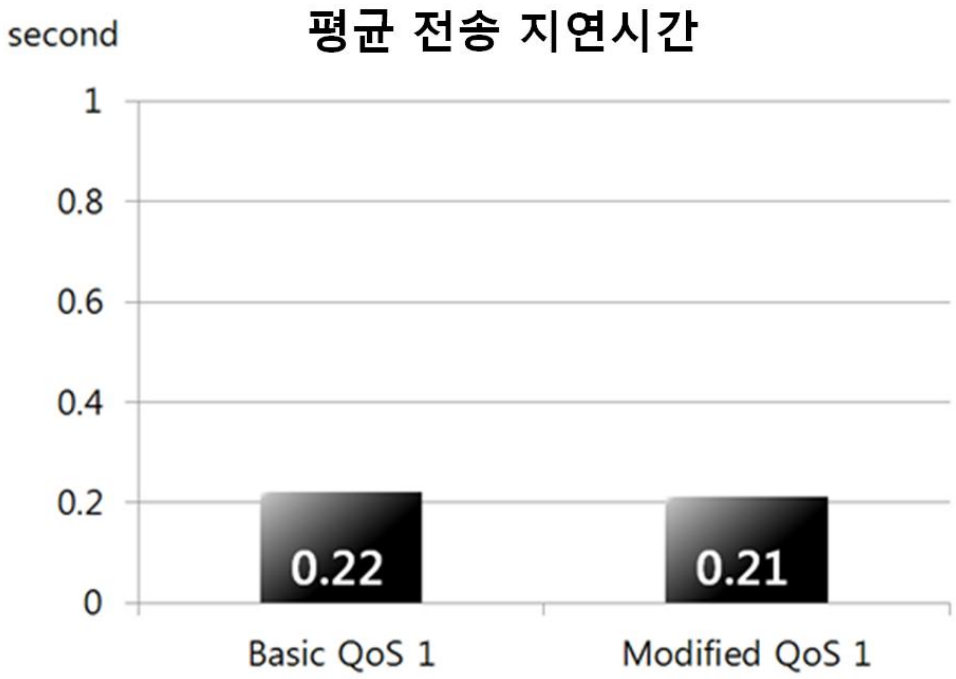
QoS 1단계는 ACK패킷을 통해 데이터의 신뢰성을 파악하기 때문에 데이터 지연이 발생한다. 이에 [그림 4-11]에서는 QoS 1단계의 특성 때문에 패킷의 용량증가가 데이터 전송지

연에 영향을 줄 가능성이 있어서 기존 방식과 비교하여 지연시간을 그래프로 나타내었다. 그래프를 보면 오히려 0.21sec로 기존방식보다 줄어들었다. 용량이 증가했음에도 데이터의 지연이 줄어들었는데 이는 데이터 전송 시 WIFI나 LAN과 같은 외부요인에 영향을 받아서 나타나는 현상이다. 오히려 이는 패킷에 1byte의 추가가 데이터 전송지연 시간에 영향을 끼치지 않는다는 반증이다.

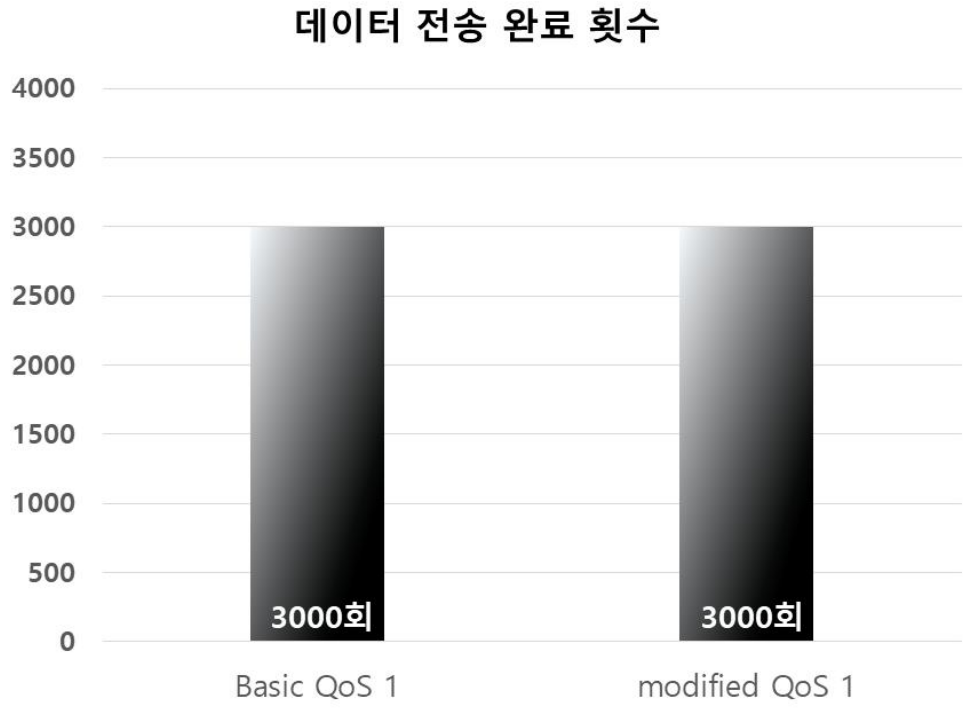
데이터 신뢰성에서 중요한 부분은 데이터가 중간에 유실될 가능성이 있다는 것이다. 이에 제안한 방식에서 데이터의 손실이 있는지 파악하기 위하여 송신과 수신의 카운트타이머를 작동시키고 손실여부를 테스트하였다. [그림 4-12]는 데이터 송신과 수신을 카운트한 값을 표기한 그래프이다. 3000회의 송신 중에서 0건의 손실을 확인할 수 있었다.



[그림 4-10] QoS 1의 평균 전류소모량



[그림 4-11] QoS 1의 평균 데이터전송 지연시간



[그림 4-12] QoS 1의 데이터 전송완료 횟수

Ⅴ. 결론 및 제언

본 논문에서는 기존 MQTT 송수신시 발생하는 불필요한 전력소모를 감소시키기 위해 Publiser의 데이터 전송 주기를 조절하기 위한 방안으로 MQTT가 제공하는 QoS기능의 단계에 따라 Publisher가 Broker의 통신상황을 인지하여 전송 주기를 조절하도록 하였다.

QoS 0단계에서는 ACK패킷이 존재하지 않아 일 방향 통신으로 진행되어 새로운 통신수단이 필요했기 때문에 Nodejs를 사용해 서비스 노드를 구현하였고 Publisher가 Web으로부터 데이터를 수신 받아 송신 시 Broker의 통신상황을 알 수 있도록 하였다.

QoS 1단계는 송수신 할 때 반환되는 PUBACK패킷의 구조를 변경하여 추가한 새로운 바이트공간에 정보를 담아 반환시켜 Publisher가 Broker의 통신상황을 인지 할 수 있게 하였다. 기존 2바이트의 고정헤더를 가지고 있는 PUBACK 패킷에 Topic의 존재유무를 알려주는 플래그를 추가하여 총 3바이트 형태로 주고받도록 하였다.

QoS에 따라 제안한 방법을 적용하여 직접 테스트를 진행하였다. QoS에 따라서 Publisher의 전송 주기를 수신자의 유무에 따라 0.5sec에서 1.0sec, 2.0sec, 3.0sec, 4.0sec 4가지로 변경하였고 해당 IoT장비의 CPU점유율과 소모전류량을 측정하여 감소하는 효과가 있는 것을 확인 할 수 있었다. 그리고 제안하는 방식이 기존방식보다 신뢰성이 떨어 질 수 있다는 것을 감안하여 데이터 전송지연시간과 전류소모량 및 데이터 손실 여부를 측정하여 비교하였고 데이터 송수신에 영향을 끼칠 정도로 큰 차이가 없다는 것을 확인 할 수 있었다.

이러한 Publisher의 불필요한 전력소모를 줄이는 것은 제한된 컴퓨팅 환경을 가진 IoT디바이스의 성능향상에 도움이 될 것으로 기대한다. 향후 연구로는 극단적인 하드웨어 환경에서의 통신 신뢰도 테스트를 통해 제한된 컴퓨팅 환경에서 더욱 적합한 통신형태에 대해 연구할 예정이다.

참고문헌

[1] Al-Fuqaha, Ala, et al. "Internet of things: A survey on enabling technologies, protocols, and applications." IEEE Communications Surveys & Tutorials 17.4 (2015): 2347-2376

[2] Ryu Kuem Gang. "A Study on the Comparison of Reliability Between MQTT and CoAP (Confirmative message)" .Proceedings of Symposium of the Korean Institute of communications and Information Sciences, 2017.11, 522-523 (2 pages)

[3] Lee, Shinho, et al. "Correlation analysis of MQTT loss and delay according to QoS level." The International Conference on Information Networking 2013 (ICOIN). IEEE, 2013.

[4] Xu, Yiming, V. Mahendran, and Sridhar Radhakrishnan. "Towards SDN-based fog computing: MQTT Broker virtualization for effective and reliable delivery." 2016 8th International Conference on Communication Systems and Networks (COMSNETS). IEEE, 2016

[5] Github(웹사이트), https://wnsgml972.github.io/mqtt/mqtt.html, Accessed Oct 10, 2018

[6] IBM(웹사이트), https://www.ibm.com/support/knowledgecenter/ko/SSFKSJ_7.5.0/com.ibm.mm.tc.doc/tc60340_.htm, Accessed Oct 10, 2018

[7] Yun-Hee Kang. "Design of a Seamless Messaging Application using WebSocket in IoT Environment". Proceedings of KIIT Summer Conference, 245-247.

[8] 심상민. "WebSocket과 Socket.io", Naver D2, 2011

[9] MQTT(docs), http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html

[10] Peter lubbers, Frank greco. "HTML5 Web Sockets: A Quantum Leap in Scalability for the Web." Microservices Expo : Article, 2010

[11] Joo-hyeon Kim, Jae-hun Kim. "Proposal of MQTT Broker's adjustment of pub / sub cycle through reinforcement learning." The Korean Operations Research and Management Science Society (2018.04) : 628-633

[12] Seung-woo Kang. "Characterizing Power Consumption of MQTT Protocol

Usage on Raspberry Pi" The Korea Institute of Information and Communication Engineering (2017.12) : 2347-2356

[13] Sang-hyun Kim, Dong-hwi Kim, Hyeung-seok Oh, Hyun-sig Jeon, Hyun-ju Park. "The Data Collection Solution Based on MQTT for Stable IoT Platforms". Journal of the Korea Institute of Information and Communication Engineering 20(4), 2016.4, 728-738 (11 pages)

[14] Sung-jin Kim, Chang-heon Oh. "Method for Message Processing According to Priority in MQTT Broker". Journal of the Korea Institute of Information and Communication Engineering 21(7), 2017.7,1320-1326 (7 pages)

[15] Seungwoo Kang. "Characterizing Power Consumption of MQTT Protocol Usage on Raspberry Pi". Journal of the Korea Institute of Information and Communication Engineering 21(12), 2017. 12, 2347-2356 (10 pages)

[16] Se Jong Lee, Joohan Park, Jaewon Noh, Sunghyun Cho. "MQTT Broker caching to reduce processing burden of IoT sensors". Proceedings of the Korean Society of Computer Information Conference 26(2), 2018.7, 223-224 (2 pages)

[17] Yue Ma. Ruiyang Yan. Jianwei Sun. Kaifeng Yao. "A MQTT Protocol Message Push Server Based on RocketMQ", Intelligent Computation Technology and Automation (ICICTA), 2017 10th International Conference on 2017 Oct, 2017, pp.295 - 298

[18] hangavel Dinesh, Xiaoping Ma, Valera Alvin, Hwee-Xian Tan, Tan Colin Keng-Yan, "Performance evaluation of MQTT and CoAP via a common middleware", Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on 2014 Apr, 2014, pp.1 - 6

[19] Shinho Lee, Hyeonwoo Kim, Dong-kweon Hong, Hongtaek Ju. "Correlation analysis of MQTT loss and delay according to QoS level" Information Networking (ICOIN), 2013 International Conference on 2013 Jan, 2013, pp.714 - 717

[20] Yokotani Tetsuya, Sasaki Yuya. "Comparison with HTTP and MQTT on required network resources for IoT". Control, Electronics, Renewable Energy

and Communications (ICCEREC), 2016 International Conference on 2016 Sept, 2016, pp.1 - 6

[21] Github(웹사이트), https://github.com/andsel/moquette, Accessed Nov 1, 2018

[22] Eclipse(웹사이트), https://www.eclipse.org/paho/, Accessed Nov 2, 2018