Journal of Korea Multimedia Society Vol. 21, No. 9, September 2018(pp. 1090-1098)
https://doi.org/10.9717/kmms.2018.21.9.1090

RPL 기반 분산 MQTT 브로커 구조 설계 및 구현

안현성[†], 사우진[††], 김승구[†††]

Design and Implementation of RPL-based Distributed MQTT Broker Architecture

Hyunseong An[†], Woojin Sa[††], Seungku Kim[†††]

ABSTRACT

MQTT is a lightweight messaging protocol that can be used for low power IoT devices. The MQTT basically uses single MQTT broker to indirectly share message information between publishers and subscribers. This approach has a weakness in regard to traffic overflow, connection fault, security, etc. In this paper, we propose a distributed MQTT broker architecture that solves the problems in single MQTT broker structure. The distributed MQTT broker architecture is expected to support new application services that cannot be supported by a conventional MQTT architecture. We have designed and implemented a distributed MQTT broker architecture based on the RPL protocol that has been widely used for IoT network. The experiment results show that the proposed MQTT broker architecture represents better publishing/subscribing latency and network stability than the conventional MQTT broker architecture.

Key words: IoT, MQTT, RPL, WSN

1. 서 론

Internet of Things(IoT)는 통신 가능한 각종 사물들을 인터넷으로 연결시키는 기술이다. 통신 가능한 사물로는 스마트폰, 가전제품, 웨어러블 장치, 센서 등의 다양한 종류의 임베디드 장치들이 있다. 최근 IoT는 다양한 산업분야에 적용되어 스마트 시티, 스마트 홈, 스마트 그리드, 스마트 팩토리 등과 같은 새로운 서비스에 활용되고 있다. IoT를 위해 다양한 통신 프로토콜이 개발되었으며 가까운 미래에 IoT 기기의 수가 기하급수적으로 늘어날 것으로 기대된다. IoT 기기를 인터넷과 연결하기 위해서는 반드시 Internet Protocol(IP)주소가 필요하다. 하지만 기존 IPv4 주소체계로는 기하급수적으로 늘어난 IoT 기기들에 주소를 할당하는 것이 어려워 대부분의 IoT 프로토콜들은 IPv6 주소체계를 활용하고 있다.

Message Queuing Telemetry Transport(MQTT)[1,16]는 TCP/IP 기반의 경량 메시징 프로토콜로 센서와 같이 기능이 제한된 IoT 장치에 활용 가능한 기술이다. MQTT는 서버 역할을 하는 브로커를 통

※ Corresponding Author : Seungku Kim, Address: (28644) Chungbuk National University, Chungdaero 1, Cheongju-si, Chungbuk, Korea, TEL : +82-43-261-2479, E-mail : kimsk@cbnu.ac.kr
Receipt date : May 30, 2018, Revision date : Jul. 30, 2018
Approval date : Aug. 14, 2018
[†] Dept. of Electronic Eng., Graduate School, Chungbuk National University
(E-mail : ahj3460@cbnu.ac.kr)
[††] College of Electrical & Computer Eng., School of Electronics Engineering, Chungbuk National University
(E-mail : wjin444@cbnu.ac.kr)
[†††] Dept. of Electronic Eng., Graduate School, Chungbuk National University
※ This research was supported by Korea Electric Power Corporation. (Grant number: R17XA05-69)

해 발행/구독(publish/subscribe) 기반으로 토픽 기반의 메시지를 주고받는다. 토픽을 브로커에 발행하기 위한 publisher는 주로 저전력 센서들을 활용한다. Publisher는 지정된 하나의 MQTT 브로커에 이벤트 발생시 또는 주기적으로 토픽을 발행한다. Subscriber는 MQTT 브로커에 발행된 토픽을 구독한다. 이러한 단일 MQTT 브로커를 활용한 토픽 공유 방식은 트래픽 과부하, 연결 오류, 보안 등과 같은 문제를 발생시킬 수 있다. 기존의 구조는 하나의 브로커를 중심으로 운용된다. 단일 브로커에 연결된 모든 기기들의 토픽을 종합적으로 관리 및 사용 할 수 있다. 하지만 동시에 다수의 기기들이 하나의 브로커와 연결되면 트래픽 과부하가 일어날 수 있다. 또한 브로커의 보안이 취약하면 브로커가 가진 모든 정보가 유출된다. 그리고 브로커와의 연결 오류가 발생하면 시스템이 멈추게 된다. 이러한 문제를 해결하기 위해 다수의 MQTT를 활용하여 정보를 분산시키는 방법[2]과 MQTT 브로커 브릿지를 이용하여 브로커 간 메시지를 교환하고 백업하는 방법[3]이 활용되고 있다. 하지만 이러한 방법들로 기존 문제를 모두 해결하기는 어렵다. 제안하는 분산형 브로커는 다수의 브로커를 운용하는 형식이다. 그렇기 때문에 트래픽 과부하 및 보안문제 해결이 가능하다. 또한 분산형 브로커가 고장난다면 기존의 단일 브로커로 연결하여 시스템의 유지가 가능하다. 하지만 다수의 브로커를 운용하기 때문에 설비의 비용이 높아지게 된다.

본 논문에서는 단일 MQTT 브로커에서 발생 가능한 문제를 해결하기 위해 분산 MQTT 브로커 구조를 제안한다. 분산 MQTT 브로커 구조를 통해 publisher와 subscriber는 위치나 환경에 따라 다른 MQTT 브로커에 토픽을 발행/구독한다. 본 논문에서는 RPL(IPv6 Routing Protocol for Low-power and Lossy networks) [4] 기반으로 분산 MQTT 브로커 구조를 설계하고 구현한다. RPL은 IPv6 기반 라우팅 프로토콜로 센서와 같은 경량 IoT 기기들이 많이 활용한다. RPL은 트리 구조로 네트워크를 구성한다. 트리의 루트는 6LoWPAN Border Router (6LBR)로 자식 노드들로부터 정보를 수집하여 인프라 네트워크로 전달하는 게이트웨이 역할을 한다. 제안하는 분산 MQTT 브로커 구조를 지원하기 위해 6LBR마다 활용 가능한 MQTT 브로커의 IP 주소를 저장한다. 6LBR과 연결된 자식 노드는 6LBR이 저장하고 있는 MQTT 브로커의 IP주소나 자신이 저장하고 있는 MQTT 브로커의 IP 주소 중 위치나 환경에 따라 선택이 가능하다. 이러한 분산 MQTT 브로커 구조로 동작하기 위해 RPL 프로토콜의 Destination Advertisement Object(DAO) 메시지의 8비트 reserved 영역 중 1bit를 이용한다. 표준 기반으로 제안된 방식이기 때문에 기존 RPL 프로토콜로 동작하는 장치들과 호환이 가능하다는 장점을 갖는다.

제안하는 분산 MQTT 브로커 구조 구현을 위해 CC2650 sensortag 1.3 [5]과 raspberry pi 2.0 [6]을 사용한다. Sensortag를 이용하여 RPL 노드를 구현하고, raspberry pi와 sensortag를 결합하여 6LBR을 구현한다. Sensortag는 RPL과 MQTT 프로토콜이 구현된 Contiki 3.0 [7] OS를 사용하고 MQTT 브로커는 Mosquitto[8]를 활용하여 노트북에 구현한다. 라우팅 거리에 따른 MQTT 토픽 발행시간을 비교하는 실험을 통해 분산 MQTT 브로커로 토픽 발행시간이 짧은 것을 확인할 수 있었다. 실시간 보장이 필요한 서비스에 분산 MQTT 브로커가 활용 가능할 것으로 기대된다. Publisher나 subscriber가 이동 가능한 장치라고 한다면 6LBR과 재연결에 대한 고려가 필요하다. 실험을 통해 재연결에 필요한 시간을 분석하고 응용 요구사항에 따라 재연결 시간 조절이 가능한 것을 확인한다. 마지막으로 MQTT 브로커에 트래픽이 집중되었을 때 메시지 전송 지연시간에 대해 확인한다. 이를 통해 분산 MQTT 브로커의 필요성을 확인한다.

본 논문은 다음과 같이 구성된다. 2장에서는 관련 기술로 MQTT의 전체적인 특징과 개요, 기능들을 서술하며 WSN 라우팅 프로토콜인 RPL의 동작 방식에 대해 소개한 후 3장에서 기존의 단일형 MQTT 브로커의 문제점과 그에 관한 해결책들, 그리고 분산형 브로커의 필요성과 그 동작방식, 필요성을 부각하는 실험들의 결과와 대해 서술하고 마지막 4장에서 결론을 맺는다.

2. 관련 연구

본 장에서는 MQTT와 RPL에 관한 기본적인 동작과정과 특징에 대해 소개한다.

2.1 Message Queuing Telemetry Transport(MQTT)

MQTT는 TCP/IP 기반으로 동작하는 IoT용 경량

메시지 프로토콜로서 IBM에서 개발하여 2013년 OASIS에서 표준화하였다. 2015년 10월 MQTT 3.1.1 버전에 배포되어 다양한 IoT 응용에서 사용 중이다. MQTT 브로커를 통해 Publish/Subscribe 방식으로 메시지를 교환하여 리소스 점유를 최소화하고, 낮은 전력, 낮은 대역폭, 낮은 지연시간의 특징을 갖는다. IPv6 주소체계 사용이 가능하도록 설계되어 향후 IP 주소고갈 문제가 발생하더라도 계속 사용이 가능한 프로토콜이다. Facebook 메신저[9], 아마존 웹 서비스, 마이크로소프트 Azure[10]와 같은 프로젝트에서 MQTT를 활용중이다.

MQTT는 Fig. 1과 같이 브로커, Publisher, Subscriber 3가지의 구성요소를 갖는다. Publisher는 토픽이라는 계층적 구조를 갖는 메시지를 MQTT 브로커에 전달하는 역할을 한다. MQTT 브로커는 publisher와 메시지 버스를 생성하고, 이 버스를 통해 토픽 정보를 저장, 관리한다. Publisher는 특정 이벤트나 주기적으로 MQTT 브로커에 토픽을 발행한다. Subscriber는 MQTT 브로커가 저장하는 토픽을 구독한다. 토픽의 계층적 구조를 활용하여 subscriber는 한 번에 관련된 다수의 토픽을 수신할 수 있어 다수의 IoT 기기로부터 정보를 효과적으로 수집 가능하다.

MQTT는 서비스 종류에 따라 메시지 신뢰성을 보장하기 위해서 총 3단계의 Quality of Service (QoS)를 지원하여 서비스 특성에 따라 메시지 관리가 가능하다. QoS 0단계는 publisher가 메시지를 한 번 전송한 후 즉시 연결을 종료하는 방식이다. 전송속도가 가장 빠르지만 MQTT 브로커에 메시지가 성공적으로 전송 되었는지의 여부를 확인할 수 없어서 메시지가 손실될 가능성이 있다. QoS 1단계는 publisher가 메시지를 전송한 후에 MQTT 브로커로부터 메시지를 받았다는 응답을 받은 뒤 연결을 종료하는 방식이다. MQTT 브로커에게 응답을 받을 때까지 계속 메시지를 전송하기 때문에 메시지가 손실될 가능성은 낮지만 메시지가 중복될 가능성이 발생한다. QoS 2단계는 QoS 1단계와 같이 publisher가 메시지를 전송하고 MQTT 브로커로부터 메시지를 받았다는 응답을 받는다. 그리고 연결을 종할 때 추가적으로 MQTT 브로커로부터 연결 종료에 대한 응답을 받은 뒤 연결을 종료한다. 이를 통해 메시지 중복 문제는 해결 가능하지만 추가적인 메시지 교환이 필요하기 때문에 전송속도는 가장 느린 방식이다. MQTT 메시지의 기본 고정 헤더의 크기는 최소 2byte로 매우 간소화 되어 있어 저전력 통신에 활용 가능하다. IBM에서 발표한 자료[11]에 따르면 MQTT는 HTTP에 비해 1/10~1/100 크기의 트래픽으로 10~100배의 처리량을 나타내며 배터리 소모량은 1/10이하로 줄어든다. MQTT는 보안을 위한 간단한 사용자 이름과 암호를 설정하여 MQTT 브로커에 연결을 하는 방법을 제공한다. MQTT는 C로 구현한 Mosquitto [8], Java, Python 등 다양한 언어를 지원하는 IMB Paho [12], HiveMQ [13] 등 이 있다. 본 논문에서는 Mosquitto를 이용하여 MQTT 브로커를 구현한다.

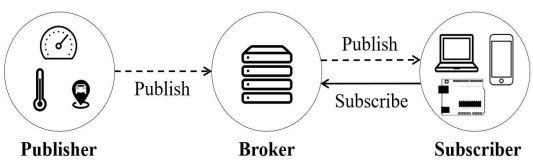


Fig. 1. MQTT Publish/Subscribe architecture.

단일 MQTT 브로커 구조로 발생하는 취약점을 해결하기 위한 몇 가지 해결책들이 있다. 스마트 유지보수 서비스(M Maritsch et al.,2016)[2]에서는 지역, 네트워크 별로 계층을 형성하고 각 계층마다 MQTT 브로커를 설치한다. 하위 계층의 MQTT 브로커가 토픽을 수집하고 상위 계층의 MQTT 브로커에 토픽을 다시 전달해주는 방식으로 다수의 MQTT 브로커가 토픽을 저장할 수 있는 구조를 설계하였다. 이러한 계층적 구조를 활용하여 위험부담을 분산시키는 해결책을 제시하였다. 이 외에도 MQTT 브로커의 내부에 토픽의 계층적 특징을 이용한 추가적인 제어시스템인 Topic Access Control System(TACS)을 설치하여 특정 토픽에 접근하려는 노드를 대상으로 액세스 권한을 부여하거나 거부하는 기능을 추가하여 보안을 강화하였다. Mosquitto[3]는 단일 MQTT 브로커의 위험 분산을 위해 MQTT Bridge 기능을 제공한다. MQTT Bridge는 두 개의 MQTT 브로커가 토픽 정보를 공유하여 위험을 분산시킬 수 있다. SMQTT[14]는 기존의 MQTT 프로토콜의 보안을 강화한 방법이다. MQTT 발행 메시지 헤더부분의 4bit reserved 공간을 활용하여 경량화 타원 곡선 암호화법(Elliptic Curve Cryptography, ECC)을 기반으로 한 ABE 암호화법을 적용시킨 SPublish를 제안한다. 이러한 보안의 강화로 악의적인 공격에 대학

취약점이 보완이 가능하며 기존 MQTT 데이터 포맷의 reserved 공간 활용으로 데이터 크기에 변화 없이 적용이 가능하다. 하지만 암호 해독을 위한 Key 값을 교환하는 과정이 필요하기 때문에 메시지 지연시간이 길어진다는 단점이 있다.

2.2 IPv6 Routing Protocol for Low-power Lossy networks(RPL)

RPL은 저전력, 저품질 네트워크 환경인 LLNs (Low power and Lossy Networks)에서 동작하는 IPv6 라우팅 프로토콜이다. 네트워크 토폴로지는 방향성 비순환 형태(Directed Acyclic Graph, DAG)이며 트리 구조와 유사하다. 라우팅의 최종목적지는 하나의 노드인 Destination Oriented DAG(DODAG)가 된다. 즉, RPL은 최종 목적지인 루트 노드와 정보를 수집하는 센서 노드로 구성되는데, 루트 노드는 다른 네트워크와 연결 가능한 게이트웨이 역할을 한다. 센서 노드들은 주변의 모든 이웃노드를 탐색하고, 그 중 연결효율이 가장 좋은 노드를 부모로 선택하여 네트워크에 참여한다. 루트 노드로부터 떨어진 정도를 랭크라고 하며 루트 노드와 가장 가까운 노드의 랭크는 1이 된다. RPL의 최종 목적은 항상 최적의 트리를 유지하는 것인데 이를 위해 노드 간 지속적으로 패킷을 주고받는다. 트리 네트워크로부터 떨어져 루트 노드로 연결이 될 수 없는 노드들은 플로팅 되었다고 하며 물리적으로 위치를 바꾸어주거나 새로운 노드를 이용하여 루트 노드로의 연결을 해야한다. 연결이 안정화 되면 각 노드들은 어떠한 식으로 라우팅이 되어야하는지 저장해놓는 라우팅 테이블을 업데이트하고 테이블을 참조하여 라우팅을 수행한다. 트리를 구성하는 RPL 프로토콜에 사용되는 패킷은 4개 있다.

2.2.1 DODAG Information Solicitation(DIS)

DIS 패킷은 DIO 패킷을 요청하기 위해 사용하는 패킷으로 DIO 패킷을 정상적으로 받지 못하거나, 새로운 노드가 트리에 참여하고자 할 때 주변 노드에 이 패킷을 보낸다. 해당 패킷을 받은 트리의 구성 노드는 DIO 패킷을 송신한다.

2.2.2 DODAG Information Object(DIO)

DIO는 노드가 주기적으로 멀티캐스트 또는 유니캐스트 하는 패킷이다. 멀티캐스트 패킷은 모든 자식 노드들에게 주는 패킷이다. 해당 패킷으로 노드의 하위 트리 연결을 유지하게된다. 유니캐스트 패킷은 부모노드와의 연결유지 및 확인을 위한 패킷이다. 해당 패킷들에는 노드가 소속되어있는 트리의 종합적인 정보가 들어있다. 이러한 정보들을 이용하여 랭크의 최신화를 통해 보다 통신이 원활한 트리를 구성하게 된다. RPL의 표준문서인 RFC6550[4]에서 DIO 송신 간격에 대해서 정의한다. 최소 2^3ms부터 시작하여 최대 2^{3+x}ms로 x의 값을 1씩 늘리며 간격을 증가시킨다. x의 기본값은 20으로 정의했으며 이는 간격의 최대시간이 2.3시간임을 뜻한다. 또한 송신간격의 초기화는 특정 상황들로 제한되어있다. 노드로부터 DIS의 패킷을 수신하거나 새로운 노드가 추가된 것이 감지가 되면 최소값으로 돌아가게된다. 또한 네트워크의 오류가 감지가 되면 네트워크의 안정성을 높이기위해 최소값으로 돌아가게된다. 그렇기에 송신 간격이 크게 유지될수록 네트워크의 안정성이 높다고 할 수 있다.

2.2.3 Destination Advertisement Object(DAO)

DAO는 자식 노드로서 트리에 참여를 요청하기 위한 패킷이다. DIO 패킷을 수집을 통해 정한 일정 랭크 이상의 노드에게 DAO 패킷을 보낸다. DAO 패킷을 통해 연결된 모든 노드의 정보를 루트 노드에게 전달하여 최적의 경로를 결정한다.

2.2.4 DAO-ACK

DAO-ACK 패킷은 DAO 패킷에 대한 응답으로 그 노드와 연결이 가능한지의 여부를 응답해주는 패킷이다. 해당 패킷을 받은 노드는 자식노드로 연결되어 트리 구조가 형성된다.

3. RPL 기반 분산 MQTT 브로커 구조

본 장에서는 제안하는 RPL 기반 분산 MQTT 브로커 구조에 대해 설계한다.

3.1 단일 MQTT 브로커 구조의 문제

MQTT publisher는 하나의 지정된 MQTT 브로커에 토픽을 발행한다. 이러한 단일 MQTT 브로커 구조는 몇 가지 취약점이 존재한다 [2]. 단일 MQTT

브로커는 모든 토픽 정보가 하나의 MQTT 브로커에 집중되기 때문에 위험부담이 크다. MQTT 브로커의 연결이 끊어지는 경우 모든 publisher와 subscriber의 발행과 구독이 불가능해진다. 이는 서비스 품질의 저하를 야기할 수 있다. 악의적 사용자에 의해 MQTT IoT, MQTT, RPL, WSN, 6LowPAN브로커 보안 시스템이 해킹되면 토픽 삭제 또는 임의 변경이 쉽게 일어날 수 있다. 그리고 하나의 MQTT 브로커에 트래픽이 집중되게 되면 발행과 구독의 지연시간이 늘어나 서비스 품질이 떨어질 수 있다. 예를 들어 병원에서 환자 상태를 모니터링하기 위한 응용을 MQTT로 사용시 이러한 단일 MQTT 브로커 사용으로 인한 문제는 자칫 큰 사고로 이어질 수 있다. 따라서 이러한 위험부담을 분산시키는 방법에 대한 고려가 필요하다.

실시간으로 서비스를 제공해야하는 응용의 경우 데이터 지연시간에 의해 서비스 품질이 결정된다. 데이터 지연시간을 줄이기 위해서는 publisher와 subscriber의 거리가 MQTT 브로커와 가까운 것이 유리하다. 거리가 가까울수록 네트워크에 의한 지연시간이 줄어들기 때문에 위치와 환경을 고려하여 MQTT 브로커의 분산이 필요하다.

3.2 분산 MQTT 브로커 구조의 필요성

다수의 MQTT 브로커를 통해 위험부담을 분산시켜 단일 MQTT 브로커 구조에 의한 문제를 해결 할 수 있다. 하지만 다수의 MQTT 브로커 사용으로 인해 메시지 지연시간이 길어진다는 단점이 있다. 본 논문에서는 IoT 장치의 위치나 환경에 따라 분산적으로 MQTT 브로커를 선택할 수 있는 구조를 제안한다. 분산적으로 MQTT 브로커를 배치하는 경우 위험부담을 분산하여 단일 MQTT 브로커 구조에 의한 문제를 해결할 수 있으며 위치나 환경에 따라 분산 MQTT 브로커를 선택하기 때문에 메시지 지연시간도 줄일 수 있다. 제안하는 구조는 publisher나 subscriber가 이동하는 개체일 때 유용한 기술로 다양한 응용에서 활용 가능할 것으로 기대된다. 예를 들어 전기차 충전소 관리를 위한 응용에서 MQTT를 활용하는 경우 전기차는 publisher 역할을 하여 충전설비를 통해 배터리 충전률 등과 같은 정보를 발행한다. 전기차 충전소는 전기차가 발행한 토픽 구독을 통해 전체 충전소의 사용현황과 예상 대기시간 등에 대한 정보를 수집할 수 있다. 반대로 전기차가 가장 가까운 충전소의 정보를 받는 것 또한 가능하다. 단일 브로커의 구조에서는 모든 충전소의 현황정보를 받아오게 된다. 분산형 브로커에서는 연결된 브로커에 있는 소수의 충전소 정보만 가져와 가까운 충전소의 사용가능 여부 등의 충전소 정보를 구독할 수 있다. 이러한 응용기술은 단일 MQTT 브로커를 사용하는 경우 많은 제약사항이 따른다. 기기가 접속한 위치를 알기위한 GPS 장치가 추가적으로 요구된다. 또한 브로커가 다수의 충전소 정보를 저장하게 된다. 이는 토픽의 계층이 많아지고, 데이터 관리의 복잡성을 유발시킨다. 제안하는 구조는 단일 MQTT 브로커에서 발생하는 취약점을 해결 가능하며 새로운 응용 모델에도 활용할 수 있는 기술이다.

3.3 RPL 기반 분산 MQTT 브로커 구조 설계

본 절에서는 제안하는 분산 MQTT 브로커 구조에 대해 설명한다. 분산 MQTT 브로커 구조에서는 IoT 장치들의 위치나 환경에 따라 가까운 거리에 있는 지역 MQTT 브로커에 토픽을 발행하거나 구독한다. 이 기술은 응용 모델에 따라 단일 MQTT 브로커 구조 또는 분산 MQTT 브로커 구조로 동작 가능하다. 본 논문에서는 RPL 기반으로 분산 MQTT 브로커 구조를 설계한다.

모든 IoT 장치는 이동성을 갖는 개체로 가정하고 있으며 위치마다 연결되는 게이트웨이가 달라진다. Fig. 2는 IoT 장치가 게이트웨이 1번에 연결된 상태에서 이동하여 게이트웨이 2번에 연결되는 예다. 게

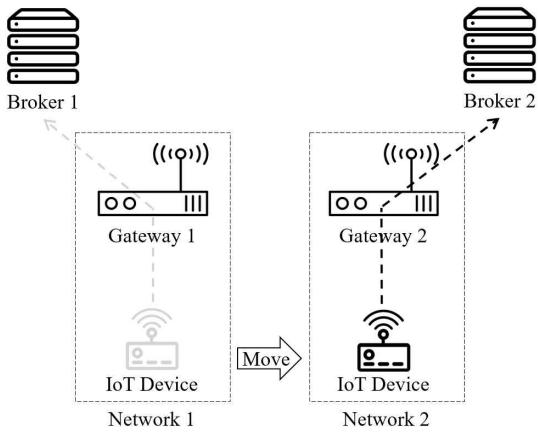


Fig. 2. Distributed MQTT broker's architecture.

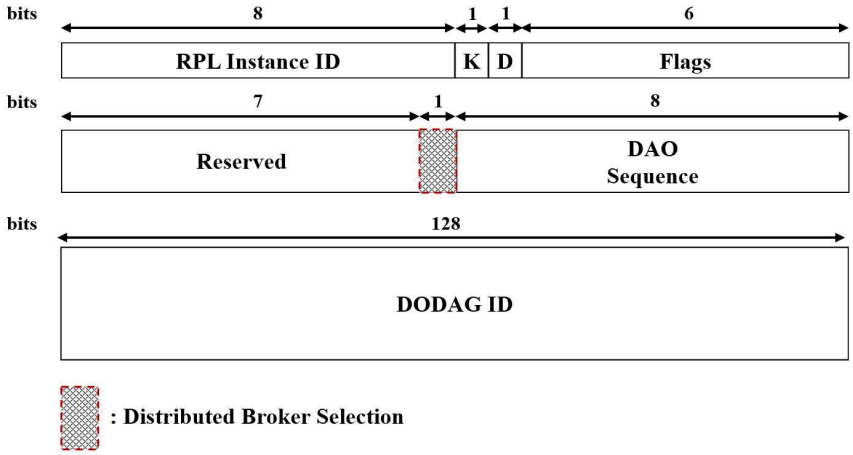


Fig. 3. RPL DAO packer format and Distributed Broker Selection (DBS).

이트웨이 1번의 경우 분산 MQTT 브로커 1번에 대한 IP 주소를 저장하고 있기 때문에 IoT 장치가 게이트웨이 1번과 연결된 경우에는 토픽을 MQTT브로커 1번에 발행 또는 구독한다. IoT 장치의 이동에 의해 게이트웨이 2번과 연결되는 경우 IoT 장치는 게이트웨이 2번이 저장하고 있는 MQTT 브로커 2번의 IP 주소로 토픽을 발행 또는 구독한다. 이러한 단순한 동작방식을 통해 위치나 환경에 따라 사용하는 MQTT 브로커를 자동으로 선택하여 단일 MQTT 브로커 대비 위험부담을 줄이고 새로운 응용 기술에 활용 가능하다.

RPL 기반으로 분산 MQTT 브로커 구조를 설계하기 위해 RPL 장치가 자식 노드로서 6LBR에 참여를 요청하는 DAO 패킷을 활용한다. DAO 패킷의 구조는 Fig. 3와 같다. DAO 패킷을 통해 RPL 장치가 분산 MQTT 브로커에 토픽을 발행 또는 구독할지 결정하기 위해 8비트의 reserved 영역 중 1비트를 분산 브로커 선택(Distributed Broker Selection, DBS) 비트로 활용한다. DBS 비트 값이 1인 경우 6LBR이 저장하는 분산 MQTT 브로커로 토픽을 발행 또는 구독하고, DBS 비트 값이 0인 경우 장치가 저장하는 단일 MQTT 브로커로 토픽을 발행 또는 구독한다. MQTT의 reserved 값은 0이기 때문에 제안하는 분산 MQTT 브로커 구조가 구현되지 않은 RPL 장치는 기존 방식대로 호환성 문제없이 단일 MQTT 브로커를 사용할 수 있다.

Fig. 4는 RPL 기반 분산 MQTT 브로커 동작과정을 보여준다. RPL 장치는 DIS 패킷을 이용하여 6LBR에 DIO 패킷을 요청하거나 6LBR에서 주기적으로 전달하는 DIO 패킷을 수신하여 이웃 노드의 정보를 업데이트 한다. 이웃 노드 탐색을 마친 RPL 장치는 DAO 패킷을 통해 루트 노드까지의 경로를 확보한다. 이때 DAO 패킷의 DBS 비트 값에 따라 6LBR은 이후 MQTT 메시지 수신했을 때 전달할 MQTT 브로커를 결정하다. 만약 DAO 패킷의 DBS 값이 1인 경우 DAO 패킷을 보낸 RPL 장치의 IP 주소를 6LBR 테이블에 저장한다. 6LBR은 MQTT 메시지를 수신할 때마다 테이블에 저장된 IP 주소와 MQTT 메시지의 소스 IP 주소를 비교하여 분산 MQTT 브로커로 전달할지 말지를 판단한다. DAO 패킷의 DBS 값

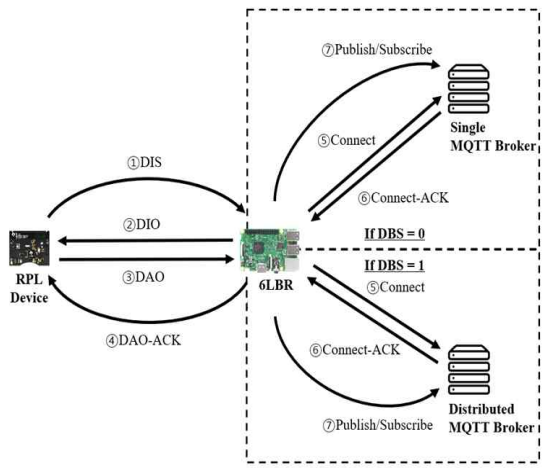


Fig. 4. Distributed MQTT broker's architecture and process.

이 0인 경우에는 기존 MQTT 동작 방식과 동일하게 동작한다. DAO 패킷을 정상적으로 수신하게 되면 6LBR은 DAO-ACK 패킷을 RPL 장치에 전송하고 MQTT 브로커와 연결하기 위한 연결 과정을 수행한다. 이러한 모든 과정이 끝나면 RPL 장치는 MQTT 발행 또는 구독 가능한 상태가 된다.

4. 성능 평가

본 장에서는 RPL 기반 분산 MQTT 브로커 구조에 대한 실험을 통한 성능평가 결과를 소개한다.

4.1 실험 환경 설정 및 실험 결과

실험은 Contiki 3.0[7]을 OS로 사용하였고 TI 사의 CC2650 Sensortag[5] 1.3을 노드로 사용하였다. Raspberry PI 2.0[6]을 6LBR로 사용하여 6LowPAN WSN을 구축하였다. 그리고 브로커는 노트북에 mosquitto 브로커를 설치하였으며 6LBR과 같은 인터넷망에 배치하였다. Fig. 5와 같이 환경 설정을 완료하였으며 총 두 개의 실험을 진행하였다.

첫 번째 실험에서는 서로 다른 브로커로의 패킷 크기별 전송시간을 비교한 것이다. 총 5개의 브로커를 배치 및 실험 하였다. 실험 환경 구축을 통해 설치한 실험실과 같은 네트워크에 있는 노트북의 Mosquitto 브로커와 비교 대상인 MQTT 브로커는 IBM 사에서 제공하는 브로커인 IBM Cloud Quickstart[15]를 사용하였다. 또한 추가적으로 실험실과 동일한 Mosquitto 브로커를 학교 내부의 다른 건물에 설치하였다. 그리고 학교 외부에 두 대를 추가로 설치하였다. 전송하는 패킷의 크기는 50,250,450bytes 세가지 경우로 실험하였다. 연구실 내의 같은 Network를 사용하는 윈도우 PC에서 tracert 명령어로 라우터의 수를 확인하였다. IBM Cloud Quickstart의 주소를 추적한 결과 25개의 라우터를 거쳐서 브로커까지 도달하는 것을 확인하였다. 마찬가지로 Mosquiito 브로커는 1개, 즉 AP를 거쳐서 바로 도달하게 되는 것도 확인하였다. 학교 내부 다른 건물의 브로커로는 4개의 라우터를 거치는 것을 확인하였다. 또한 학교 외부의 브로커로는 각각 9개, 11개의 라우터를 거쳐서 도달하는 것도 확인하였다. 실험 결과는 Fig. 6과 같다. 이러한 결과를 통해 두 가지의 사실을 알 수 있다. 먼저 패킷의 크기가 클수록 전송 시간이 더욱 소비되는 것이다. MQTT 프로토콜은 가벼운 패킷이기 때문에 크기에 제한이 있다. 그렇기 때문에 큰 데이터의 경우에는 여러번 나누어서 보내게 된다. 그리고 거치는 라우터의 수가 많아질수록 브로커로의 전송 시간이 증가하는 것을 확인할 수 있다. 또한 IBM 브로커로의 라우팅 시간이 가장 많이 걸리는 것을 확인 할 수 있다. 이러한 결과가 나오는 이유는 브로커로 가는 Network상의 라우터의 수가 많아질수록 시간이 더 소비되기 때문이다. 그리고 해외로의 라우팅은 국내에서의 라우팅에 비해 시간이 더욱더 소비되는 것을 확인 할 수 있다. 해외로의 라우팅은 매우 긴 해저 케이블을 통해 통신을 한다. 그리고 사진 및 동영상 파일 등의 사이즈가 큰 데이터와 시간과 관련된 데이터는 우선순위가 높게 설정되어 있다. 따라서 이외의 패킷들은 우선순위가 낮기 때문에 해외로의 라우팅은 지연 시간이 국내보다 더 걸리게 된다. 이러한 결과들을 통하여 패킷의 크기가 클수록, MQTT 브로커와의 거리가 멀수록 전송시간이

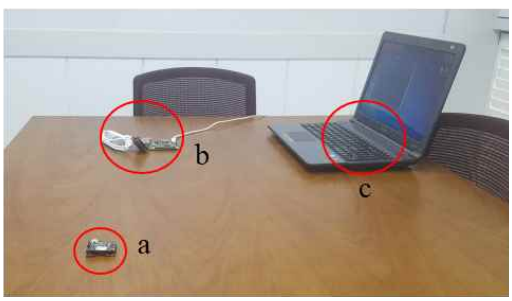


Fig. 5. Experiment Environment (a) RPL Node, (b) 6LBR, (c) Mosquitto MQTT Broker.

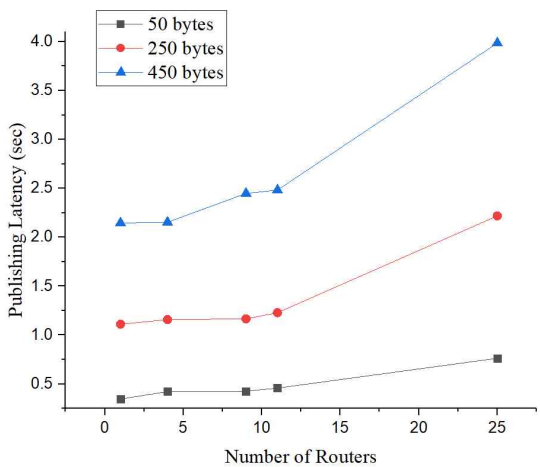


Fig. 6. MQTT Packet Size vs. Publishing Latency.

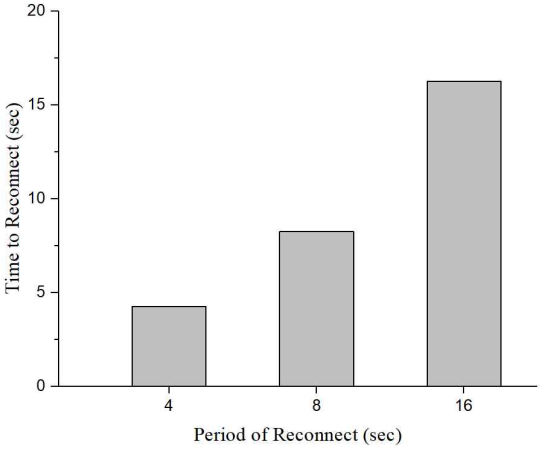


Fig. 7. The Number of Reconnection vs. Reconnection Latency.

느려지는 것을 확인할 수 있다. 이러한 특징들을 분산형 MQTT에 적용한다면 서비스의 질 향상이 이루어 질 수 있다.

기존의 MQTT 브로커 구조는 노드와의 연결이 끊어지면 통신할 대상이 없어져 통신을 할 수 없게 된다. 그러나 제안하는 분산형 구조는 기존 MQTT 브로커와 연결이 끊어진 후 다른 MQTT 브로커에게 연결이 가능하다. 두 번째 실험은 분산형 MQTT 브로커의 재연결 규칙의 변화에 따른 연결시간을 측정한 것이다. 재연결의 시도횟수가 한번일 때 연결요청 주기를 변화시켰다. 연결요청 주기 동안 브로커가 연결이 될 때까지 연결요청을 보내게 된다. 그리고 연결요청 시간이 끝나면 연결이 완료된다. 두 번째 실험의 결과는 Fig. 7과 같다. 주기가 짧을수록 짧은 시간 안에 재연결이 이루어진다. 재연결 주기는 개발자가 임의로 지정 가능한 값으로 주기가 짧은 경우 연결 지연시간이 짧은 반면 노드의 통신 부담은 증가한다. 반대로 재연결 주기가 긴 경우 재연결에 필요한 시간은 늘어나지만 노드의 통신 부담은 줄어든다. 빠른 시간에 재연결이 필요한 응용은 짧은 재연결 주기로 동작해야하고, 통신 부담을 고려해야하는 응용은 긴 재연결 주기로 동작해야한다. 실험 결과를 고려하여 개발자는 재연결에 필요한 시간을 결정할 수 있을 것으로 기대된다.

5. 결 론

본 논문은 MQTT 브로커의 구조적 문제에 대한 해결책을 제안한다. 기존의 MQTT는 하나의 브로커에 집중되어 트래픽 문제와 보안 문제들이 발생할 수 있다. MQTT의 문제점을 해결하기 위한 많은 연구들이 진행되었지만 여전히 문제점들은 남아있다. 본 논문에서는 이러한 문제를 해결하기 위해 RPL 기반 분산형 MQTT 브로커 구조를 제안한다. RPL 기반 분산형 브로커는 RPL 프로토콜의 DAO 패킷 내부 reserved 공간을 이용하여 프로토콜의 큰 수정 없이 분산형 MQTT 브로커 구조를 구현할 수 있으며 기존 RPL과 MQTT와 호환 가능한 기술이다. 게이트웨이가 DBS 비트를 수신하여 그 값에 따라 선택적으로 브로커와 연결할 수 있다. 실험을 통해 성능을 분석한 결과 발행 시간이 줄어드는 것을 확인할 수 있었고, 재전송 횟수와 시간 간격에 따라 재연결에 필요한 시간을 확인할 있었다. 분산형 MQTT 브로커 구조는 기존 단일 MQTT 브로커 구조에 비해 안정적이고 효율적으로 서비스 제공이 가능하고, 새로운 응용 서비스에 활용 가능할 것으로 기대된다.

REFERENCE

[1] OASIS, MQTT Version 3.1.1, 2014.

[2] M. Maritsch, C. Lesjak, and A. Aldrian, “Enabling Smart Maintenance Services: Broker-Based Equipment Status Data Acquisition and Backend Workflows,” *Proceeding of 2016 IEEE 14th International Conference on Industrial Informatics,* pp. 699-705, 2016.

[3] Mosquitto.conf Man Page, https://mosquitto.org/man/mosquitto-conf-5.html (accessed May, 28, 2018).

[4] Internet Engineering Task Force, *RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks*, 2012.

[5] Simplelink SensorTag, http://www.ti.com/ww/en/wireless_connectivity/sensortag (accessed May, 28, 2018).

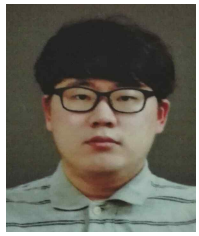
[6] Raspberry Pi 2 Model B, https://www. raspberrypi.org/products/raspberry-pi-2-model-b (accessed May, 28, 2018).

[7] Contiki: The Open Source Operating System for the Internet of Things, www.contiki-os.org (accessed May, 28, 2018).

[8] Eclipse Mosquitto, https://mosquitto.org (accessed May, 28, 2018).
[9] MQTT Used by Facebook Messenger, https://mqtt.org/2011/08/mqtt-used-by-facebook-messenger (accessed May, 28, 2018).
[10] Communicate with Your IoT Hub Using the MQTT Protocol, https://docs.microsoft. com/en-us/azure/iot-hub/iot-hub-mqtt-support (accessed May, 28, 2018).
[11] What is MQTT, http://ngins.blogspot. kr/2015/02/mqtt.html (accessed May, 28, 2018).
[12] Eclipse Paho-MQTT and MQTT-SN Software, https://www.eclipse.org/paho/ (accessed May, 28, 2018).
[13] HiveMQ-Enterprise MQTT Broker, https://www.hivemq.com/hivemq/ (accessed May, 28, 2018).
[14] M. Singh, M.A. Rajan, V.L. Shivraj, and P. Balamuralidhar, "Secure MQTT for Internet of Things (IoT)," *Proceeding of 2015 Fifth International Conference on Communication Systems and Network Technologies*, pp. 746-751, 2015.
[15] IBM's Quickstart, https://quickstart.internetofthings.ibmcloud.com/#/ (accessed May, 28, 2018).
[16] D. Kim, S. Kim, and S. Kwon, "Real-Time Transmission System for Greenhouse Information Using MQTT and RTS," *Journal of Korea Multimedia Society*, Vol. 18, No. 8, pp. 935-942, 2015.

안 현 성

2017년 2월 충북대학교 전자공학부 졸업
2017년 3월~현재 충북대학교 전자공학전공 석사
관심분야 : 센서네트워크, 스마트그리드



사 우 진

2012년 3월~현재 충북대학교 전자공학부 학사
관심분야 : 임베디드 시스템, 무선네트워크

김 승 구

2007년 2월 고려대학교 전기전자전파공학부 공학사
2010년 2월 고려대학교 전자컴퓨터공학과 공학석사
2013년 8월 고려대학교 전자컴퓨터공학과 공학박사
2013년 9월~2015년 8월 삼성전자 소프트웨어센터 책임연구원
2015년 9월~현재 충북대학교 전자공학부 조교수
관심분야 : WSN, WBAN, VANET, Bluetooth