([http://www.ciokorea.com/news/26305?page=0,0) 뉴스 & 블로그 3](http://www.ciokorea.com/news/26305?page=0,0)%20뉴스%20&%20블로그%203)개 정도 참고

최근 IoT 기기들에 쓰이는 주요 실시간 프로토콜들은 XMPP, CoAP, MQTT의 3가지.

-XMPP

XMPP(eXtensible Messaging and Presence Protocol)는 2인 이상의 참여자 간에 구조적 데이터를 거의 실시간에 가깝게 교환할 수 있게 해주는 XML 기반 TCP 커뮤니케이션 프로토콜이다.  
  
XMPP의 기능들 중에는 대화 참여자 정보를 보여주는 것과 연락처 리스트 관리 기능이 있다. 두 기능 모두 인스턴트 메시징을 위해 제작된 것이었지만 IoT에도 적용할 수 있다.  
  
XML 재단과 XMPP자체의 열려 있는 성질 때문에 XMPP는 퍼블리TL/섭스크라이브 (publish/subscribe) 시스템에도 사용되고 있다. XMPP가 IoT 애플리케이션과 잘 맞는 이유다.  
  
장점:

분산화된 성격. XMPP의 작동 방식은 이메일과 비슷하다. CoAP나 MQTT처럼 하나의 중앙 서버나 브로커에 의지하기보다는 여러 트랜스퍼 에이전트(transfer agents)에 걸쳐 작동하는 것이 특징이다.  
  
이메일과 마찬가지로 누구나 자신의 XMPP 서버를 운영할 수 있어 기기 제조업체나 API 오퍼레이터가 자신만의 기기 네트워크를 형성하고 관리할 수 있다. 또한 누구나 서버를 운영할 수 있기 때문에 보안이 요구되는 경우 내장 TLS 암호화를 사용해 회사 인트라넷에서 안전한 인증 프로토콜을 이용해 고립시킬 수 있다.

단점:

1) 엔드-투-엔드 암호화가 안 된다는 것. 물론 암호화가 굳이 필요하지 않은 경우도 있지만, 대부분 IoT 기기들은 암호화를 필요로 한다.   
  
2) QoS(Quality of Service)의 부재. 메시지가 제대로 전달되었는지 확실히 하지 못함.

-CoAP

CoAP(Constrained Application Protocol)는 간단한 전자 기기들의 인터넷 통신을 지원하기 위해 만든 프로토콜. 리소스 제약이 있는 기기들이 인터넷 상에서 TCP 대신 UDP를 사용해 커뮤니케이션 할 수 있도록 개발됐다. CoAP는 특히 인터넷을 통해 컨트롤 해야 하는 저출력 센서 및 기기와 커뮤니케이션 하는데 유용하다.

이클립스(Eclipse) 커뮤니티에서도 MQTT와 마찬가지로 CoAP를 공개 표준으로 지원하고 있다. CoAP는 상용적으로 지원되며, IoT 제공업체들과 함께 빠르게 성장하고 있다.

십여 년 전 개발된 프로토콜로부터 IoT의 요구에 맞추어 개조된 MQTT와 달리 CoAP는 IETF가 처음부터 제한된 환경에서 동작하는 제한된 기기의 경량 메시징의 IoT를 지원하기 위해 만들어졌다. CoAP는 간단한 프록시를 통해 HTTP와 RESTful 웹과 상호 운용되도록 설계되어 있어 본질적으로 인터넷에 적합하다.

CoAP는 클라이언트/서버 모델을 따르고 있다. 클라이언트는 GET, PUT, POST, DELETE 요청을 할 수 있다. CoAP 패킷은 비트필드(bitfields)를 이용해 메모리 효율을 극대화 하고 데이터 패킷을 온-디바이스로 트랜스포트 할 수 있을 정도로 작게 유지하기 위해 스트링~인티저(strings to integers) 매핑을 광범위하게 사용한다.  
  
장점 :

RESETful 프로토콜로 HTTP를 기반으로 REST 아키텍처를 적용할 수 있다. 동기, 비동기 메시지 교환을 모두 지원하며 제약조건이 많은 단말이나 네트워크를 고려해 신뢰성있는 유니캐스트, 멀티캐스트를 UDP 프로토콜을 이용하여 지원한다. 적은 헤더 오버헤드로도 데이터 전달이 가능해 간단하게 파싱할 수 있는 헤더이다.

1. 매우 작은 패킷 사이즈
2. 의도적으로 TCP보다 신뢰성이 낮은 UDP에서 실행되며, 일관된 연결 대신 반복적 메시징에 의존해 신뢰성을 제공한다. 데이터 그램을 사용하기 때문에 SMS같은 패킷 기반 테크놀로지보다 성능이 뛰어나다.

모든 CoAP 메시지는 ‘확인 가능’ 또는 ‘확인 불가능’으로 표시될 수 있어 애플리케이션 레벨의 QoS로 행동할 수 있다. SSL/TLS 암호화가 UDP에서는 불가능 하지만 CoAP는 TLS의 TCP버전과 비슷한 DTLS(Datagram Transport Layer Security)를 사용한다.

UDP의 비연결 데이터그램은 더 적은 오버헤드와 더 작은 패킷, 그리고 보다 빠른 웨이크업과 전송 사이클을 실행할 수 있게 한다. 따라서 디바이스가 보다 오랫동안 휴면 상태를 유지할 수 있으므로 배터리 전력을 절약할 수 있다.

(2) 멀티캐스트 지원

CoAP 네트워크는 기본적으로 일대일 방식이지만, 일대다 또는 다대다 멀티캐스트 요구사항을 지원한다. CoAP 네트워크는 IPv6 위에 구축되기 때문에 멀티캐스트는 CoAP 내에 본질적으로 내재한다고 볼 수 있다. 따라서 일반 IPv6 주소뿐 아니라 디바이스를 위한 멀티캐스트 주소 지정이 가능하다. 그러나 휴면 상태의 디바이스에 전달되는 멀티캐스트 메시지는 신뢰할 수 없거나, 이러한 메시지를 수신하기 위해 정기적으로 웨이크업이 발생한다면 디바이스의 배터리 수명에 영향을 미칠 수 있다.

(3) 보안

CoAP는 UDP 전송 프로토콜 상에서 DTLS를 사용한다. TCP와 마찬가지로 UDP는 암호화되지 않지만 DTLS를 이용할 수 있으며, 이를 이용해 보안을 강화해야 한다.

(4) 리소스 & 서비스 검색

CoAP는 URI를 사용하여 네트워크 노드를 위한 표준을 제시하고 상호작용을 기대한다. 타겟 노드의 기능이 URI 세부사항에 의해 부분적으로 이해되기 때문에 이것은 메시지 패킷에 어느 정도의 자율을 허용한다. 달리 말하면, 배터리 구동 센서 노드는 한 종류의 URI를 가질 수 있으며, 라인 구동 흐름 제어 액추에이터는 다른 종류의 URI를 가질 수 있다.

배터리 구동 센서 노드와 통신하는 노드는 더 긴 응답 시간, 더 많은 반복 정보, 제한된 메시지 유형을 기대하도록 프로그래밍할 수 있으며, 라인 구동 흐름 제어 액추에이터와 통신하는 노드는 풍부하고 상세한 메시지를 매우 신속히 기대하도록 프로그래밍할 수 있다.

(5) 비동기식 통신

CoAP 프로토콜 내에서 대부분의 메시지는 요청/보고 모델을 사용하여 송수신되지만, 노드를 조금 떨어뜨려놓을 수 있는 다른 동작 모드도 있다. 예를 들면 CoAP는 MQTT의 pub/sub와 유사하면서 단순한 ‘observe’ 방식을 제공한다. 이 모드에서는 노드가 실제로 참여하지 않으면서 다른 노드를 관찰할 수 있다.

‘observe’ 모드의 한 예로, 노드 1은 특정 전송 유형에 대해 노드 2를 관찰할 수 있으며, 그런 후에 노드 2가 관련 메시지를 발행하면 언제라도 노드 1은 다른 노드를 깨우고 조회하면서 그 메시지를 수신한다.

중요한 점은 네트워크 노드 중 하나는 반드시 옵저버 (observer)를 위한 메시지를 보유하고 있어야 한다는 것이다. 이러한 특성은 MQTT의 브로커 모델과 유사하지만, CoAP에는 브로커 요구사항이 없다는 점이 다르다. 따라서 이 모드가 아닌 경우 옵저버를 위한 메시지를 보유하거나 대기열에 저장하기를 기대할 수 없다.

단점:

1. 원-투-원 프로토콜이라는 점. 그룹 브로드캐스트를 가능하게 한 확장을 이용할 수도 있지만 브로드캐스트 기능은 원-투-원 프로토콜에 내재적인 것이 아니다.
2. 퍼블리시-섭스크라이브 메시지 큐의 부재. (->간결x 직관x)
3. 표준 성숙도

현재 MQTT는 CoAP보다 성숙되고 안정적인 표준이다. 많은 IoT 개발업체들은 CoAP를 사용하는 유사한 네트워크보다 더 쉽게 MQTT 네트워크를 구축하고 매우 빠르게 실행할 수 있다.

1. 메시지 신뢰성

CoAP의 신뢰성은 MQTT의 QoS에 해당된다. CoAP는 ‘확인형(confirmable)’ 메시지와 ‘비확인형(non-confirmable)’ 메시지를 갖는 매우 간단한 방법을 제공한다. 확인형 메시지는 의도된 수신자로부터 확인 메시지(ACK)를 통해 수신을 확인 받는다. 이는 메시지가 수신되었다는 것을 확인하며, 내용이 올바르게 해독되었는지 여부를 확인할 수 없는 경우 전송을 멈춘다. 비확인형 메시지는 ‘fire and forget’ 방식이다.

-MQTT

MQTT는 1999년 IBM에서 개발된 프로토콜로서 제 한된 컴퓨팅 성능과 네트워크 연결 환경에서의 동작을 고려하여 설계된 대용량 메시지 전달 프로토콜이다. 2013년엔 OASIS (Organization for the Advancement of Structured Information Standards)에 의해 사물인터넷을 위한 메시지 프로토콜로 선정되었을 뿐만 아니라 배터리 에너지 소모 측면에서 효율적인 프로토콜임이 검증 되고 있다 (‘안정적인 사물인터넷 플랫폼을 위한 MQTT 프로토콜 기반 데이터 수집 솔루션에 관한 연구’,JKICE,2016)

MQTT(Message Queue Telemetry Transport)는 원격제어, 원격 측정을 위한 통신 프로토콜. 원격 검침기는 대부분 소형이며 통신 대역폭과 전원이 한정적인 환경에서 동작한다는 점에서 스마트폰 환경과 매우 유사하다. (그래서 MQTT의 Telemetry가 Thing으로 바뀌는 추세이다.) MQTT는 메모리 및 전력 이용을 효율화하기 위한 가벼운 패킷 구조를 채택하고 있으며,여기에 연결된 기기는 MQTT 브로커 상에 호스팅 되는 토픽에 인용된다. 어떠한 기기나 서비스가 데이터를 토픽에 발행하면, 여기에 인용된 모든 기기들은 자동적으로 갱신된 정보를 획득하게 된다.

MQTT는 ‘publish/subscribe’ 모델을 사용하며, MQTT 네트워크 노드 간에 메시지를 관리하고 라우팅하기 위해 중앙 MQTT 브로커(broker)를 필요로 한다. 이클립스는 MQTT를 중앙 브로커를 통해 다중 클라이언트 간에 메시지를 전달하는 ‘다대다(many-to-many)’ 통신 프로토콜로 설명하고 있다. MQTT는 TCP를 사용하여 ‘고신뢰성, 정렬, 에러 검사’를 특징으로 하는 트랜스포트 계층을 구현한다.

CF) IBM은 유전 장비와의 위성 통신을 위해 MQTT를 개발했다. MQTT는 기본적으로 신뢰성과 저전력을 특징으로 하므로 IoT 네트워크에 적용하기 적합하다.

그 후 MQTT 표준은 개방형 표준을 위한 단체인 OASIS 에 의해 채택되었으며 버전 3.1.1로 발표되었다. 또한 오픈 소스 스택과 컨설팅을 제공하는 많은 상업 회사와 이클립스(Eclipse) 커뮤니티 내의 지원을 받고 있다.

장점 :

퍼블리쉬/섭스크라이브 메시지 큐(message queue)와 다-대-다 송출. MQTT 브로커에 수명이 긴 방출 TCP 연결을 이용하고 제한된 대역폭의 메시지를 전후방으로 전송함으로써 MQTT는 간결함과 직관성을 보장한다.

실시간 푸시 전송을 하면서도 전력사용은 최소화하며 비동기 메시징과 신뢰성이라는 특징. 경량 메시지 포맷을 사용하며 80~100Kb 코드 메모리만 사용해 간단한 구현 가능.

이벤트 방식을 사용해 다수의 사용자와 디바이스 지원. 응용코드 변경 없이도 기능 확장 가능.

탐지, 저장, 전달, 발행/구독 기능을 제공하는데에 있어 별도의 응용 로직이 필요하고 간단한 개념으로 만들어져 개발자의 학습에 용이함.

1. Publish/Subscribe 모델

MQTT의 ‘pub/sub’ 모델은 확장성이 뛰어나고 전력 효율적이다. 브로커와 노드는 정보를 발행하며, 다른 노드들은 메시지 내용, 종류 또는 주제에 따라 구독한다. 일반적으로 브로커는 모든 메시지를 구독한 다음 각 노드에 대한 정보 흐름을 관리한다. publish/subscribe 모델에는 몇 가지 뚜렷한 장점이 있다.

2) 공간 분리

(노드와 브로커는 서로의 IP 주소를 가질 필요가 있지만, 노드는 정보를 발행할 수 있으며, 모든 것이 중앙 브로커를 거쳐 진행되기 때문에) 서로를 전혀 모르고 있어도 다른 노드에서 발행하는 정보를 구독할 수 있다. 이는 TCP 세션과 포트에 수반될 수 있는 오버헤드를 감소시키고 최종 노드가 서로에 대해 독립적으로 동작할 수 있게 한다.

3) 시간 분리

노드는 다른 노드의 상태와 관계없이 정보를 발행할 수 있다. 그 후에 활성화될 때 다른 노드는 브로커로부터 발행된 정보를 수신할 수 있다. 이를 통해 다른 노드가 해당 노드에 직접 관련된 메시지를 발행할 때조차 노드가 휴면 상태를 유지할 수 있게 한다.

4) 동기화 분리

동작 중인 노드는 구독 중인 메시지가 발행되더라도 이를 수신하도록 인터럽트되지 않으며, 메시지는 수신 노드가 기존 동작을 완료할 때까지 브로커에 의해 대기열에 저장된다. 이는 진행 중인 동작 또는 휴면 상태에 대한 인터럽트를 피함으로써 동작 전류를 절약하고 반복되는 동작을 줄여준다.

5) 보안

MQTT는 새롭게 개발된 보안 형식은 아니며, 암호화되지 않은 TCP를 사용한다. TCP를 사용하는 이유는 TLS/SSL 인터넷 보안을 사용할 수 있기 때문이다. 그러나 TLS는 필요한 핸드셰이크와 증가되는 패킷 오버헤드로 인해 경량화된 클라이언트에게 리소스 집약적이다. 에너지가 매우 높은 우선순위를 가지고 있고 보안이 훨씬 덜 중요한 네트워크의 경우 패킷 페이로드 암호화만으로 충분할 수 있다.

6) MQTT 서비스 품질 단계

‘서비스 품질(QoS, Quality of Service)’이라는 용어는 MQTT 밖에서는 다른 것을 의미한다. MQTT에서 ‘QoS’는 0, 1, 2 단계가 있으며, 메시지 전달 보장이 증가되는 수준을 가리킨다.

7) LWT

MQTT는 노드가 예기치 않게 네트워크로부터 연결이 끊어지는 경우 MQTT 브로커에 저장할 수 있는 ‘LWT(last will and testament)’ 메시지를 제공한다. 이 LWT는 발행하고 구독된 명령의 종류를 포함하여 노드의 상태와 목적을 보유한다.

노드가 사라지면, 브로커는 모든 구독자에게 노드의 LWT를 통지한다. 노드가 되돌아오면, 브로커는 노드에게 이전 상태를 통지한다. 이 기능은 손실이 많은 네트워크와 확장성을 수용하기에 적합하다.

8) 토픽 구독

MQTT 노드는 주어진 기능 안에서 모든 메시지를 구독할 수 있다. 예를 들어 ‘키친 오븐 노드(kitchen oven node)’는 와일드카드로서 ‘+’와 함께 ‘kitchen/oven/+’에 대한 모든 메시지를 구독할 수 있다. 이는 최소한의 코드(메모리 및 비용)를 사용할 수 있게 한다.

또 다른 예로, 만약 키친 안의 노드가 최종 노드의 기능과 관계없이 모든 온도 정보에 관심을 갖고 있다면, ‘kitchen/+/temp’는 ‘temp’를 보고하는 모든 노드로부터 키친 안의 모든 메시지를 수집한다. MQTT 와일드카드는 코드 풋프린트를 줄이고, 그에 따라 메모리 크기와 비용을 줄일 수 있어 매우 유용하다.

[CF]

MQTT QoS 0

흔히 ‘Fire and forget’이라고 하며, 단 한 번의 전송으로 끝나므로 메시지 도착을 보장하지 않는다. 매우 반복적인 메시지 유형이나 업무에 중요하지 않은 메시지에 사용할 수 있다.

MQTT QoS 1

이 단계는 메시지가 의도된 수신자에 의해 최소 한 번 수신되는 것을 보장한다. 발행된 메시지가 수신되고 의도된 수신자에 의해 이해되면, 발행 노드로 보내는 확인 메시지(PUBACK)를 통해 메시지의 수신을 확인한다. 발행자가 PUBACK를 수신할 때까지 메시지가 저장되고 정기적으로 재발송된다. 이러한 종류의 메시지는 중요하지 않은 노드 셧다운에 유용할 수 있다.

MQTT QoS 2

이 단계는 의도된 수신자에 의해 메시지가 수신되고 해독되는 것을 보장한다. 이 단계는 가장 높은 보안의 신뢰할 수 있는 MQTT QoS 단계이다. 발행자는 메시지를 전송하면서 QoS 2 메시지를 갖고 있다는 것을 알린다. 의도된 수신자는 알림을 수집하고, 이를 해독한 다음, 메시지를 수신할 준비가 되었다는 것을 나타낸다. 발행자는 메시지를 중계한다. 수신지가 메시지를 이해하면, 수신 확인을 이용해 트랜잭션을 완료한다. 이러한 종류의 메시지는 가정에서 조명 또는 경보를 켜거나 끄는 용도에 활용할 수 있다.

단점 :

1) 항시 연결을 유지하는 구조는 기기의 휴식 시간을 제한한다. 하지만 MQTT는 기기의 대부분이 휴식하는 시점에도 다른 MQTT 프로토콜인 MQTT-S(TCP가 아닌 UDP와 동작한다)를 활용함으로써 이러한 문제를 방지한다.  
  
2) 베이스 프로토콜의 암호화 부재. MQTT는 가벼운 프로토콜로 제작되었기 때문에 여기에 암호화를 더하게 되면 커넥션에 엄청난 오버헤드를 더하게 될 것을 방지해 암호화를 하지 않았다. 때문에 커스텀 보안을 애플리케이션 레벨에서 더할 수는 있지만 꽤 성가신 작업이 될 수도 있다.

3) 중앙 브로커

중앙 브로커를 이용하는 것은 분산화된 환경의 IoT 시스템에는 단점이 될 수 있다. 예를 들어 시스템은 리모트컨트롤과 윈도우 셰이드만 갖추고 작게 시작할 수 있으며, 이때 중앙 브로커는 필요하지 않다. 그런 후 보안 센서나 전등 또는 기타 다른 윈도우 셰이드를 추가하면서 시스템이 확대되면 네트워크는 자연적으로 확장하며 중앙 브로커가 필요할 수 있다.

이미 중앙 브로커가 있는 시스템의 경우, 만약 브로커가 배터리 백업 없이 구동되는 노드라면, 전기 공급이 끊기는 경우 배터리로 구동되는 노드들은 계속 동작할 수 있는 반면, 브로커는 오프라인에 있게 되므로, 네트워크는 제대로 동작할 수 없게 된다.

4) TCP

TCP는 원래 많은 메모리와 프로세싱 리소스를 갖는 기기를 위해 설계되었다. 따라서 경량 IoT 방식의 네트워크에서 사용하기에는 적합하지 않다.

TCP 프로토콜은 메시지를 교환하기 전에 다중 단계의 핸드셰이크 과정을 통해 연결을 맺어야 한다. 이러한 방식은 웨이크업 시간과 통신 시간을 늘리고, 장기적으로 배터리 수명을 단축시킨다.

또한 TCP에서는 두 통신 노드가 지속적 세션에서 연속적으로 서로에 대해 TCP 소켓을 개방하는 것이 이상적이다. 이러한 특성은 에너지와 리소스가 제한된 디바이스에는 적용하기 어려울 수 있다.

5) 웨이크업 시간

세션의 지속 없이 TCP를 사용하는 것은 연결을 맺는 데 증가되는 전송 시간을 필요로 할 수 있다. 정기적, 반복적 트래픽을 갖는 노드의 경우 이는 동작 수명을 줄일 수 있다.

사용 사례 : 페이스북 메신저. 페이스북의 안드로이드용 메신저 앱은 MQTT 푸시 서비스가 백그라운드에서 실행됨. -> 주기적으로 폴링하는데 비해 배터리 및 패킷 소모량이 적음. 수많은 사용자가 동시에 메시지를 전송하더라도 짧은 시간 내에 메시지가 전송될 수 있는 구조.

[IDG Tech Focus에서 발간한 실시간&IoT: 빅데이터 분석의 새로운 도전과 해법]

데이터 획득 및 저장 IoT 기기의 데이터를 수신할 수 있는 여러 프로토콜이 있다. IoT 장치가 블 루투스, 셀룰러, 와이파이, 또는 하드웨어 연결 등 어떤 연결 방식을 사용하는지는 여기서 문제가 아니다. 다른 프로토콜을 사용하는 ‘중개’ 장치에 메시지를 전송할 수 있으면 된다. 가장 광범위하게 지지받고 있는 인기 프로토콜은 MQTT(Message Queue Telemetry Transport)다. 이 밖에도 CAP(Constrained Application Protocol), XMPP 등의 여러 대안들이 있다. 필자는 다른 대안 기술을 선택해야만 하는 확실한 이유가 없다면 MQTT를 출발점으로 삼을 것을 권장한다. 폭 넓게 지지를 받으면서 활용되고 있고, 여 러 오픈 소스 클라이언트와 중개 애플리케이션이 있다는 점을 감안해서다. 참고로 모스키토(Mosquitto)는 가장 잘 알려져 있고, 가장 많이 사용되는 오픈소스 MQTT 중개 기술(broker)로, 애플리케이션에 대한 좋은 선택이 될 수 있다. 오픈소스이기 때문에 소규모 예산으로 추진하는 개념 증명 프로젝트에 특히 도움이 된다. 값비싼 사유(proprietary) 시스템에 투자할 필요가 없기 때문이다. 어떤 프로토콜을 선택하든 연결된 장치의 관찰이나 이벤트에 해당하는 메시 지를 갖게 될 것이다. 모스키토 같은 중개 기술은 이 메시지를 수신한 후 이를 분석 시스템으로 보낼 수 있다. 여기서 베스트 프랙티스는 소스 데이터를 변환 또는 변경하기 전에 저장하는 것이다. 변환 단계의 디버깅 문제를 해결해야 할 때, 또는 전체 테스트나 히스토리 분석을 위해 메시지 시퀀스를 재전송할 때 아 주 유용한 역할을 하기 때문이다.

* (‘실시간 교통빅데이터 시뮬레이션을 위한 IoT 시뮬레이터 설계 및 구현’, 충남대 대학원, 안상희, 2017) 이라는 논문에서도 MQTT 사용 (QoS1).

참고하면 도움이 될 듯.

* (정인택, 성홍기, 장윤미, 정규수 (2018). MQTT 기반 실시간 차량센싱데이터 수집 및 정보제공 플랫폼 개발. 대한공간정 보학회 학술대회, 256-257)

MQTT를 통해 실시간으로 수집되는 센싱데이터와 Open API를 통하여 실시간으로 수집되는 공공데이터를 수집, 저장, 가공, 분석 및 시각화하기 위한 플랫폼을 개발하였다.

* (하승현, 김관형 (2019). MQTT 기반 데이터 관리시스템. 한국정보통신학회 종합학술대회 논문집, 23(2), 168-169)

MQTT는 전력이 적게 들며 여러 곳과 통신을 할 수 있어 작은 데이터만으로 구현이 가능한 IOT 시스템에 매우 효과적이다. 그러나 많은 데이터를 보내는 분야에는 적절하지 않고 아직 정보가 누락되는 문제가 있다.

0.3초마다 보내는 신호를 받았을 때 통신 데이터의 누락되는 현상을 확인할 수 있었다. -> 이 이상으로 해야 안정적으로 데이터 수신 가능. 더 많은 수 노드 연결했을 때 통신주기가 더 길어질 수 있을 듯.

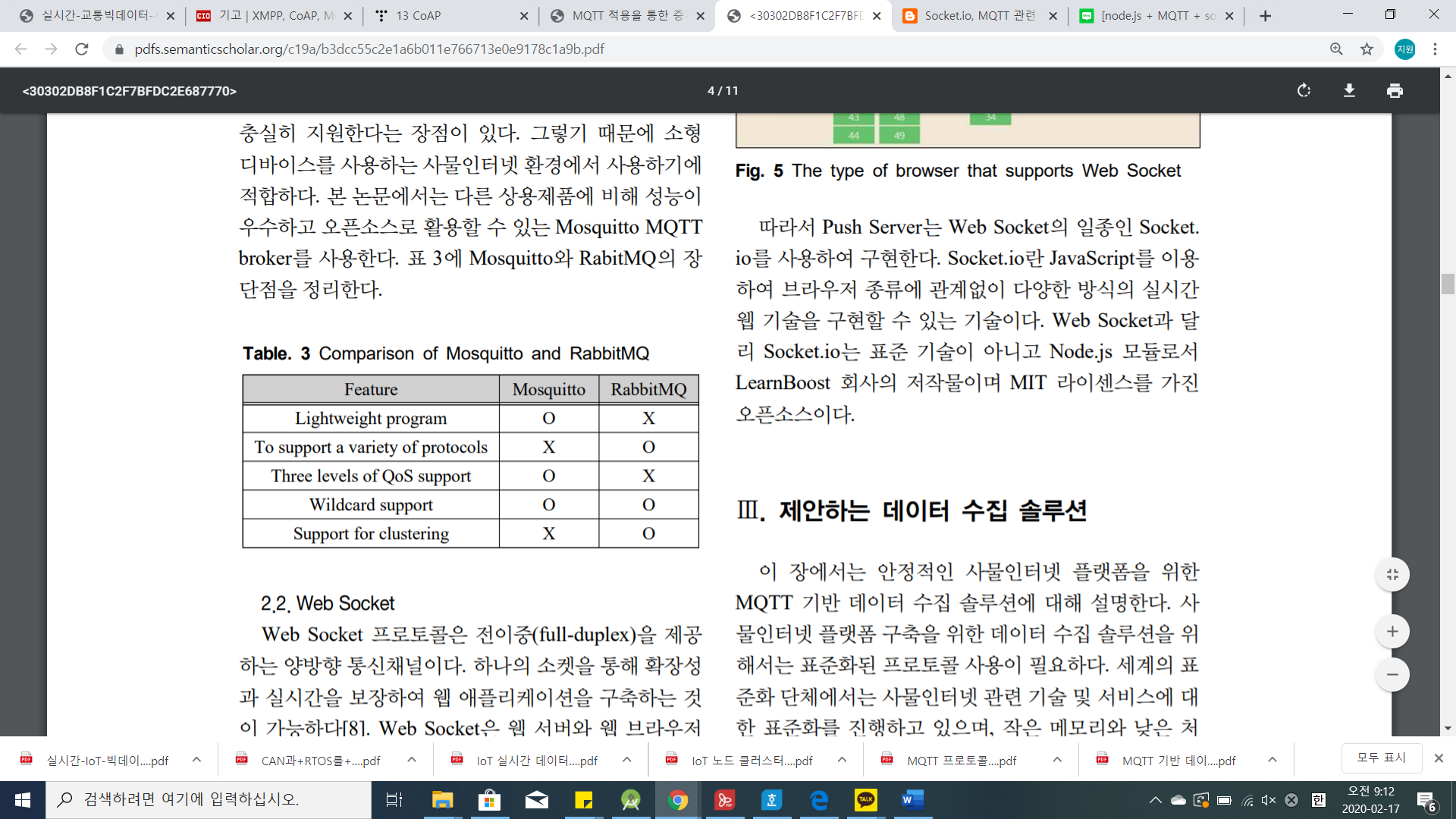
MQTT 브로커 종류:

Mosquitto, RabbitMQ, Adafruits, Thingspeak, HiveMQ.. ~!~

대부분 Mosquitto를 가장 많이 이용하고, RabbitMQ와 HiveMQ도 많이 사용하는 듯하다. 개인 취향에 따르는 듯하다.

* (‘안정적인 사물인터넷 플랫폼을 위한 MQTT 기반 데이터 수집 솔루션 관한 연구’, 한국정보통신학회논문지,김상현,김동회,오형석,전현식,박현주,2016)

MQTT는 다양한 종류의 Broker를 가지고 있다. 그 중에서도 가장 대중적으로 활용되는 Mosquitto와 RabbitMQ가 있다. Mosquitto 브로커는 MQTT 전용 브 로커로 경량이며 브로커가 가져야 할 대부분의 기능을 충실히 지원한다는 장점이 있다. 그렇기 때문에 소형 디바이스를 사용하는 사물인터넷 환경에서 사용하기에 적합하다. 본 논문에서는 다른 상용제품에 비해 성능이 우수하고 오픈소스로 활용할 수 있는 Mosquitto MQTT broker를 사용한다. 표 3에 Mosquitto와 RabitMQ의 장 단점을 정리한다.



(<https://github.com/mqtt/mqtt.github.io/wiki/server-support>)

