논문 19-44-07-13 The Journal of Korean Institute of Communications and Information Sciences '19-07 Vol.44 No.07
https://doi.org/10.7840/kics.2019.44.7.1318

MQTT 프로토콜 기반 IoT 환경 및 솔루션의 효율성 향상을 위한 연구

이 근 혁◆, 김 동 휘*, 전 철 호*, 전 현 식*, 박 현 주°

A Study for Improving Efficiency of IoT Environment and Solution Based on MQTT Protocol

Geunhyeok Lee◆, Donghwi Kim*, Cheolho Jeon*, Hyunsig Jeon*, Hyunju Park°

요 약

IoT 기술의 발전이 가속화됨에 따라 IoT 연결 기기의 대수는 급격히 증가하였고 IoT 플랫폼의 활용 분야도 다양해졌다. 자연스럽게 각각의 분야에서 새로운 요구사항이 생겨났으며, 그 결과 IoT가 적응해야 할 환경도 다양해졌다. 하지만 클라이언트의 요구사항을 충족시키기 위한 기술은 미흡하다는 의견들이 대두하고 있으며, 현재의 IoT 프로토콜을 사용하는 다양한 플랫폼은 프로토콜을 단순하게 사용함으로써 효율성의 극대화를 끌어내지 못하고 있다. 그 결과 IoT 환경을 위해 설계된 프로토콜과 플랫폼은 효율성 문제로 논란이 야기되었다. 그중 IoT 프로토콜로 주목받고 있는 MQTT는 경량의 메시지 프로토콜로 배터리 소모 측면에서 효율적이라는 검증을 받았지만, 이는 디바이스가 통신이 필요하지 않은 시간에도 서버와 연결을 유지해야 한다는 등의 단점이 주목받고 있다. 이에 본 논문에서는 이와 같은 문제들을 해결하고자 일반적으로 사용되는 MQTT 기반 플랫폼의 문제점을 분석하고 MQTT 프로토콜 사용의 효율성을 향상하기 위한 MQTT Device Polling, Broker Persistence, 동적 토픽 생성 및 동적 Subscribe 솔루션을 제안한다.

Key Words : IoT, MQTT, MQTT Deivce Polling, Dynamic Topic, Broker Persistence

ABSTRACT

As the development of IoT accelerated, the number of IoT connected devices has increased sharply, and the application areas of the IoT platform have also diversified. Naturally, new requirements have emerged in each field, and as a result, the environment in which IoT has to adapt has also diversified. However, there are opinions that the technology to satisfy the client's requirements is insufficient, and various platforms using the current IoT protocol can not maximize the efficiency by simply using the protocol. As a result, protocols and platforms designed for the IoT environment have been controversial because of efficiency issues. Among them, MQTT, which is popular as IoT protocol, has been proved to be efficient in terms of battery consumption due to its lightweight message protocol. However, it is pointed out that the device has to maintain connection with the server even when communication is not needed.In this paper, we analyze the problems of MQTT based platform which is commonly used to solve these problems and suggest three solutions including MQTT Device Polling, Broker Persistence, Daymic Topic Generation and Dynamic Subscribe to improve the efficiency of using MQTT protocol.

◆ First Author : Hanbat National University Department of Mobile Convergence Enginerring, cheesam31@gmail.com, 학생회원
° Corresponding Author : Hanbat National University Department of Information and Communication Enginerring, phj@hanbat.ac.kr, 정회원
* Hanbat National University, Ubiquitous Media Technology
논문번호 : 201811-371-D-RN, Received November 27, 2018; Revised March 18, 2019; Accepted May 15, 2019

Ⅰ. 서 론

최신 ICT 동향에서 빠짐없이 등장하는 IoT는 사물과 사물, 사람과 사물 간의 정보 교환을 통해 인간의 삶에 밀접한 관계를 맺고 있으며, 이에 관한 연구와 개발이 활발해졌고 국내외 대기업뿐만 아니라 스타트업, 중소기업, 공기업 및 다양한 기관에서 IoT 발전에 이바지하면서 IoT 서비스 및 애플리케이션은 다양한 콘텐츠를 제공하고 있다[1]. 현재 IoT의 분야는 건설, 의료, 교통, 제조, 의류 등 다양한 곳에서 영향력을 끼치고 있으며, 여러 분야에서 다양한 IoT 디바이스가 등장했다.

IT 조사 기관과 전문가의 의견에 따르면 2020년엔 200억대 이상의 IoT 디바이스가 전 세계 곳곳에 연결될 것이라 예상했고[2], 그로 발생하는 트래픽 양도 급격히 증가할 것이라 예상했다. 대표적인 예로 현재 모바일 분야에서는 5G 표준화에 박차를 가하며 2018년에 상용화를 목표로 하고 있다. 이러한 이유는 급격히 증가하는 디바이스와 이들이 뿜어내는 트래픽 양 등을 수용하기 위함이며, 초고속 데이터 전송, 초저지연, 초대용량을 목표로 하고 있다[3].

이러한 발전은 IoT 서비스 플랫폼 및 프로토콜을 등장시켰다. 아래 그림 1.은 2018년에 Eclipse 재단에서 IoT 개발자를 대상으로 한 설문조사이다. 2018년 기준으로 1위는 MQTT로 62.61%, 2위는 HTTP 54.1% 3위는 Websockets 34.95%로 집계되었으며, IoT 개발자들이 가장 많이 사용하고 있는 프로토콜은 MQTT라는 것을 확인할 수 있다[4].

하지만 IoT 환경에서 가장 많이 사용되며 주목받고 있는 MQTT는 IoT 환경에서 사용하기에 적합하도록 설계되었음에도, 장시간 사용하지 않는 디바이스가 서버와 연결을 유지하면서 발생하는 자원 낭비가 비효율적일 것이라는 의견이 대두했다. 이는 급격히 증가하는 디바이스만큼 연결 유지에 소모되는 자원과 통신에서 발생하는 트래픽 양이 급격히 증가하는 문제가 됐다.

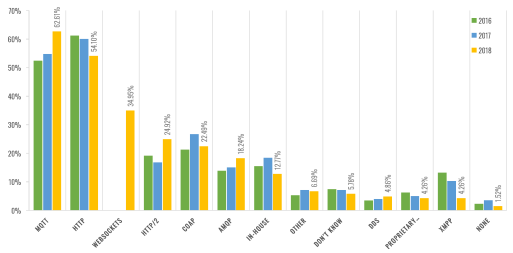

그림 1. IoT 개발자들의 프로토콜 사용 설문조사 결과
Fig. 1. IoT developers' protocol usage survey results

이러한 문제점을 견주어 볼 때 IoT 기술은 더욱 향상되어야 하며 이를 위해서는 효율적인 프로토콜의 사용이 기본 되어야 한다. 이에 본 논문에서는 MQTT 기반 IoT 환경 및 솔루션의 효율성을 향상하기 위한 세 가지 솔루션을 제안했으며, 이를 통해 IoT 환경에서 발생할 수 있는 문제점을 개선하고 다양한 분야에서 발생하는 클라이언트의 요구사항을 충족시킴으로써 신뢰성, 편의성, 확장성, 효율성 등을 증대시킬 수 있을 것이라 예상한다.

Ⅱ. 관련 연구

MQTT란 Message Queuing Telemetry Transport의 약자로, 제한된 네트워크 및 다중 플랫폼 환경에서 성능이 제한된 노드들이 경량의 메시지로 통신할 수 있도록 설계된 프로토콜이다. 다른 프로토콜에 비해 작은 고정 헤더와 오버헤드의 최소화로 제한된 환경에서 사용하기 적합한 프로토콜이다[5]. 1999년에 IBM과 파트너사인 Eurotech에서 개발됐고, 2013년에 국제 표준화 단체인 OASIS(Organization for the Advancement of Structured Information Standards)는 사물 인터넷을 위한 메시징 프로토콜로 MQTT를 선정했다. 또한, HTTP가 웹을 통해 정보를 공유할 수 있는 기반을 만들었던 것과 같이 MQTT가 사물인터넷 환경의 디바이스 정보를 공유할 수 있는 기반이 될 것이라 평가받았다[7].

MQTT는 발행자, 구독자, 중개자라는 세 가지 개체를 이용해 양방향 발행/구독 통신으로 메시지를 송수신할 수 있다. 메시지를 송수신하는 행위를 발행과 구독이라 부르며, 이러한 행위를 하는 개체를 발행자, 구독자라 지칭한다. 중개자는 발행자와 구독자 사이에서 중계자 역할을 하며, 발행자와 구독자 모두 클라이언트로 간주한다. 발행자와 구독자는 Topic이라는 문자열을 통해 메시지를 송수신한다. Topic은 슬래쉬(/)를 통해 계층구조를 나타낼 수 있다[6]. 또한 다양한 영역을 데이터를 수집하기 위해 와일드카드(Wildcard)인 '+'와 '#'을 제공한다. '+'는 같은 계층의 모든 Topic을 수용할 수 있음을 말하고, '#'은 같은 계층을 포함한 하위 계층의 모든 Topic을 수용할 수 있음을 말한다.

메시지 송수신에 대한 신뢰를 위해 MQTT는 QoS(Quality of Service) 옵션을 제공한다. 기본적으로 QoS 레벨은 3 단계로 나뉘며 레벨 0은 최대 한 번, 레벨 1은 최소 한 번, 레벨 2는 정확히 한 번을 의미

한다. 또한, 효율적인 통신을 위해 'Keep Alive', 'Clean Session', 'Client Identifier'와 같은 다양한 옵션들이 사용된다. 'Keep Alive'는 클라이언트가 하나의 패킷 전송을 완료하는 시점과 그다음 패킷 전송을 시작하는 시점 사이에 경과하는 것이 허용되는 최대 시간 간격을 의미한다. 다른 패킷 전송의 부재가 발생하는 경우 Keep Alive 시간을 갱신시키기 위해 클라이언트는 Pingreq 패킷을 송신해야 한다. 만일 Keep Alive 시간이 만료되면 서버와 클라이언트는 연결이 해제된다. 'Clean Session' 은 세션 상태의 처리를 지정한다. 클라이언트와 서버는 네트워크 연결 흐름을 통해 지속해서 신뢰할 수 있는 메시지를 사용할 수 있도록 세션 상태를 저장할 수 있다. Clean Session은 기본이 true이며, 클라이언트와 연결이 해지될 때 메모리에서 클라이언트 정보를 삭제한다. 'Client Identifier'는 중개자에서 클라이언트를 식별할 때 사용되는 속성이다. Client Identifier는 고유한 값이며, 세션과 연관된 정보를 식별하기 위해서 사용된다.

Ⅲ. 제안하는 솔루션

본 논문에서 제안하는 MQTT 프로토콜 기반의 IoT 환경 및 솔루션의 효율성 향상을 위한 연구는 그림 2와 같이 크게 세 가지의 솔루션을 제안한다. MQTT 프로토콜 기반 디바이스의 배터리 소모 효율 향상과 트래픽 양을 줄이기 위한 MQTT 디바이스 Polling, 서버와 디바이스 정보를 영구적으로 저장하며 데이터의 안정성과 신뢰성을 향상하기 위한 중개자 Persistence, 불필요한 구독을 방지하며 사용자의 요청에 따라 동적으로 구독 함 으로써 서버의 확장성 및 편의성 향상을 위한 동적 토픽 생성 및 동적 구독으로 구성되며, 설계 및 구현 방법은 하위 절에서 중점적으로 기술했다.

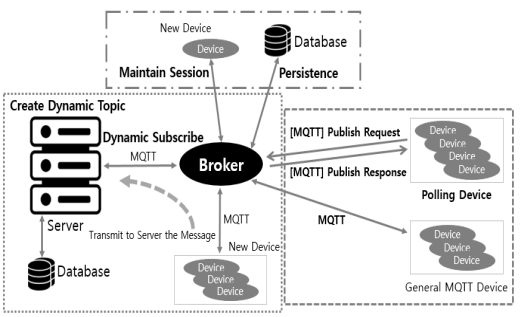

그림 2. 제안하는 솔루션 아키텍처
Fig. 2. Proposed solution architecture

3.1 MQTT 디바이스 Polling

MQTT는 연결 시 초기 비용이 많이 들기 때문에 연결 해제를 하는 것이 아닌 Pingreq와 Pingresp 패킷을 송수신함으로써 중개인과 연결을 유지한다. 이러한 이유로 일반적인 MQTT 기반의 디바이스는 서버로부터 메시지를 수신하기 위해 중개인과 연결을 유지하며 대기한다. 만일 서버가 해당 디바이스와 특정 통신을 하지 않을 때 디바이스는 중개인과 연결을 유지하며 불필요한 통신이 행해진다. 이러한 문제를 해결하기 위해서 MQTT 기반의 디바이스가 특정 목적의 작업을 수행해야 할 때만 서버와 연결하도록 설계함으로써, 서버에게 데이터 조회 요청을 보내고 그에 대한 응답을 받을 수 있도록 구현했다. 또한, 초기 연결 시 통신량을 줄이기 위해 디바이스와 중개인 연결 시 클라이언트 세션을 삭제하는 옵션인 Clean Session 플래그를 false로 변경하고 Client Identifier를 설정했다. 결과적으로 중개인가 클라이언트의 세션 정보를 유지하고 있을 때, 디바이스는 첫 연결 시의 구독를 제외한 나머지 연결에서 구독를 생략할 수 있고 기존의 MQTT 통신에서의 연결 유지에 대한 자원 낭비를 막을 수 있다.

MQTT 디바이스 Polling의 전체적인 흐름은 다음과 같다. 디바이스와 서버가 중개인에게 Connect 패킷을 송신하여 연결을 요청하고 특정 토픽으로 구독한다. 그 후 디바이스는 클라이언트로부터 수신된 명령이 있는지에 대한 명령 조회 요청으로 발행 한다. 중개인은 디바이스로부터 온 발행을 서버에게 전달한다. 이를 수신한 서버는 로컬 데이터베이스에서 디바이스와 관련된 데이터를 조회한다. 서버는 조회에 관한 결과를 특정 토픽으로 중개인에게 발행 하며, 중개인은 이를 해당 디바이스에게 전달한다. 최종적으로

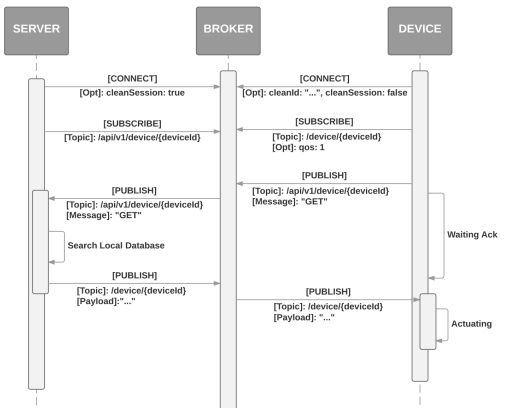

그림 3. MQTT 디바이스 Polling 기본 흐름
Fig. 3. MQTT 디바이스 Polling basic flow

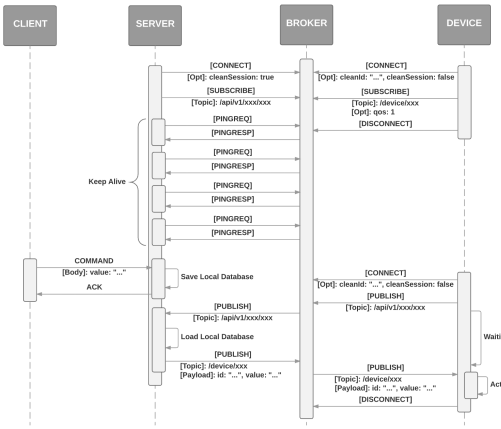

그림 4. MQTT 디바이스 Polling의 통신 흐름
Fig. 4. MQTT 디바이스 Polling communication flow

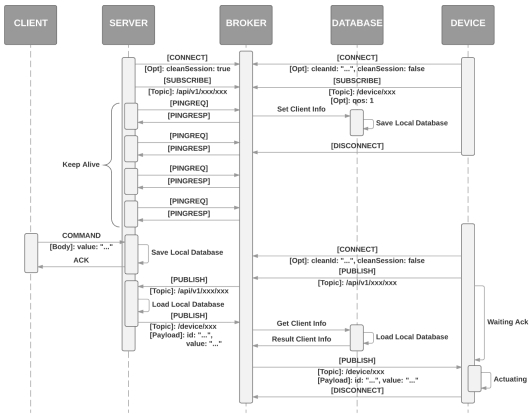

그림 5. Persistence를 이용한 MQTT 디바이스 Polling 흐름
Fig. 5. MQTT 디바이스 Polling Flow Using Persistence

데이터를 받은 디바이스는 그에 맞는 엑추에이팅을 한다. 1회의 MQTT 디바이스 Polling 통신은 이처럼 표현되며, 그림 3 과 같은 흐름을 가진다. 간단하게 디바이스가 서버에게 데이터 요청을 하고 그에 적합한 응답을 받는 형식이 주된 목적이다.

제안하는 솔루션의 MQTT 디바이스 Polling은 디바이스가 주기적으로 서버에 연결하여 명령 조회를 하는 로직으로 아래 그림 4와 같이 표현된다.

그림 4의 디바이스는 특정 주기에 맞춰 서버에 연결해서 자신에게 온 데이터가 있는지 확인한 후 중개인과 연결을 해제는 작업을 반복한다. HTTP를 사용하는 디바이스가 서버에게 GET을 통해 데이터를 요청하는 흐름과 비슷하다. 이때 디바이스는 Client Identifier와 Clean Session 옵션을 사용한다. Clean Session 옵션을 false로 설정하여 중개인에서 디바이스와 연결이 끊겨도 세션 정보를 유지하고, Client Identifier를 통해 클라이언트를 식별하여 재 연결 시 세션 정보를 불러올 수 있다. 이를 이용하여 첫 연결 시 한 번만 구독하며, 그 이후 연결부터는 다시 구독하지 않아도 된다. 그 결과로 MQTT를 사용하는 디바이스는 절전 상태 또는 저 전력 상태로 진입할 수 있는 여건을 제공하며, 서버와의 연결 유지비용을 절약하고 트래픽 양을 줄일 수 있다.

3.2 중개자 Persistence

일반적으로 중개인은 서버와 디바이스를 모두 클라이언트로 간주하며, 클라이언트의 정보를 메모리에 저장한다. 메모리에 저장된 정보는 중개인이 비정상적으로 종료될 경우 유실된다. 클라이언트 정보 유실은 데이터 안정성 문제와 서버의 신뢰성 문제를 초래한다. 이러한 문제를 해결하기 위해 mosca라는 중개인을 이용하며, 이 중개인에서 제공하는 기능인 Persistence 기능을 이용한다. mosca도 클라이언트의 정보를 일반적으로 메모리에 저장하지만, Persistence 기능을 사용하여 데이터베이스에 클라이언트 정보를 저장할 수 있다. 따라서 중개인가 비정상적으로 종료되어도 클라이언트의 정보는 유실되지 않으며, 데이터에 대한 안정성과 신뢰성을 지킬 수 있다.

그림 5는 3.1절에서 제안한 솔루션에 mosca의 Persistence 기능을 추가한 sequence chart 이다. mosca의 Persistence를 통해 디바이스의 세션 정보는 메모리가 아닌 데이터베이스에 저장된다. 따라서 세션 정보가 중개인의 데이터베이스에 저장된 경우 디바이스는 매 연결 시 구독을 하지 않아도 되며, 비정상적인 종료 후에도 데이터는 안전하게 유지된다.

3.3 동적 토픽 생성과 동적 구독

일반적으로 서버는 제한적으로 특정 토픽만 수용할 수 있다. 이는 정적이며 추가적인 토픽에 대해 구독해야 하는 경우 빠른 대응이 힘들고 서버가 상황에 따라

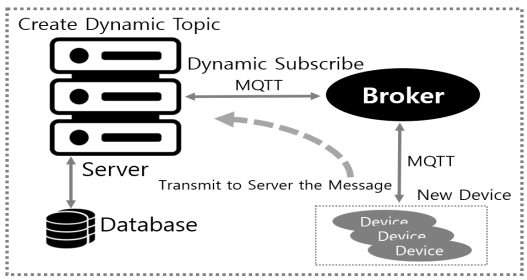

그림 6. 동적 토픽 생성과 동적 구독
Fig. 6. Dynamic Topic Generation and Dynamic 구독

불필요한 토픽도 구독해야 한다는 단점이 생긴다. 이에 클라이언트의 요청 시 서버가 동적으로 토픽을 생성하고 중개인에게 구독 할 수 있도록 로직을 설계했다.

그림 6은 서버가 동적으로 토픽을 생성하여 중개인에게 구독하고, 새로운 디바이스가 생성된 토픽으로 발행 했을 때 서버의 데이터베이스에 저장하는 과정을 도식화한 그림이다.

3.3.1 토픽 설계

동적으로 토픽을 만들기 위해서는 서버에게 토픽을 만들기 위한 요청의 토픽인지, 와일드카드를 사용하는지, 와일드카드를 사용한다면 어디에 위치하는지 등의 특정 규칙과 어떤 옵션을 사용하는지에 대한 정보 등이 필요하다. 제안하는 솔루션에서는 이러한 정보를 담을 수 있도록 특정 규칙을 정의했고, 클라이언트가 서버에게 토픽 생성 요청을 보내기 위한 메시지 포맷은 그림 7과 같다.

그림 7에서의 "/notification"은 동적 토픽 생성에 대한 클라이언트의 요청에 기본적으로 사용되는 문자열이다. 하위 토픽으론 사용할 토픽을 선언하거나 "`D" 문자열을 사용한다. "`D" 문자열은 사용되는 와일드카드의 위치를 표시하기 위한 문자열이며, 해당 자리에 '+'문자로 교체된다. '#'와일드카드를 사용할 경우 토픽 끝에서만 사용되기 때문에 "`D" 문자열을 사용하지 않고, "/notification" 문자열 뒤로 만들고자 하는 토픽을 기술한다.

결과적으로 그림 8과 같은 형식으로 클라이언트가 서버에게 발행 하면 서버는 토픽과 메시지를 분석하여 동적으로 토픽을 생성하고 중개인에서 구독 한다.

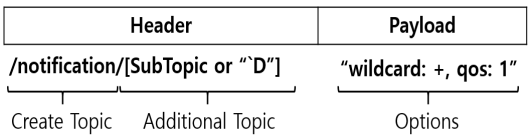

그림 7. 클라이언트 토픽 생성 요청 포맷
Fig. 7. Client Topic Generation Request Format

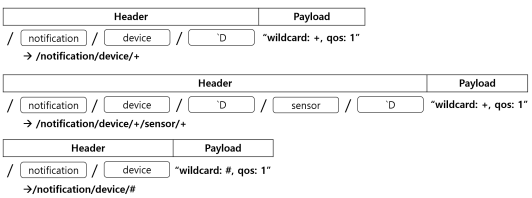

그림 8. 동적 토픽 생성 시 포맷 예시
Fig. 8. Example format for dynamic topic creation

3.3.2 Scenario

그림 9는 동적 토픽 생성을 요청하는 클라이언트가 MQTT 프로토콜을 사용한다는 가정 하에 만들어진 sequence chart 이다. 전체적인 흐름을 보면 서버는 중개인에게 "/notification/#"으로 구독 한다. 다음 클라이언트가 "/notification/xxx/xxx/xx"로 발행 하면 중개인은 서버에게 전달한다. 메시지를 전달받은 서버는 동적으로 토픽을 만들고 중개인에게 구독 한다. 그 후 디바이스가 해당 토픽으로 중개인에게 발행 하면 서버는 중개인을 통해 메시지를 수신할 수 있다. 서버는 로컬 데이터베이스가 있으면 디바이스로부터 온 정보를 저장한다.

클라이언트가 HTTP를 사용할 경우 그림 10과 같이 표현할 수 있다.

이 경우 서버 측에서 HTTP를 수용할 수 있는 프록시를 사용하여 HTTP인 데이터를 MQTT로 변환한 후 MQTT 서버로 전달한다. 일반적으로 클라이언트가 사용하는 프로토콜은 HTTP임으로 이 프로토콜로 동적 토픽 생성에 대한 요청이 왔을 때 이를 수용할 수 있는 로직이 필요하다.

그림 10에 대한 흐름은 그림 11과 같다. HTTP의 경우 MQTT에서 사용하는 토픽을 API에 사용할 수 없기 때문에 동적 토픽 생성용 API를 만들고 MQTT에서 필요한 부분인 토픽, 와일드카드, QoS를 body에 담아서 전송한다.

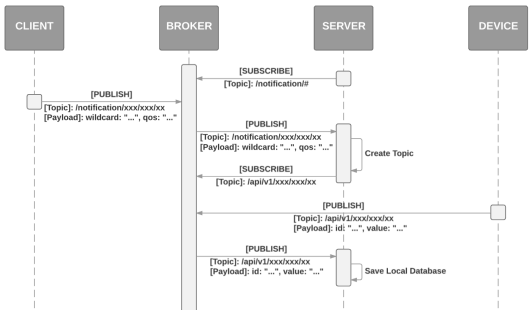

그림 9. 동적 토픽 생성 및 동적 구독 흐름
Fig. 9. Dynamic Topic Generation and Dynamic 구독 Flow

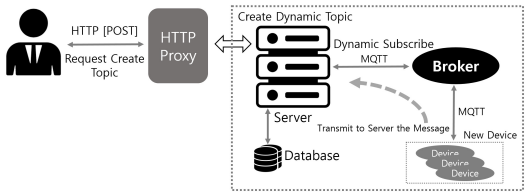

그림 10. HTTP를 이용한 동적 토픽 생성 및 동적 구독
Fig. 10. Dynamic Topic Generation and Dynamic subscribe Using HTTP

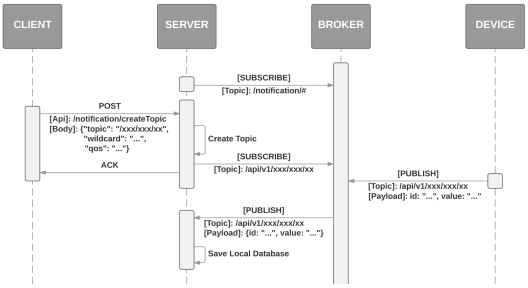

그림 11. HTTP를 이용한 동적 토픽 생성 및 동적 구독 흐름
Fig. 11. Dynamic Topic Generation and Dynamic 구독 Flow using HTTP

위와 같이 클라이언트의 요청에 따라 동적으로 토픽을 생성하고 구독함으로써 서버가 제한적으로 수용했던 특정 토픽 이외에도 다양한 토픽을 수용할 수 있도록 설계했다.

Ⅳ. 검증 및 평가

4.1 테스트 환경

본 논문에서 제안하는 솔루션의 효율성을 검증하기 위해 같은 시나리오로 제안하는 솔루션과 일반적으로 사용되는 MQTT 기반의 통신에서 발생하는 트래픽 양을 비교・분석한다. 그림 12는 검증 및 평가를 위한 테스트 환경을 요약한 그림이다.

그림에서 나타나듯이 MQTT 서버와 중개자는 같은 local 환경에서 구동되고 디바이스는 WiFi통신으로 중개인에 접근하는 형태로 구성되어 있다. 아래 표 1은 테스트 환경의 사양을 요약해 놓은 것이다.

디바이스는 센서, 엑추에이터를 연결하기 쉽고 아두이노 우노와 같은 성능을 가진 오렌지보드를 사용하였다. 중개자 및 MQTT서버는 패킷 분석 도구인 와이어샤크를 설치하기 위해 고성능 PC를 이용했다. 트래픽양 만을 확인하기 위해 고성능 PC를 이용했지만, 실제 중개자를 운용하기 위해 고성능 PC를 구매할 필요는 없다.

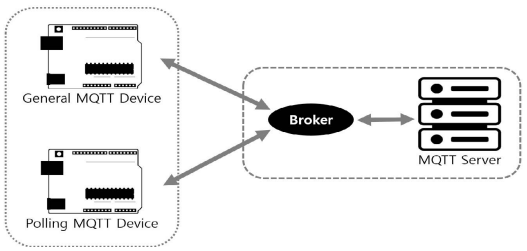

그림 12. 테스트 환경
Fig. 12. Test environment

표 1. 각 디바이스의 사양
Table 1. Specification of each device

Entity	Specification	
MQTT Server	CPU	Intel(R) Core(TM) i7-4770 CPU 3.40 GHz
	Memory	16 GB
	OS	Windows 10
	Platform	node 7.7.1
Broker	CPU	Intel(R) Core(TM) i7-4770 CPU 3.40 GHz
	Memory	16 GB
	OS	Windows 10
	Broker	mosca 2.3.0
Device	Controller	ATmega328p
	Flash Memory	32 KB
	SRAM	2 KB
	Clock Speed	16 MHz

4.2 검증 및 비교

본 절에서는 일반적으로 사용되는 MQTT 기반 디바이스와 제안하는 솔루션을 시나리오를 통해 트래픽 양을 비교하고 효율성을 검증한 후 표로 기술하였다. 테스트 시나리오는 디바이스가 중개자에게 CONNECT 메시지를 보내면 중개자가 CONNACK 메시지로 응답을 하여 연결이 성립된 후 사용자가 디바이스에게 간단한 on/off 명령을 주기적으로 송신하여 그에 따른 동작을 취하도록 구현하였다. 테스트 환경의 Keep Alive의 송수신은 default 값인 10s로 설정하였으며 QoS의 level은 1, 테스트 주기는 60s 이다.

일반적인 MQTT 통신의 경우 Keep Alive의 값을 10초로 설정하여 60초 동안 6번의 Pingreq와 Pingresp를 송수신함으로 1,065 bytes의 트래픽이 발생한다. 또한, MQTT는 초기 연결 비용이 많이 들기 때문에 일반적인 MQTT 통신의 초기 연결 비용, 구독 1회, 연결 유지 60초, 발행 1회를 수신할 경우 총 1,853 bytes의 트래픽이 발생한다. 반면에 제안하는 솔루션에서는 연결 유지비용이 소모되지 않는다. 대신 Connect 패킷 전송 시 클라이언트 아이디가 부여되며 Connect, 요청과 응답에 대한 발행, TCP 연결 해제 비용으로 총 1,085 bytes가 소모된다.

표 2는 실제 테스트 결과를 바탕으로 트래픽의 양을 기술한 것이다. 표 2의 합계를 비교했을 때 두 시나리오는 대략 1.8배 정도 차이가 난다. 외관상 큰 차

표 2. 60초 1회 통신에 대한 트래픽 양 비교
Table 2. Comparison of the amount of traffic for one 60-second communication

트래픽 종류	Typical MQTT communication	Suggested solutions
TCP connect	180	180
Connect (TCP)	172 (60)	196 (60)
Pingreq/Pingresp (TCP)	116 (60)*6	0
Publish (TCP)	135 (60)	135 (60)
Subscribe(TCP) (+Disconnect)	136 (54)	226 (54)
TCP disconnect	0	174
Total (Bytes)	1,853	1,085

이가 나지 않아 보이지만, 제안하는 솔루션에서 더 적은 트래픽이 발생함을 뜻한다. 이는 다수의 디바이스를 사용하여 트래픽을 비교해볼 때 더 명확해질 것이다.

그림 13은 표 2에서 제시한 1회 통신에서 60초를 기준으로 발행와 대기 시간을 증가시켰을 때의 합계에 대한 그래프이다. 차례로 20분까지 측정한 결과에서 일반적인 MQTT 디바이스의 경우 1분당 약 1,250 bytes씩 증가했으며, 제안하는 솔루션의 디바이스는 약 900 bytes씩 증가했다. 이 결과는 60초라는 시간 안에 한 번의 발행가 있을 경우의 차이에 해당한다. 위와 같은 환경으로 120초당 1회의 통신이 이루어졌을 경우 두 시나리오의 트래픽 양 차이는 크다. 제안하는 솔루션의 경우 특정 조건에만 서버와 연결하여 통신하기 때문에 시간적 영향을 받지 않지만, 일반적인 MQTT의 경우 서버와 연결을 유지하기 위한 비용이 소모되기 때문에 시간적인 영향을 받아 소모되는 트래픽 양이 급격히 증가한다.

그림 14는 120초에 한 번씩 명령을 받거나 요청을 하는 경우 발생하는 트래픽 양을 표현한 그래프이다.

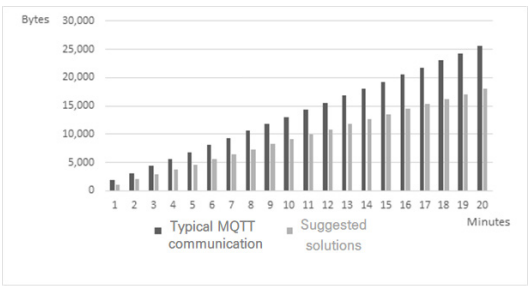

그림 13. 시간과 통신 횟수를 증가에 따른 트래픽 양 비교 (60초 기준)
Fig. 13. Comparing the amount of traffic with increasing time and communication frequency (in 60 seconds)

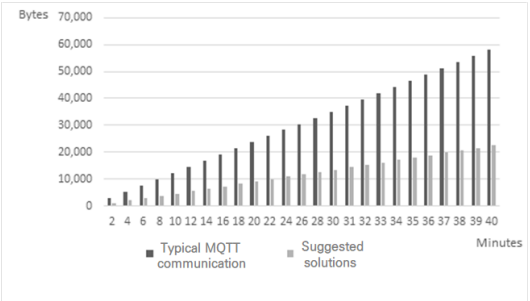

그림 14. 시간과 통신 횟수를 증가에 따른 트래픽 양 비교 (120초 기준)
Fig. 14. Comparing the amount of traffic according to the increase of the time and communication frequency (in 120 seconds)

앞의 그림 12와는 다르게 두 시나리오의 트래픽 양을 비교하였을 때 시간에 따라 급격히 차이가 나는 것을 확인할 수 있다. 일반적인 MQTT 통신과 제안하는 솔루션은 시간적 영향을 크게 받으며 그에 따라 효율성의 차이를 예측할 수 있다. 결과적으로 제안하는 솔루션의 경우 IoT의 환경에 따라 적합하게 적용해야 하며 제안하는 솔루션에 적합한 환경의 경우 효율성이 더욱 증대될 것이다.

V. 결 론

MQTT는 경량의 메시지 프로토콜로 전력 소모 효율에 우수한 평가를 받았음에도 불구하고, 여러 플랫폼에서 단순하게 사용되고 있어 문제가 발생한다. MQTT 특성상 중개자와 연결하고 특정 토픽으로 구독 하여 메시지가 수신될 때까지 연결을 유지하며 대기한다. 이 점에서 연결을 유지하기 위해 사용되는 통신으로 인해 불필요한 트래픽이 발생함으로 인해 트래픽 양, 배터리, 메모리 등의 자원이 낭비됐다. 이러한 비효율적인 자원 낭비 방지를 목표로 MQTT 특성상 낭비될 수밖에 없는 자원 소모에서 MQTT 디바이스 Polling과 중개자 Persistence를 제시함으로써 트래픽 양과 배터리 소모를 줄이고, 동적 토픽 생성 및 동적 구독을 제안하면서 MQTT를 사용하는 플랫폼의 편리성과 확장성을 높였다. 이를 통해 일반적인 MQTT 통신과 제안하는 솔루션을 동일 시나리오로 테스트한 결과는 60초의 경우 절대적으로 비교했을 때 약간의 트래픽 양을 줄일 수 있었으며, 120초의 경우 현저한 트래픽 양의 차이로 두 배 이상의 트래픽 양을 확인할 수 있었다. 이는 결과적으로 통신이 빈번하게 일어나지 않는 디바이스일수록 효율성이 더 증

대될 것으로 예상하며, IoT 환경과 솔루션에 따라 이 차이가 뚜렷해짐을 예상할 수 있다. 또한, 동적 토픽 생성을 통해 제한적인 토픽을 수용하는 서버에게 확장성과 편의성을 향상하며 다양한 토픽을 수용함으로써 IoT 발전에 가속할 수 있을 것이라 예상한다.

향후 연구로는 MQTT 초기 연결 시 소모되는 트래픽 양을 줄이고, 제안하는 솔루션을 플랫폼에 적용하여 성능 및 효율성 등의 비교, 분석을 진행한다. MQTT는 초기 연결 시 사용되는 트래픽 양이 다른 프로토콜에 비해 상당하다. 이때 소모되는 배터리 양도 비교적 크다. 이에 MQTT의 초기 연결을 줄이는 방안을 연구하여 제안하는 솔루션에 적용한다면 더 높은 효율성을 갖게 될 것이라 예상한다. 더불어 제안하는 MQTT 디바이스 Polling과 일반적인 MQTT 통신을 상황에 따라 선택적으로 사용할 수 있는 로직을 설계 및 구현함으로써 프로토콜 사용의 효율성을 극대화하기 위한 연구를 진행한다.

References

[1] J.-O. Seo, “Design and implementation of realtime things control system using MQTT and WebSocket in IoT environment,” *J. KIECS*, vol. 13, no. 3, pp. 517-524, Jun. 2018.

[2] Rob van der Meulen, *Gartner Says 8.4 Billion Connected “Things” Will Be in Use in 2017, Up 31 Percent From 2016*(2017), Retrieved Feb. 19, 2017, from https://www.gartner.com/en/newsroom/press-releases/2017-02-07-gartner-says-8-billion-connected-things-will-be-in-use-in-2017-up-31-percent-from-2016.

[3] H. Lee, *Concept and characteristics of 5G mobile communication systems*(2015), Retrieved Feb. 19, 2017, from http://www.netmanias.com/ko/post/blog/7049/5g/concept-and-characteristics-of-5g-mobile-communication-systems.

[4] Benjamin Cabé, *IoT Developer Survey 2018* (2018), Retrieved Apr. 30, 2018, from https://www.slideshare.net/kartben/iot-developer-survey-2018.

[5] Arnaud Giuliani, *MQTT : The Open Road to Internet of Things(2013)*, Retrieved Jan. 28, 2017, from https://www.ekito.fr/people/mqtt-the-open-road-to-internet-of-things/.

[6] Dave Locke, *MQTT: Enabling the Internet of Things(2013)*, Retrieved Jan. 30, 2017, from https://www.ibm.com/developerworks/community/blogs/c565c720-fe84-4f63-873f-607d87787327/entry/tc_overview?lang=en.

[7] S. Kim, D. Kim, H. Oh, H. Jeon, and H. Park, “The data collection solution based on MQTT for stable IoT platforms,” *J. KIICE*, vol. 20, no. 4, pp. 728-738, Apr. 2016.

이 근 혁 (Geunhyeok Lee)

2018년 2월 : 한밭대학교 정보통신공학과 졸업

2018년 3월~현재 : 한밭대학교 모바일융합공학과 석사과정

<관심분야> 사물인터넷, 임베디드, 인공지능

김 동 휘 (Donghwi Kim)

2015년 2월 : 한밭대학교 전파공학과 졸업

2017년 8월 : 한밭대학교 전파공학과 석사

2018년 5월~현재 : 지니뮤직 재직

<관심분야> 사물인터넷, 시스템프로그래밍, 데이터베이스

전 철 호 (Cheolho Jeon)

2018년 2월 : 한밭대학교 정보통신공학과 졸업

2018년 3월~현재 : 한밭대학교 모바일융합공학과 석사과정

<관심분야> 웹 프로그램밍, 데이터베이스, 백엔드

전 현 식 (Hyunsig Jeon)

2005년 2월 : 한밭대학교 전파공학과 석사

2011년 2월 : 한밭대학교 전파공학과 박사

<관심분야> 데이터베이스, 운영체제, 실내 위치 추정 알고리즘

박 현 주 (Hyunju Park)

1990년 2월 : 서울시립대학교 전산통계학과

1992년 2월 : 서울대학교 전산학과 석사

1998년 4월~2003년 3월 : 대전산업대학교 정보통신공학과 전임강사

2004년 4월~현재 : 국립 한밭대학교 정보통신학과 교수

<관심분야> 프로그래밍 언어, 운영체제, 데이터베이스