공학석사 학위논문

ZigBee 네트워크 상에서 CoAP 사용을 위한 시스템 설계

컴 퓨 터 정 보 공 학 과

정 민 근

지도교수 김 창 훈

2014년 6월

대구대학교 대학원

ZigBee 네트워크 상에서 CoAP 사용을 위한 시스템 설계

이 논문을 공학석사 학위논문으로 제출함.

컴 퓨 터 정 보 공 학 과

정 민 근

지도교수 김 창 훈

정민근의 공학석사 학위논문을 인준함.

2014년 6월

심사위원장 ——————(인)

심 사 위 원 ——————(인)

심 사 위 원 ——————(인)

대구대학교 대학원

목 차

표 목차

그림 목차

ZigBee 네트워크상에서 CoAP 사용을 위한 시스템 설계

(요약)

최근 사물 인터넷(Internet of Things)이 화제가 되고 있다. 사물인터넷과 관련된 각종 표준화가 진행되고 있는 가운데, CORE(Constrained RESTful Environments) 워킹그룹에서는 메모리, 에너지, 성능 등에 제약이 있는 통신 환경을 위한 웹 기반 프로토콜인 CoAP(Constrained Application Protocol) 이라는 프로토콜을 만들고 있다. CoAP은 기존의 웹에서 사용하던 HTTP가 통신이 불안정하고 메모리가 작은 저사양 노드들에게 적용하기는 무겁고 적용에 어려움이 있어 그를 대신하기 위해 개발된 프로토콜이다. UDP 프로토콜의 데이터그램을 이용하여 UDP 계층과 어플리케이션 계층 사이에서 추가 레이어를 사용하여 웹을 지원하기 위한 기술이다. 따라서 기존의 ZigBee를 사용하던 센서 노드들에게 적용하기 위해서는 6LowPAN과 UDP의 추가적인 구현이 필요 하였다. 본 논문에서는 ZigBee를 사용하던 기존 센서 노드들의 스택은 변경하지 않고 어플리케이션 계층에서의 구현을 통해 간단하게 CoAP을 적용할 수 있는 시스템을 제안 한다.

제안된 시스템에서는 게이트웨이 역할을 하는 브로커(Broker)라는 디바이스를 추가적으로 사용해 기존의 ZigBee 센서 노드들이 브로커를 통해 웹과 연결을 가능하게 한다. 브로커는 ZigBee 스택과 TCP/IP 스택을 모두 가진 상태에서 멀티 통신이 가능하도록 설계되며 HTTP패킷과 CoAP패킷 간의 변환 또한 가능하도록 되어있다. 본 논문에서 제안한 ZigBee 네트워크상에서 CoAP 사용을 위한 시스템을 이용하여 사물인터넷에 좀 더 빠르고 쉽게 다가갈 수 있을 것으로 판단된다.

I. 서론

1. 연구배경 및 필요성

최근 사물 인터넷(Internet of Things)이 뜨고 있다.(이대영, 2013) 사물 인터넷이란 물건 하나하나가 인터넷과 연결된다는 의미이다. 사물 인터넷은 1, 2년 전에 불쑥 등장한 개념이 아니라 아주 오래 전부터 존재해 왔다. M2M, 사물지능통신, 유비쿼터스, RFID/USN 등이 그 예라고 할 수 있다. 이 가운데서도 M2M은 사물 인터넷과 가장 유사한 개념으로 이해할 수 있다. M2M의 가장 대표적인 비즈니스 모델은 Forrester Research(forrester Research)에 의하여 처음 제안된 Invisible Mobile 개념으로 사람이 직접 개입하지 않아도 디바이스로부터 자동적으로 이루어지는 통신으로 정의하고 있다.(이근호, 2003) M2M과 사물 인터넷은 사물 간 통신을 한다는 점에서 같아 많이 혼용해서 쓰이기도 한다. 사물 인터넷은 이동통신망을 이용하여 사람과 사물, 사물과 사물 간 지능통신을 할 수 있는 M2M의 개념을 인터넷으로 확장하여 사물을 몰론, 현실과 가상세계의 모든 정보와 상호작용하는 개념으로 진화되었다고 볼 수 있다.

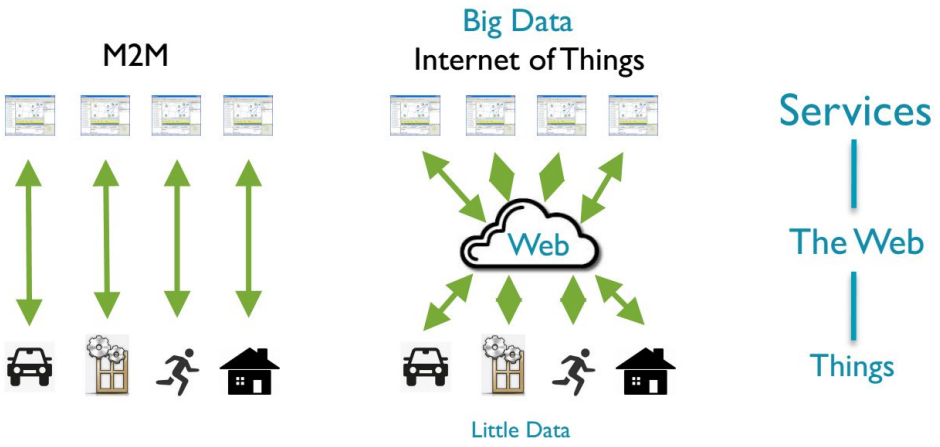

그림 1. M2M과 IoT(Zach Shelby, 2014)

가트너 그룹에서 2014년 가장 주목해야 할 10대 전략 기술(Gartner, 2013) 가운데 하나로 사물 인터넷을 뽑을 정도로 그 중요성이 확대 되고 있는 가운데 센서 및 각종 디바이스 간에 상호 통신할 수 있는 공통 프로토콜이 필요하게 됐다. 인터넷 표준 단체인 IETF(Internet Engineering Task Force)에서는, 다양한 기기가 인터넷에 연결될 것을 예상하여 IPv6, 6LowPAN 등 표준 활동을 진행해 왔으며, 최근에는 6LoWPAN, CORE, ROLL, LWIG 워킹그룹 등이 저전력, 소형 장치에 들어가는 표준을 개발하고 있다. 그 중 CORE(Constrained RESTful Environments) 워킹그룹에서는, 메모리, 에

너지, 성능 등에 제약이 있는 통신 환경을 위한 웹 기반 프로토콜인 CoAP (Constrained Application Protocol) 이라는 프로토콜을 만들고 있다.(Z. Shelby외, 2014) (고석갑 외, 2013)

2. 연구 목표

본 논문은 센서 노드들에게 CoAP을 적용하는 데 있어 기존의 6LowPAN과 UDP를 사용하여 작동하도록 하는 것이 아닌 ZigBee를 이용하여 바로 동작 할 수 있는 시스템을 제안 한다. 제안된 시스템에서는 Non-IP 프로토콜인 ZigBee를 그대로 사용하기 때문에 IP프로토콜인 6LowPAN에 있는 IP를 직접적으로 할당하지는 못하지만 게이트웨이 역할을 하는 브로커(Broker)라는 디바이스를 하나 더 추가함으로써 CoAP 사용이 가능하게 만든다. 웹서버를 통해 특정 센서 노드의 데이터를 읽기 위해 요청하게 되면 그 요청이 브로커로 전송되고 브로커에서 HTTP 패킷을 CoAP 패킷으로 변환시킨 후 노드 테이블을 통해 특정 노드에게 해당 요청을 전송한다. 요청받은 노드에서는 응답 메시지를 브로커로 전송하게 되고 브로커 에서는 HTTP 패킷으로 변환시킨 후 웹서버로 응답 메시지를 전달하게 된다. 이러한 방법을 통해 ZigBee를 사용하는 센서 노드들에게 어플리케이션 레벨에서 CoAP을 바로 사용할 수 있도록 지원한다.

3. 논문의 구성

본 논문의 구성으로는 1장에서 연구의 배경 및 필요성과 목표에 대해 논하고, 2장에서는 본 연구에 필요한 배경 기술인 ZigBee, HTTP, CoAP에 대해 살펴본다. 3장에서는 본 논문에서 제안한 시스템의 설계에 대해 상세히 논하며 4장에서 제안한 시스템을 구현한 결과를 정리한다. 마지막으로 5장에서 결론을 맺는다.

II. 관련 연구

본 장에서는 논문과 관련된 기술인 ZigBee와 CoAP 그리고 CoAP의 근본이 되는 HTTP에 대하여 설명한다.

1. ZigBee 프로토콜

1) ZigBee 특징

ZigBee는 저전력, 저가격, 사용의 용이성을 가진 근거리 무선 센서 네트워크의 대표적 기술 중 하나로 2003년 IEEE 802.15.4 작업분과위원회에서 표준화된 PHY/MAC 층을 기반으로 상위 프로토콜 및 응용을 규격화한 기술이다.(문용선 외, 2010) IEEE 802.15 표준을 기반으로 만들어졌다. ZigBee 장치는 메시 네트워크 방식을 이용, 여러 중간 노드를 거쳐 목적지까지 데이터를 전송함으로써 저전력 임에도 불구하고 넓은 범위의 통신이 가능하다. 애드혹 네트워크적인 특성으로 인해 중심 노드가 따로 존재하지 않는 응용 분야에 적합하다.

ZigBee의 다음과 같은 특징을 가진다.

- 저전력 소모, 간단한 구현
- 활성 모드(수신, 송신), 슬립 모드를 가짐.
- 디바이스, 설치, 유지 등 모두 상대적으로 낮은 비용으로 가능
- 안전성(보안성)
- 신뢰성
- 유연성
- 매우 작은 프로토콜 스택
- 상호 호환가능 및 어느 곳에서나 사용 가능
- 네트워크 당 높은 노트 밀집(ZigBee의 IEEE 802.15.4 사용은 네트워크에서 많은 디바이스를 다루는 것을 가능케 함. 이러한 특징으로 방대한 센서 배열과 네트워크의 통제가 가능)
- 간단한 프로토콜, 국제적으로 구현(ZigBee 프로토콜 스택 코드의 크기는 블루투스나 802.11의 사이즈에 비해 4분의 1 정도에 불과하다.)

2) ZigBee 구조

ZigBee의 구조는 크게 MAC 계층, 네트워크 계층, 어플리케이션 계층으로 구성되는 계층 구조를 갖는다. 보다 상세하게는 ZigBee 아키텍처의 가장 하위 스택에 PHY 계층도 포함시켜 4 계층으로도 생각할 수 있다. 하지만, MAC 계층과 PHY 계층이 한 개의 표준 사양으로 묶여져 있고, PHY 계층의 서비스는 물리계층에 프레임 데이터의 송수신과 송수신 트랜시버의 점멸 기능만을 포함하는 매우 단순히 기능을 지니므로 일반적으로 MAC 계층에 함께 포함시켜 다룬다. 다음 페이지의 그림2는 ZigBee의 스택 구조(ZigBee Specification, 2008)를 보여준다.

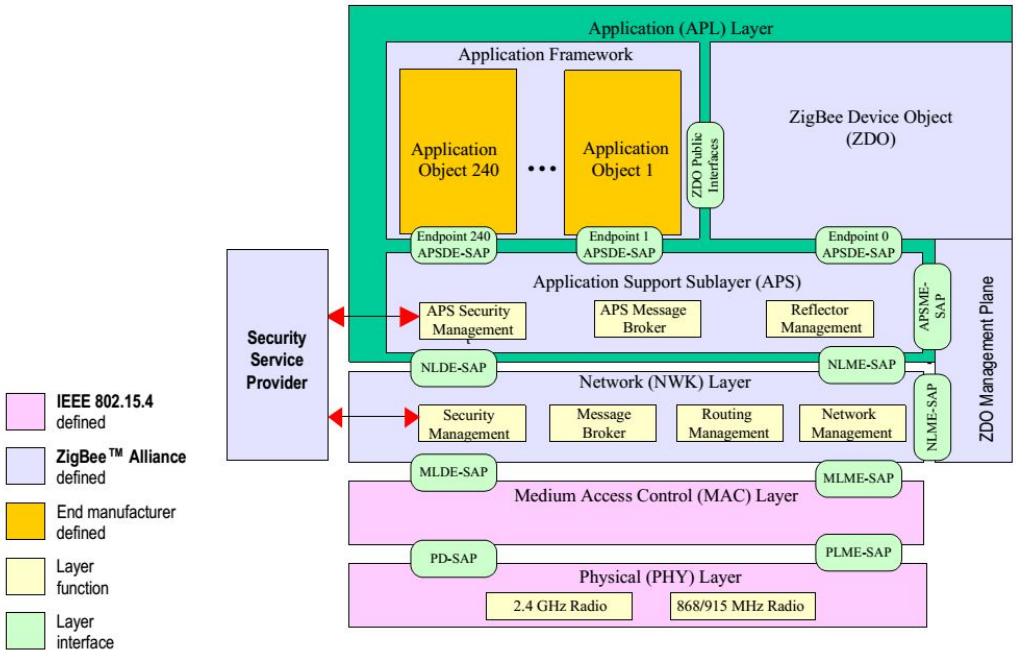

그림 2. ZigBee 스택 구조

■ Application 계층

Application 계층에는 지정된 응용 영역의 어플리케이션 로직을 구현한 어플리케이션 객체(Application Object)들이 위치한다. 이 어플리케이션 객체들은 일반 컴퓨터 아키텍처로 치면 운영체제 상위에 위치해 동작하는 어플리케이션 프로그램에 해당한다. ZigBee 스택의 어플리케이션 계층의 S/W 구성요소는 사용자 어플리케이션과 스택의 어플리케이션으로 구분할 수 있다. 사용자 어플리케이션은 개발자가 구현해 탑재시키는 어플리케이션이며, 스택 어플리케이션은 스택에서 제공하는 일종의 시스템 S/W이다. 어플리케이션의 주로 메시지 송수신과 관련 지원 기능을 제공하는 APS (Application Service Sub-layer)도 이 Application 계층에 속한다.

○ 어플리케이션 : 일반적으로 어플리케이션이라고 용어는 사용자 어플리케이션을 뜻하다. 어플리케이션은 응용 로직을 구현한 독립적인 프로그램이라고 볼 수 있다. 한 노드에는 탑재되어 구동할 수 있는 ZigBee의 어플리케이션의 최대 개수는 약 240개이다.

○ ZDO : ZigBee는 오직 ZDO(ZigBee Device Object)라는 한 개의 스택 어플리케이션만 제공하고 있다. ZDO는 노드가 네트워크 스택을 초기하고, 네트워크에 가입 등과 같은 네트워크 관리 기능을 담당한다. 아울러 어플리케이션에게 다른 노드로부터 정보를 취득 또는 탐색할 수 있는 정보관리 서비스를 제공해서 어플리케이션이 네트워크상의 다른 노드로부터 필요한 정보를 취득하거나 다른 노드에 변경할 수 수 있도록 한다.

○ APS : APS는 어플리케이션에 Fragmentation과 같은 메시지 송수신의 상세 기능과 관련 출발지와 도착지 관리 등의 기능을 지원한다. S/W 구현 측면에서는 일종의 라이브러리 API 모음으로 보면 된다.

■ 네트워크 계층

네트워크 계층은 (NWK 계층)의 주된 역할은 메시지의 라우팅 기능이다. ZigBee는 라우팅 프로토콜로서 AODV(Ad-hoc On-Demand Vector)를 약간 변형해 얻은 프로토콜을 사용한다. 네트워크 계층의 사양은 End-to-End 라우팅을 위한 데이터 서비스와 네트워크 형성 및 구축에 필요한 제반 관리서비스로 제공한다. 이 관리 서비스의 사례로는 주변의 네트워크 탐색, ZC의 네트워크 초기 형성, 기 구축되어 있는 네트워크에의 가입 등이 있다. 일반적으로 데이터 서비스와 관리서비스 모두를 하나의 S/W 모듈로 구현하므로 NWK계층에는 이외에 S/W 구성요소는 없다.

■ MAC 계층

MAC 계층은 데이터 링크 계층에서 이웃노드 간의 Point-to-Point 데이터 포워딩을 위한 데이터 서비스와 네트워크 형성에 필요한 관리 서비스를 제공한다. PHY를 포함한 MAC계층은 데이터 링크 계층에서 이웃노드 간의 Point-to-Point 데이터 포워딩을 위한 데이터 서비스와 네트워크 형성에 필요한 관리 서비스를 제공한다. ZigBee 아키텍처의 가장 하위 계층의 이 MAC 계층에는 ZigBee 자체의 사양이 아닌 IEEE 802.15.4 표준을 채택해 사용하고 있다. 따라서 ZigBee 스택의 구현을 위해서는 MAC S/W 모듈이 필수적으로 포함되어야 하므로 IEEE 802.15.4 표준에 대한 ZigBee 스택의 개발자는 명확한 이해가 필요하다. MAC계층을 구성하는 S/W 구성요소도 NWK 계층과 마찬가지로 오직 MAC 모듈뿐으로 보면 된다.

ZigBee의 버전별로 스택의 아래의 표1과 같이 기능과 성능의 차이가 있다.

표 1. ZigBee 버전별 스택 비교

기능	ZigBee 2006	ZigBee 2007	ZigBee Pro
Size in ROM/RAM	Smallest	Small	Bigger
Stack Profile	0x01	0x01	0x02
Maximum hops	10	10	30
Maximum nodes in network	31,101	31,101	65,540
Mesh Networking	Yes	Yes	Yes
Broadcasting	Yes	Yes	Yes
Tree routing	Yes	Yes	No
Frequency Agility	No	Yes	Yes
Bandwidth used by Protocol	Least	More	Most
Fragmentation	No	Yes	Yes
Multicasting	No	No	Yes
Source routing	No	No	Yes
Symmetric Links	No	No	Yes
Standard Security (AES 128 bit)	Yes	Yes	Yes
High Security (SKKE)	No	No	Yes
Profiles support	Home Automation	Home Automation	Home Automation Smart Energy Commercial Building Industrial Plant Monitor

3) ZigBee 스택의 소프트웨어 구성

소프트웨어 구성 관점에서 보면, ZigBee 스택의 기본적인 구성요소는 MAC 모듈, 네트워크 모듈, APS, ZDO, 어플리케이션들로 생각할 수 있다. 이 기본적인 구성요소와 더불어 ZigBee 스택의 중요한 구성요소 중의 하나는 보안서비스이다. ZigBee 보안 서비스는 APS 계층과 NWK 계층에 거쳐 암호화와 무결성 유지를 통한 데이터 보호, 암호화키의 관리 및 분배 등을 위한 보안 아키텍처와 서비스를 제공한다. 이 보안 서비스는 ZDO 와 어플리케이션에 필요한 API를 제공한다. 소프트웨어 구현 관점에서 보면, 각 S/W 구성요소는 독립적으로 모듈로 존재하지는 않는다. 각 벤더들의 스택을 구현하는 자체 방식에 의거 필요에 따라 여러 구성요소들이 하나의 모듈에 합쳐진 형태로 구현될 수 있다.

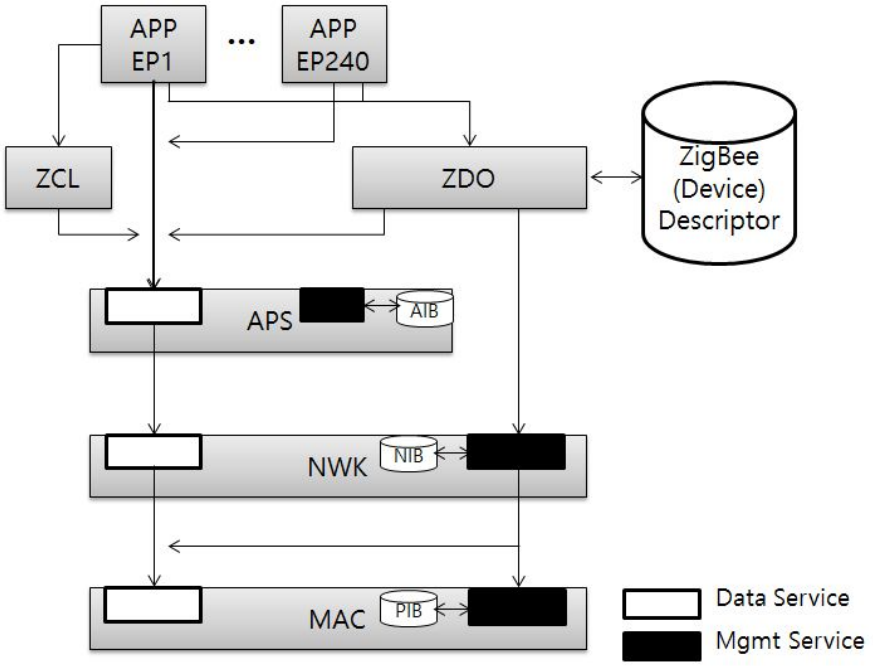

그림 3. ZigBee 데이터 흐름도

4) 네트워크 노드

무선 네트워크는 지정된 라우팅 방식을 의거해 노드 간에 메시지를 전달하며 서로 라디오 전송(Radio transmission)을 통해 서로 통신하는 여러 개의 노드로 구성된다. ZigBee 네트워크도 다수의 노드들로 구성되며, 일반적으로 역할에 따라 노드들을 3개의 유형으로 구분한다. 첫 번째는 코디네이터(Coordinator)이며, 두 번째는 라우터(Router), 그리고 마지막은 종단 디바이스(End Device)이다. 경우에 따라, 코디네이터

는 ZC (ZigBee Coordinator), 라우터는 ZR(ZigBee Router), 종단 디바이스는 ZED(ZigBee End Device)로 약어로 불린다.

각 노드는 유형에 따라 네트워크에서 특정 역할을 가지만, 어플리케이션 레벨에서는 일반적인 노드로서의 동작하며 자신을 위한 데이터 송수신은 수행한다.

ZigBee 네트워크에서 일부 디바이스는 ZC, ZR, ZED 의 역할을 가질 수 있으나, 그 역할에 관계없이 각 노드는 조도 측정 어플리케이션을 탑재해 실행시킬 수 있다. 다음의 표2는 각 역할에 대해 간략히 나타낸 것이다.

표 2. ZigBee 디바이스 알림표

유형	설명	비고
ZC	○ PAN에 한 개만 존재하며, PAN을 형성시킴 ○ 802.15.4 스펙에서는 Coordinator로 호칭 ○ 기능 - Beacon 모드에서는 Beacon 전송 - Relay dataframe, route discovery & repair - 타 노드로부터 Join request 처리 등	○ Network 주소 값 : 0 ○ FFD (Full Functional Device)
ZR	○ ZigBee Router ○ PAN에 한개 이상 존재 가능 ○ 기능 - 메시지 중계를 통한 라우팅 기능 - Relay dataframe, route discovery & repair	○ FFD (Full Functional ○ Device) ○ Network 주소 값 : ○ JOIN시 할당
ZED	○ ZigBee end device ○ PAN에 한개 이상 존재 ○ 단순한 데이터 전송 기능	○ RFD (Reduced ○ Functional Device) ○ Network 주소 값 : ○ JOIN시 할당

■ Coordinator (ZC)

ZigBee 네트워크의 망구성은 자율적으로 이루어지며, ZC는 이 자율적 망 구성을 시작하는

역할을 갖는다. 망구성의 주도적 역할은 하나의 노드에서 수행하는 것이 자연적이므로 ZC는 ZigBee 네트워크에 한 개만 존재한다. ZC의 망 구성 역할은 다른 노드들이 자신을 통해 네트워크에 가입하도록 하는 것이다. 망 구성을 마친 후에는 수신한 메시지를 다른 노드로 전달하는 라우팅의 역할과 일반 노드로서 메시지를 송수신을 수행하게 된다.

- 망 구성 : ZC가 수행하는 망구성의 역할은 네트워크에서 사용할 주파수 채널을 선택하고, 네트워크 ID 들을 설정하며, 주위 노드들이 자신을 통해 네트워크에 가입하도록 하는 것으로 구성된다. ZC를 통해 네트워크에 가입한 노드들을 ZC의 Child 노드라고 하며, ZC는 Child 노드의 Parent 노드가 된다. ZC는 한 개 이상의 자식노드를 가질 수 있다.
- 보안관리 : 추가적으로 응용계층의 서비스로 서 보안 관리 서비스도 제공할 수 있다.

■ Router (ZR)

ZR은 수신한 메시지를 다른 노드에 전달하는 라우팅 기능을 제공하는 노드이다. ZigBee 네트워크에는 다수의 ZR이 존재할 수 있다. ZR의 라우팅 역할은 네트워크 계층에서 구현되며,

추가적인 기능으로 다른 노드를 자신을 통해 가입시켜 네트워크 확장에 기여한다.

- 메시지 중계 : 라우팅 기능을 제공하며, 타 노드를 위한 메시지 전달을 위해 라우터는 휴지상태(SLEEP) 없이 상시 동작 상태에 있도록 설정된다.
- 네트워크 확장 : ZC 와 같이 다른 노드들을 자신을 통해 가입시킬 수 있는 기능을 제공 한다.이를 통해 ZC 는 네트워크의 확장을 위한 중요한 역할을 하게 된다. 따라서 ZC 와 마찬가지로 ZR도 한 개 이상의 자식노드를 가질 수 있다.

■ End Device (ZED)

어플리케이션 수행에 중점을 둔 노드로서 라우팅 서비스는 제공하지 않고 주로 자신의 데이터의 송수신만을 수행하는 노드이다. 네트워크에는 많은 개수의 ZED가 존재한다. ZC나 ZR과는 달리, 다른 노드를 자신을 통해 가입시키는 기능을 지니고 있지 않다. 따라 ZED는 Child노드를 갖지 못하며, 네트워크상에서 항상 다른 노드의 Child 노드로서만 존재하게 된다.

○ 메시지 송수신 : 데이터 송수신은 자신의 Parent 노드를 통해서만 가능하다. 타 노드로의 메시지 전송은 자신의 Parent 노드에 메시지를 전송하게 되며, 타 노드로부터의 데이터는 최종적으로 Parent 노드에게 메시지가 전달되며, Parent 노드를 통해 이 메시지를 수신하게 된다.

○ 동작 모드 : 일반적으로 배터리로 동작하며, 에너지 절감을 위해 송수신을 하지 않는 동안에는 휴지상태(SLEEP)로 전환될 수 있다. 휴지상태의 ZED는 데이터를 수신할 수 없으며, 휴지상태에 자신에게 전송되는 메시지는 Parent 노드에 보관되며, 이후 ZED가 정상상태로 돌아오면, Parent에 요청해서 해당 메시지를 받는다.

5) 노드간 연결

각 디바이스 간 연결 모습은 그림3과 같다. 앞 절에서 설명 했듯이 ZC는 ZigBee 네트워크에서 하나만을 가지며 다른 Child 노드들을 가진다. ZR 뿐만 ZED도 직접적으로 Child 노드로 가질 수 있으며 ZR은 ZED만 Child 노드로 가질 수 있다. 그림4의 좌측 상단에 보이듯이 ZR을 중심으로 독립적인 네트워크 구성도 가능하다.

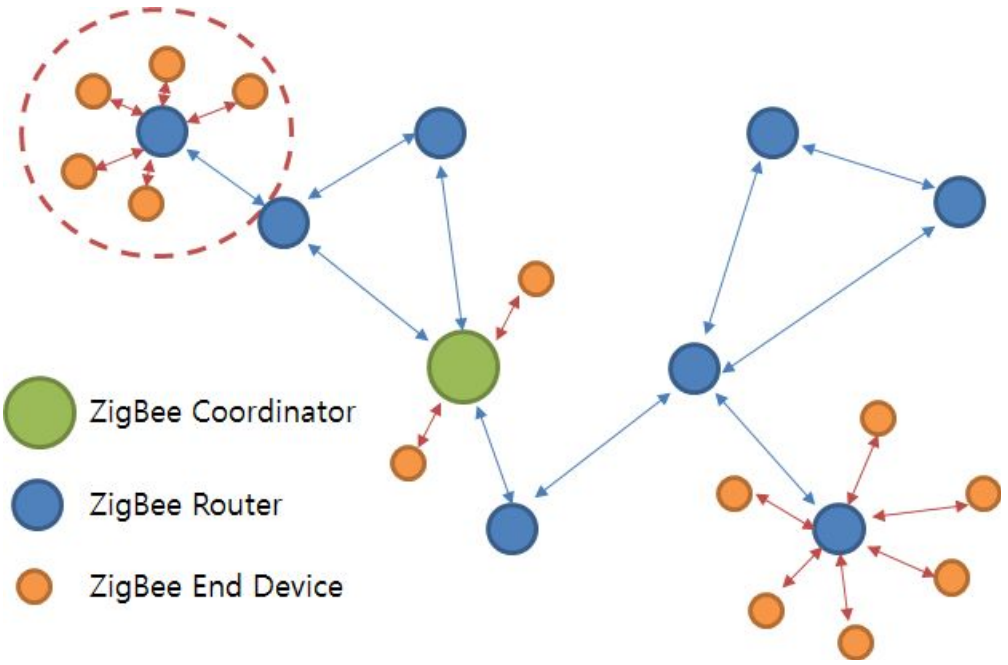

그림 4. ZigBee 네트워크 노드 간 연결

6) 토폴로지

네트워크의 토폴로지는 일반적으로 네트워크를 구성하는 노드들 간의 단일 홉뿐만 아니라 멀티- 홉 통신의 경로 정보를 나타낸다. 따라서 네트워크의 토폴로지는 주로 네트워크상에서 End-to-End 메시지 전송을 다루는 라우팅 관점에서 논의되는 것이 일반적이다. 하지만, 무선 네트워크의 경우에는 여러 노드들이 연합해서 네트워크를 구성하게 되므로 네트워크 형성 관점에서도 토폴로지를 살펴보아야 한다.

- 물리적 토폴로지 : 네트워크 형성 과정의 노드 간의 상호 연결성은 기본적으로 전파범위내의 이웃노드 간의 관계를 통해 이루어지며, 네트워크 형성 과정을 통해 구성된 네트워크 정보를 나타내는 토폴로지를 말한다. 참고로, 물리적 토폴로지는 네트워크 계층의 네이버 테이블(Neighbor table)에 반영되어 관리된다.
- 라우팅 토폴로지 : 메시지 전송을 위한 토폴로지를 말한다. 메시지 라우팅을 위한 토폴로지는 이러한 이웃노드 간의 관계를 바탕으로 여러 홉에 거친 경로 정보를 나타낸다. 참고로 라우팅 토폴로지는 네트워크 계층의 라우팅 테이블(Routing table)에 반영되어 관리된다.

ZigBee는 네트워크의 구성에 있어 트리 형태의 구조를 채택하고 있어 트리 토폴로지를 물리적 토폴로지로 채택하고 있다. 한편, ZigBee는 라우팅에 있어 초기에는 트리 구조 기반의 토폴로지를 채택하였지만, 이후 버전에서는 메시 프로토콜을 추가・지원하고 있다. 특히, ZigBee PRO는 이전의 트리 라우팅을 더 이상 지원하지 않으며, 단지 메시 토폴로지만을 지원하고 있다.(UTRC, 2013)

2. HTTP

HTTP(Hypertext Transfer Protocol)는 인터넷상에서 데이터를 주고받기 위한 서버/클라이언트 모델을 따르는 프로토콜이다.(W3C, 2014) TCP/IP 위의 어플리케이션 레벨에서 동작한다. 가장 성공적인 인터넷 프로토콜로 볼 수 있으며 HTTP로 작성된 문서뿐만 아니라 이미지, 동영상, 오디오, 텍스트 등의 어떤 종류의 데이터든지 전송할 수 있도록 설계되어 있다.(최미정 외, 2011)

1) 작동 방식

그림 5. HTTP의 클라이언트/서버 작동 방식

HTTP는 서버/클라이언트 모델을 따르며, 클라이언트에서 요청(Request)을 보내면 서버에서 요청을 처리하고 그것에 응답(Response)하는 구조로 되어있다.

2) 트랜잭션

HTTP는 연결 당 하나의 트랜잭션을 수행한다. 클라이언트가 서버와 연결 후 요청 메시지를 보내고 서버로부터 응답을 받게 되면 연결을 끊어버린다. HTTP문서, 텍스트, 이미지 등 여러 자원이 동시에 필요한 경우에도 자원 하나에 대해서 하나의 연결을 만들게 되는 것이다. 아래의 그림6에서는 HTTP 문서와 이미지 파일 하나를 읽기 위해 두 번의 연결/종료 동작을 보여주며 총 2번의 트랜잭션이 발생하는 것을 보여준다. 이와 같은 방식을 사용하게 동시에 많은 사람들이 웹 서비스를 사용하더라도 접속유지는 최소한으로 할 수 있기 때문에 더 많은 요청을 처리 할 수 있다는 장점이 있지만, 연결을 끊어버리기 때문에 클라이언트의 이전 상태를 알 수 없다는 문제점이 생긴다. 이를 쿠키를 이용하여 해결하고 있다.

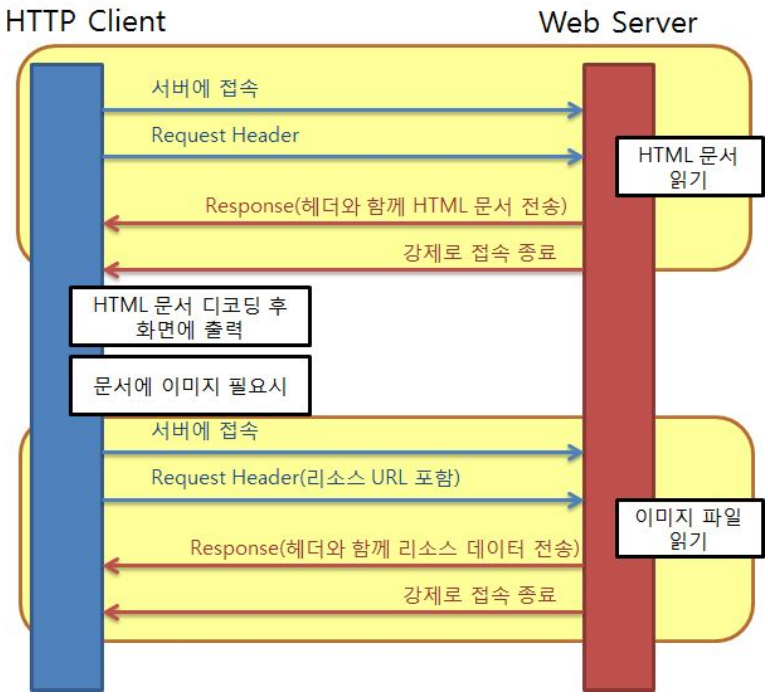

그림 6. HTTP의 트랜잭션

3) URI

URI(통합 자원 식별자 : Uniform Resource Identifier)는 인터넷상의 정보자원에 대한 식별체계로서 인터넷 주소체계인 URL과 고유일음 체계인 URN을 총칭하는 개념으로서 문서, 이미지, 음악파일, 동영상 등 다양한 정보자원에 대한 접근에 있어서 유일성을 부여하고 식별을 가능케 하는 관리체계를 의미한다.(오상훈, 2001) URI는 HTTP와는 독립된 다른 체계로 오로지 자원의 위치를 알려주기 위한 프로토콜이라 할 수 있다.

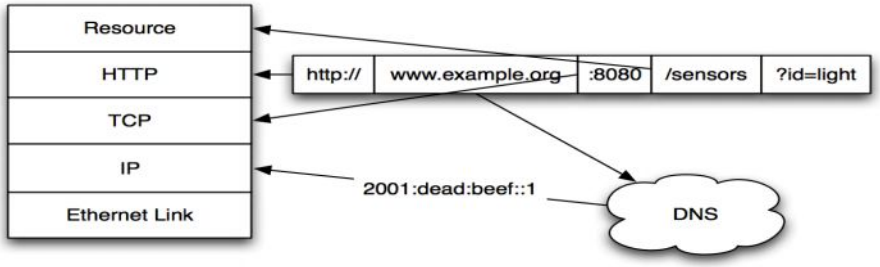

그림 7. URI 구조

4) Method

메서드는 요청의 종류를 서버에게 알려주기 위하여 사용한다. 아래의 표3은 요청에 사용할 수 있는 HTTP의 메서드들이다.

표 3. HTTP의 메서드 종류

종 류	설 명
GET	URL에 해당하는 자료의 전송을 요청한다.
POST	서버가 처리할 수 있는 자료를 보낸다.
PUT	해당 URL에 자료를 저장한다.
DELETE	해당 URL의 자료를 삭제한다.
HEAD	GET과 같은 요청이지만, 자료에 대한 정보(meta-information)만을 받는다.
TRACE	이전에 요청한 내용을 들을 것을 요청한다.
OPTIONS	서버가 특정 URL에 대해 어떠한 HTTP Method를 지원하는지 묻는다.
CONNECT	프록시가 사용하는 요청을 나타낸다.

5) 요청 데이터 포맷

웹 브라우저는 웹 서버에 데이터를 “요청”하는 하나의 “클라이언트 프로그램”이다. 요청은 서버가 인식할 수 있는 약속된 형식인 HTTP 형식을 따라야 한다. 요청 데이터는 “Header”와 “Body”로 구성된다.

헤더에는 요청과 요청 데이터에 대한 메타정보들이 들어있다. 다음은 헤더의 일반적인 모습이다.

표 4. 요청 데이터 포맷

헤더		설명
1	GET /cgi-bin/http_trace.pl HTTP/1.1\r\n	필수 요소로 요청의 제일 처음에 와야 한다. 3개의 필드로 이뤄져 있다. 요청 메서드 : GET, PUT, POST, PUSH, OPTIONS 등의 요청 방식이 온다. 요청 URI : 요청하는 자원의 위치를 명시한다.

		HTTP 프로토콜 버전 : 웹 브라우저가 사용하는 프로토콜 버전이다. (2013년 6월)현재 HTTP 최신 버전은 1.1이다. 웹 브라우저의 프로토콜 버전에 따라서 서버의 지원이 달라집니다. 대부분의 웹 서버가 1.1을 지원하고 있으니, 1.1만 사용한다고 봐도 무리는 없다. 1.1은 1.0에 keep-alive, 데이터 압축 등의 기능을 추가 지원한다.
2	ACCEPT_ENCODING: gzip,deflate,sdch\r\n	지원 인코딩 : HTTP 1.1부터 웹 서버는 네트워크 대역폭을 아끼기 위해서 데이터 압축 기능을 제공한다. 웹 브라우저가 지원하는 인코딩을 적어 보내면, 웹 서버는 이 중 지정된 방식으로 데이터를 인코딩한다. 만약 Proxy 서버를 개발한다면, 인코딩 부분을 삭제하는게 좋다. Proxy 서버는 웹 문서의 본문을 수정해야 하는 경우가 있을 수 있는데, 문서가 압축돼 오면 압축을 풀어야하기 때문에 굳이 압축할 필요가 없다.
3	CONNECTION: keep-alive\r\n	연결 방식 : HTTP 1.1은 연결유지 (persistent connection)기능을 지원한다. 기본적으로 HTTP는 하나의 요청 당 하나의 연결을 맺는 방식이다. 하나의 웹 페이지에 이미지를 포함해서 10여개의 객체가 있다면 10번 연결을 맺었다 끊어야 한다. 소켓 통신에서 연결은 매우 비용이 많이 들어가는 연산으로 비효율적인 방식이다. keep-alive를 사용하면 하나의 연결에 여러 요청을 보낼 수 있다.
4	ACCEPT: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n	ACCEPT : 클라이언트가 지원하는 미디어 타입
5	ACCEPT_CHARSET: windows-949,utf-8;q=0.7,*;q=0.3\r\n	ACCEPT_CHARSET : 클라이언트가 지원하는 문자 셋
6	USER_AGENT: Mozilla/5.0 (X11; Linux i686) AppleWebKit/535.1 (KHTML, like Gecko) Chrome/13.0.782.24\r\n	USER_AGENT : 클라이언트의 버전, 운영체제 등을 명시한다. 웹 서버는 이 정보를 읽어서 클라이언트와 운영체제에 맞게 정보를 수정할 수 있다. 예컨데 모바일기기로 접근할 경우, 서버는 이 정보를 읽어서 모바일 환경에 맞게 데이터를 조정해서 보낼 수 있다.
7	ACCEPT_LANGUAGE: ko-KR,ko;q=0.8,en-US;q=0.6,en;q=0.4\rn	ACCEPT_LANGUAGE : 지원하는 언어 종류를 나타낸다.
8	HOST: www.ex.co.kr\r\n	HOST : 정보를 요청하는 서버의 주소를 적는다. 하나의 서버가 여러 도메인을 가지고 서비스 할 수 있기 때문에 필요한 필로, HTTP 1.1에는 반드시 필요합니다. 웹 서버에 따라서는 HOST가 없을 경우 기본 도메인 서버를 연결해 주기도 한다.
9	\r\n	헤더 종료 : 줄 처음에 \r\n을 명시해서, 헤더가 끝났다는 것을 서버에게 알려준다. 이후에 body가 들어간다.

6) 응답 헤더 포맷

응답 헤더는 서버의 여러 상태 정보를 포함하기 때문에, 꽤 복잡해질 수 있다.

표 5. 응답 헤더 포맷

	헤더	설명
1	HTTP/1.1 200 OK\r\n	프로토콜과 응답코드 : 반드시 첫줄에 와야 한다. 3개의 필드로 구성돼 있다. HTTP/1.1 : 응답 프로토콜과 버전 200 : 응답 코드 OK : 응답 메시지. Not Found, Internal Server Error 등의 메시지다.
2	Date: Fri, 08 Jul 2013 00:59:41 GMT\r\n	날짜를 나타낸다.
3	Server: Apache/2.2.4 (Unix) PHP/5.2.0\r\n	서버 프로그램및 스크립트 정보.
4	X-Powered-By: PHP/5.2.0\r\n	응답헤더에는 다양한 정보를 추가할 수가 있다. 어떤 정보를 추가할지는 사실 개발자 마음이다. HTTP 기반의 서버/클라이언트 제품을 만든다면, 헤더에 어플리케이션 정보를 추가해서 사용하면 된다.
5	Expires: Mon, 26 Jul 1997 05:00:00 GMT\r\n	
6	Last-Modified: Fri, 08 Jul 2013 00:59:41 GMT\r\n	컨텐츠의 마지막 수정일을 나타낸다.
7	Cache-Control: no-store, no-cache, must-revalidate\r\n	캐쉬 제어 방식.
8	Content-Length: 102\r\n	컨텐츠 길이.
9	Keep-Alive: timeout=15, max=100\r\n	Keep Alive기능 설정이다. keep alive는 클라이언트측에 연결을 유지하라는 신호를 보내기 위해서 사용한다. 그러면 클라이언트는 최대 timeout에 지정된 시간동안 연결을 유지한다 . 이 시간동안 클라이언트는 이미 맺어진 연결로 요청을 계속 보낼 수 있다 . Keep Alive는 따로 자세히 설명한다.
10	Connection: Keep-Alive\r\n	
11	Content-Type: text/html\r\n	Content-Type. 응답에 실어 보내는 컨텐츠가 HTML 문서인지, 이미지인지, CSS, JavaScript인지 혹은 다른 어플리케이션 형태인지를 알려준다. 웹 어플리케이션들은 Content-Type에 따라서 Body의 데이터를 어떻게 읽을지를 결정한다. 따라서 전송 데이터에 맞는 content-type를 명시해야 한다.
12	\r\n	헤더 종료

7) 응답 코드

클라이언트가 서버에 접속하여 어떠한 요청을 하면, 서버는 세 자리 수로 된 응답 코드와 함께 응답한다. 응답 코드에서 1로 시작하는 코드는 정보들을 교환하기 위한 코드이고 2로 시작하는 코드는 성공 메시지, 3은는 변경된 데이터에 대한 메시지, 4는 클라이언트 오류, 5는 서버 오류들을 나타낸다. 구체적인 HTTP의 응답 코드는 다음 표6과 같다.

표 6. 응답 코드 종류

코드	메시지	설명
1XX	Informational(정보)	정보 교환.
100	Continue	클라이언트로부터 일부 요청을 받았으니 나머지 요청 정보를 계속 보내주길 바람.
101	Switching Protocols	서버는 클라이언트의 요청대로 Upgrade 헤더를 따라 다른 프로토콜로 바꿀 것임.
2XX	Success(성공)	데이터 전송이 성공적으로 이루어졌거나, 이해되었거나, 수락되었음.
200	OK	오류 없이 전송 성공.
	...	
3XX	Redirection(방향 바꿈)	자료의 위치가 바뀌었음.
300	Multiple Choices	최근에 옮겨진 데이터를 요청.
301	Moved Permanently	요구한 데이터를 변경된 URL에서 찾았음.
302	Moved Permanently	요구한 데이터가 변경된 URL에 있음을 명시.
	...	
4XX	Client Error(클라이언트 오류)	클라이언트 측의 오류. 주소를 잘못 입력하였거나 요청이 잘못 되었음.
400	Bad Request	요청 실패. 문법상 오류가 있어서 서버가 요청사항을 이해하지 못함,
401.1	Unauthorized	권한 없음 (접속실패).
	...	
5XX	Server Error(서버 오류)	서버 측의 오류로 올바른 요청을 처리할 수 없음.
500	Internal Server Error	서버 내부 오류.
	...	

8) Keep Alive

HTTP는 하나의 연결에 하나의 요청을 하는 것을 기준으로 설계가 됐다. 요즘 웹 서비스의 경우 간단한 페이지라고 해도 이미지, 문서, css, javascript 등 수십 개의 데이터가 있기 마련인데, HTTP의 원래 규격대로라면 웹페이지를 하나 표시하기 위해서 연결을 맺고 끊는 과정을 수십 번을 반복해야 한다. 당연히 비효율적일 수밖에 없다. 연결을 맺고 끊는 것은 TCP 통신 과정에서 가장 많은 비용이 소비되는 작업이기 때문이다.

20여개 정도의 이미지, css, javascript 파일이 있는 HTML로 표현된 문서를 다운받을 때 Keep alive를 지원하지 않을 경우 다음과 같은 과정을 거친다.

- 웹 서버에 연결한다.
- HTML 문서를 다운로드 한다.
- 웹 서버 연결을 끊는다.
- HTML 문서의 image, css, javascript 링크들을 읽어서 다운로드해야할 경로를 저장한다.
- 웹 서버에 연결한다.
- 첫번째 이미지를 다운로드 한다.
- 연결을 끊는다.
- 웹 서버에 연결한다.
- 두번째 이미지를 다운로드 한다.
- 연결을 끊는다.
- 모든 링크를 다운로드 할 때까지 반복한다.

Keep-alive 설정을 하면, 지정된 시간동안 연결을 끊지 않고 요청을 계속해서 보낼 수 있다.

- 웹 서버에 연결한다.
- HTML 문서를 다운로드 한다.
- Image, css, javascript 들을 다운로드 한다.
- 모든 문서를 다운로드 받았다면 연결을 끊는다.
-

keep-alive 기능을 사용함으로써 작업이 훨씬 더 간단해 짐을 볼 수 있다. keep-alive 기능은 HTTP 1.1부터 지원한다.

9) 세션 관리

HTTP를 기반으로 하는 웹 서비스는 connectless 방식으로 이전의 상태정보를 유지할 수 없다. 결국 클라이언트의 상태정보를 정보조각 형태로 서버와 클라이언트(웹 브라우저)에 남기는 방식으로 문제를 해결한다. 이 정보조각을 쿠키(cookie)라고 한다. 서버는 쿠키를 이용해서 세션정보를 관리한다. 세션과 쿠키 관계를 정리하자면 서버는 클라이언트와 쿠키를 주고받는 것으로 상태를 확인할 수 있다. 클라이언트가 로그인을 한 상태인지 서버는 쿠키를 키로 하는 값을 데이터베이스에 저장하는 방식으로 세션을 유지한다던지 로그인한 클라이언트가 "물건 - A"를 구매했다면, 클라이언트의 쿠키 id를 키로 데이터베이스에 구매정보를 update 하는 방법으로 유지 할 수 있는 것이다 .

3. CoAP(Constrained Application Protocol)

저전력 손실 네트워크 환경에서는 다양한 프로토콜을 사용하기에는 맞지 않아 적절한 제한이 필요하며, 인터넷 어플리케이션의 대부분은 웹 인터페이스를 통해 이루어지는 것을 감안할 때 웹에서 지원되는 표현적인 상태 전이 구조(Representational State Transfer Architecture)에 순응하는 어플리케이션 계층의 호환성이 요구된다.

따라서 HTTP 그대로는 사용이 어렵고 적응 화된 방안이 요구되는데, 이러한 문제에 착안하여 IETF CORE 워킹 그룹은 Constrained Application Protocol의 약자로 제한적인 환경에서 Restful 구조를 지원하기 위한 프로토콜의 상세 방안을 제한적인 어플리케이션 프로토콜 CoAP이라는 이름으로 정의하고 있다.

CoAP은 UDP와 응용계층 사이에서 추가 레이어로 웹을 지원하기 위한 기술이다. UDP는 가볍고 신뢰도가 낮은 프로토콜이라서 메시지 전송 신뢰도 향상을 위해 추가적인 요청/응답 구조를 CoAP 추상레이어에서 구현하며, 신뢰성 있는 메시지를 전송하기 위해 CoAP는 신뢰성 메시지(CON), 비 신뢰성 메시지(NON), 응답(ACK), 재요청(RST) 총4가지형태의 메시지를 정의 한다.

센서 노드와 같이 경량의 소프트웨어만 사용해야 할 경우에 웹서비스를 지원하는 HTTP프로토콜은 TCP레이어 위에서 동작하므로 너무 무겁기 때문에 CoAP 레이어에서 UDP를 사용해 가벼우면서도 신뢰성을 보장하는 프로토콜을 구현한다.

1) 작동 방식

(가) 메시지 모델

■ 신뢰성 있는 메시지

신뢰성 있는 메시지를 전달하기 위해서 CoAP는 CON 메시지를 사용하며, CON 메시지는 UDP 패킷 위에 추가적인 CoAP 헤더를 두어 메시지를 전송한다. CON 메시지를 받은 대상은 ACK를 이용하여 메시지가 잘 전달되었는지 알려준다.

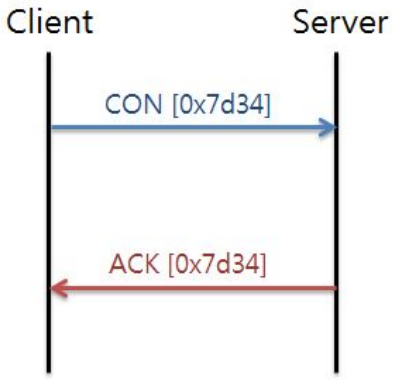

그림 8. 신뢰성 있는 메시지 전송

■ 비 신뢰성 메시지

비 신뢰 메시지 전달의 경우 NON 메시지를 통해 전송한다. NON 메시지를 받은 대상은 ACK 메시지를 전달할 필요가 없다.

그림 9. 비신뢰성 메시지 전송

(나) 요청/응답 모델

CoAP 요청 및 응답 의미는 메소드 코드 또는 응답 코드 각각이 CoAP 메시지에 포함되어 있다. 그러나 URI 및 페이로드 미디어 유형 등의 옵션 요청 및 응답 정보는 CoAP 옵션에 해당한다. 토큰은 기본 메시지에서 독립적으로 요청에 대한 응답을 일치시키기 위해 사용되는 것으로 메시지 ID와는 다른 개념이다.

CON 또는 NON메시지를 받고 즉시 응답이 가능하면 응답 ACK메시지에 응답 결과를 담아 전송할 수 있다. 이를 piggy-backed 응답이라고 한다.

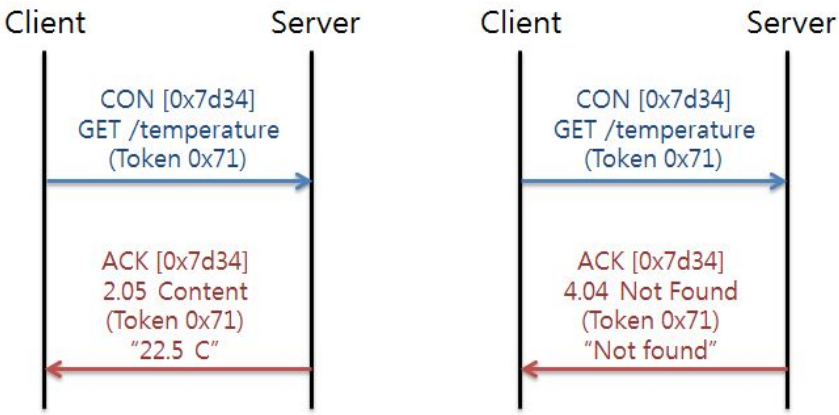

그림 10. GET 요청에 따른 piggy-backed 응답

CON 메시지로 옮겨진 요구에 서버가 즉시 응답 할 수 없는 경우가 있다. 센서 노드가 슬립 상태라 그럴 수도 있고, 다른 요청을 처리하느라 바로 응답 할 수 없는 경우도 있다. 이런 경우 클라이언트가 요청을 재전송 또는 중단 할 수 있도록 단순한 빈 메시지로 응답한다. 그 후 응답의 준비가 되면 서버는 새로운 CON 메시지를 전송하게 된다. 해당 과정을 그림11에서 보여준다.

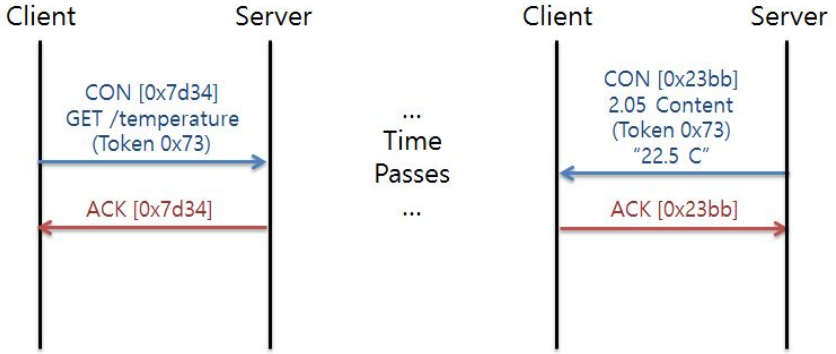

그림 11. GET 요청에 따른 별도의 응답

CON요청과 더불어 NON요청으로도 자원을 요청 할 수 있다. NON메시지에 GET요청을 담아 전송하면 되는데, 서버는 같은 NON메시지를 사용하여 응답한다.

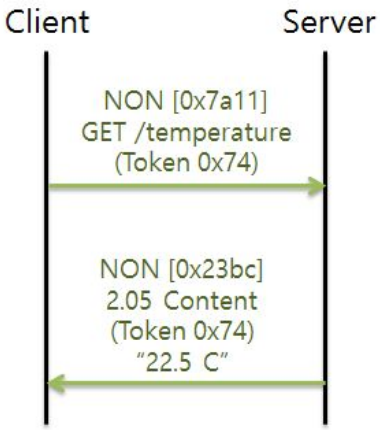

그림 12. NON 요청에 따른 응답

2) 메시지 포맷

(가) 헤더 포맷

CoAP 메시지는 2진 포맷으로 인코딩되며, 타입-길이-값(TLV) 포맷을 따라 헤더가 고정된 크기를 가지고 만들어진다. 프레임 내에 정의된 필드는 버전, 메시지는 기본 4바이트 고정 헤더를 포함하게 되고 필요에 따라 토큰, 옵션, 페이로드 데이터들이 추가 된다.

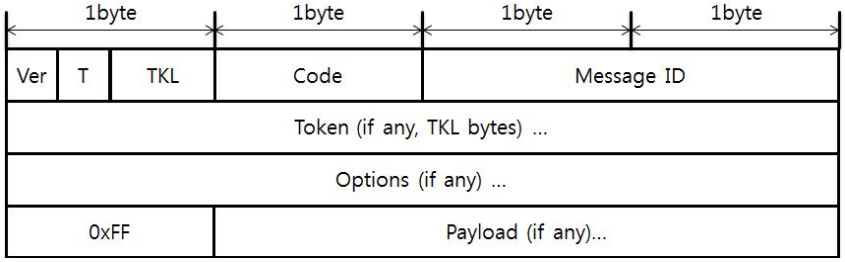

그림 13. 메시지 포맷

각 필드는 다음과 같이 정의된다.

- Version(Ver) : 2비트 부호 없는 정수. CoAP의 버전을 나타내는 것으로 현재 CoAP 버전에서는 01 값을 가져야 한다. 그 외의 값들은 이후의 버전을 위해 예약되어 있으며, 알 수 없는 버전 값을 가지는 메시지는 자동으로 무시해야 한다.
- Type(T) : 2비트 부호 없는 정수. 메시지의 타입을 표현하는 것으로 신뢰성 메시지(0), 비 신뢰성 메시지(1), 응답 메시지(2), 리셋 메시지(3) 이렇게 총 4가지의 타입을 표현한다.
- Token Length(TKL) : 4비트 부호 없는 정수. 뒤 따라오는 토큰의 길이를 나타내며 0에서 8사이의 값을 가진다. 9~15 사이의 값은 사용하지 않으며 해당 값이 들어오는 경우 메시지 포맷 오류로 처리해야 한다.
- Code : 8비트 부호 없는 정수. 앞의 3비트는 클래스(class)를 나타내고 뒤의 5비트는 자세한 내용(detail)을 나타낸다. 클래스 값은 0~7의 값이 올 수 있으며 각각 요청(0), 응답 성공(2), 클라이언트 에러 응답(4), 서버 에러 응답(5)의 값들을 나타낸다. 클래스 값과 자세한 내용을 합하여 c.dd라는 표현을 사용한다.
- Message ID : 16비트 부호 없는 정수. 각 메시지에 대한 중복 및 확인을 위한 ID 값으로, 신뢰성/비 신뢰성 메시지에 대한 응답 또는 리셋 메시지를 보내는 데에 사용된다.
- Token : TKL 만큼의 가변 길이를 가지며 메시지 내용에 대한 구분 식별자를 표현한다.
- Options : 옵션 포맷에 따른 형태로 옵션 값을 표현한다.
- Payload marker : 옵션 값 뒤에 위치하며 0xFF의 값을 가진다.
- Payload : 페이로드가 존재하는 경우 페이로드 마커 뒤에 위치하여 패킷의 데이터그램 끝까지 표시된다.

(나) 옵션 포맷

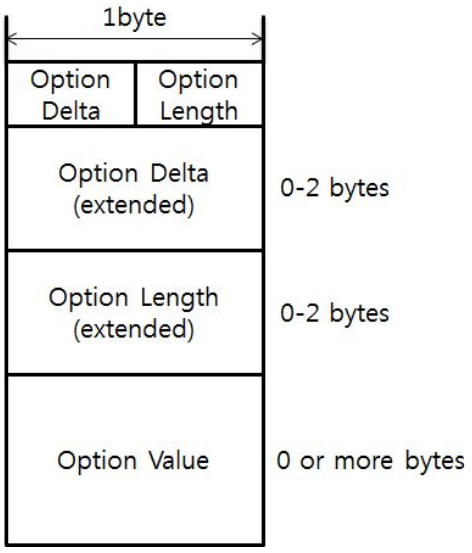

그림 14 옵션 포맷

옵션은 그림14와 같은 구조를 가지며 각 필드는 다음과 같이 정의된다.

- Option Delta : 4비트 부호없는 정수. 0에서 12까지의 값으로 옵션 델타를 나타낸다. 나머지 3가지의 값은 예약되어 있다.
  - 13 : 8비트의 옵션 델타 확장 필드를 사용하며, 옵션 델타 값은 옵션 델타 확장 필드의 값에서 13을 더한 값이 된다.
  - 14 : 16비트의 옵션 델타 확장 필드를 사용하며, 옵션 델타 값은 옵션 델타 확장 필드의 값에서 269를 더한 값이다.
  - 15 : 페이로드 마커를 나타낸다.
- Option Length : 4비트 부호 없는 정수. 옵션 값의 길이를 나타내며 옵션 델타와 마찬가지로 0에서 12사이의 값을 가지고 나머지 3가지의 값은 예약되어 있다.
  - 13 : 8비트의 옵션 길이 확장 필드를 사용하며, 옵션 길이 값은 옵션 길이 확장 필드의 값에서 13을 더한 값이 된다.
  - 14 : 16비트의 옵션 길이 확장 필드를 사용하며, 옵션 길이 값은 옵션 길이 확장 필드의 값에서 269를 더한 값이다.
  - 15 : 향후 사용을 위해 예약되어 있다. 현재 이 값이 설정되어 있으면 메시지 형식 오류로 처리해야 한다.

3) 메시지 전송

CoAP 메시지는 EndPoint 간에 비동기적으로 전송된다. CoAP이 UDP와 같은 비 신뢰성 전송을 사용하는 것으로 인해 메시지가 중복 표시 되거나 메시지 자체가 손실되거나 정해진 순서 없이 도착 할 수도 있다. CoAP에서는 이러한 문제를 해결하기 위해 경량화 된 신뢰성 메커니즘을 구현한다.

메시지에는 다양한 종류가 존재하며 CoAP 헤더의 Type 필드에서 지정된다. 메시지 유형과는 별도로 요청(Request), 응답(Response) 또는 빈 메시지(Empty)일 수 있으며 메시지 유형별로 사용 가능 메시지 종류를 표7에 정리하고 있다. 빈 메시지는 Code 필드가 0.00으로 설정되어 있다. 토큰 길이 필드가 0으로 설정되어야 하고 메시지 ID 필드 뒤에 다른 데이터가 존재해서는 안 된다. 그러한 경우 포맷 에러로 처리되어야 한다.

표 7. 메시지 타입별 사용용도

	CON	NON	ACK	RST
Request	O	O	X	X
Response	O	O	O	X
Empty	*	X	O	O

ACK 또는 RST 메시지는 해당 엔드 포인트의 추가 주소 정보와 함께 메시지 ID를 통해 CON 또는 NON 메시지와 관련이 있다. 메시지 ID는 CON 또는 NON 메시지의 CoAP 헤더에 포함된 16비트 unsigned integer이다. 메시지 ID는 수신자가 ACK 또는 RST 메시지로 반응해야 한다. 동일한 메시지 ID는 EXCHANGE_LIFETIME 내에서 재사용(같은 endpoint와 통신 중에)하면 안 된다.

메시지의 전송은 다음 표8의 매개변수들로 인해 제어된다.

표 8. 메시지 매개변수

이름	기본 값
ACK_TIMEOUT	2 초
ACK_RANDOM_FACTOR	1.5
MAX_RETRANSMIT	4
NSTART	1
DEFAULT_LEISURE	5 초
PROBING_RATE	1 Byte/초

4) URI

CoAP URI는 HTTP URI와 매우 유사한 특징을 갖는다. "coap" URI는 CoAP 자원과 자원들을 위치시키기 위한 방법들을 나타낸다.

5) Proxy

CoAP은 HTTP의 몇몇 기능들이 구현되어 HTTP와 직접적인 맵핑이 가능하다. 또한 CoAP은 세션 초기화 프로토콜인 SIP, 확장 가능한 메세징 및 상태 프로토콜인 XMPP와도 맵핑이 가능하다.

CoAP 맵핑은 CoAP 클라이언트로 하여금 HTTP 서버의 자원에 접근할 수 있도록 하며, 이는 CoAP 요청 메시지에 Proxy-Uri 옵션을 활용하여 "http" URI를 넣고 CoAp-HTTP 프록시로 보내거나 혹은 Proxy-Uri 옵션 없이 CoAP 요청을 CoAP과 HTTP를 맵핑하는 역방향 프록시(reverse proxy)에 전달하는 방법이 있다.

이와는 반대로, HTTP 클라이언트가 CoAP 서버의 자원에 접근할 수 도 있다. 구체적으로는, HTTP 요청 메시지의 Request-Line에 "coap" URI를 넣어서 HTTP-CoAP 프록시에 보내거나 HTTP 요청을 HTTP와 CoAP을 맵핑하는 역방향 프록시에 보내도록 하는 방법이 있다.

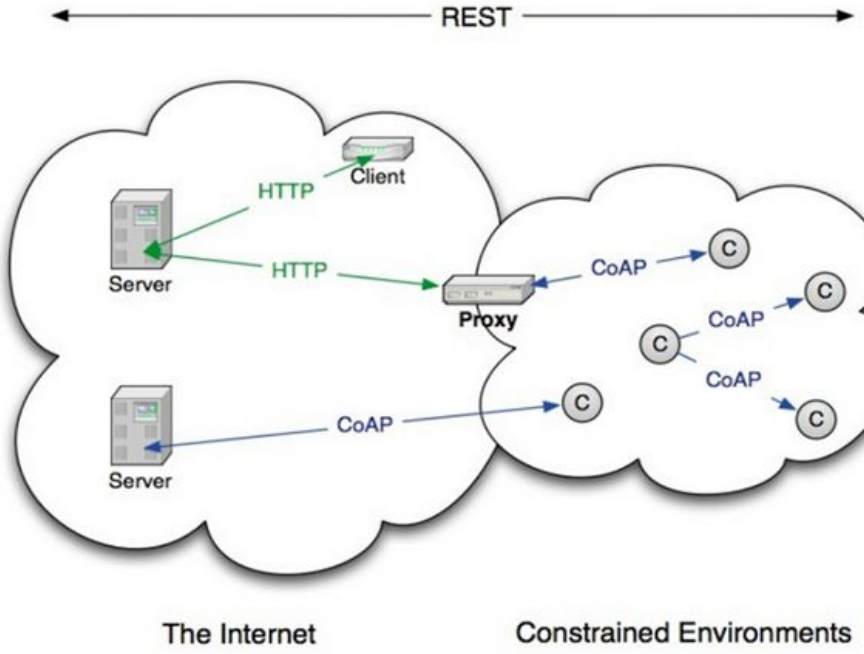

그림 15. CoAP Proxy(CoRE WG, 2010)

III. 시스템 설계 내용

본 논문에서는 Non-IP 프로토콜인 ZigBee를 사용하는 무선 센서 네트워크 상에서 CoAP 사용을 위한 시스템을 설계하고 구현하였다. 그림 16은 제안된 ZigBee 기반 네트워크상에서 CoAP 사용을 위한 전체적인 시스템 구조를 나타낸다.

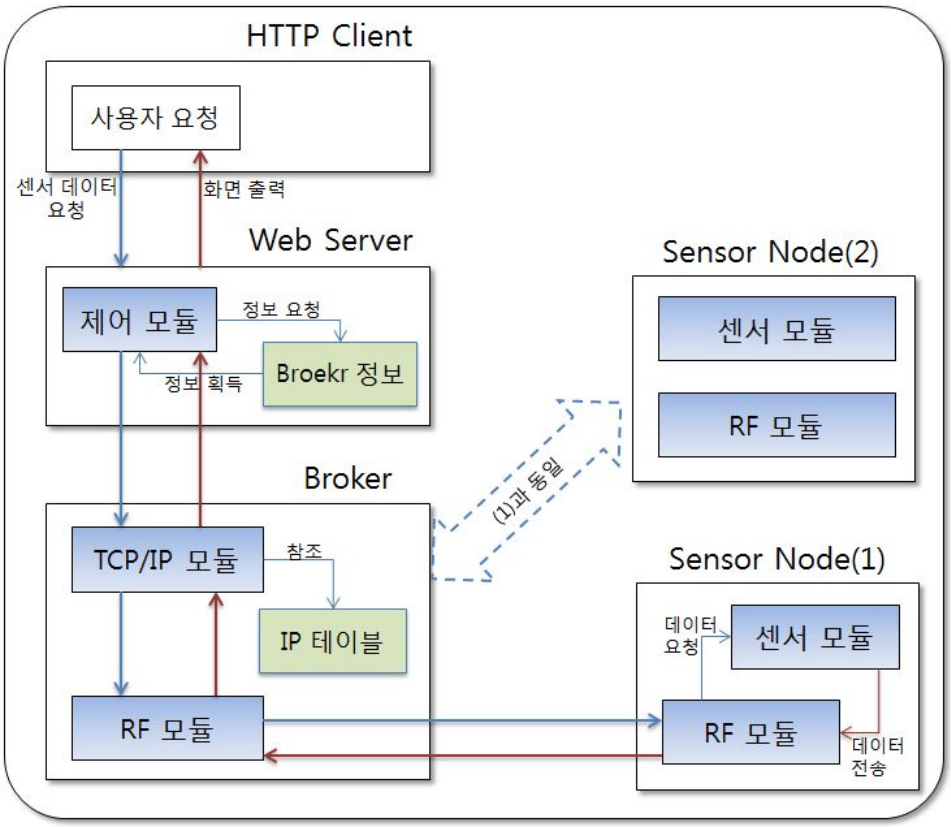

그림 16. 제안된 ZigBee 기반 CoAP 사용을 위한 시스템 구조

센서 노드들은 기본적으로 센서 역할을 하는 센서 모듈과 ZigBee 통신을 위한 RF(IEEE 802.15.4)모듈로 구성되어 있다. 특성에 따라 한 대가 아닌 여러 대가 존재할 수 있으며 각 노드들은 브로커(Broker)라고 불리는 게이트웨어와 각각 연결된다. 각 센서 노드들의 전원이 공급되면 ZigBee 상에서 주소 0x0000을 가지는 ZigBee 코디네이터(Coordinator)로 접속을 하게 되고 연결된 브로커와 1:N 통신을 하게 된다.

브로커는 게이트웨이 역할을 하는 디바이스로 ZigBee 통신을 통해 CoAP을 사용 가능하게 해줄 수 있는 결정적인 역할을 하게 된다. IP테이블을 이용하여 자신에게 접

속한 각 노드들의 주소와 IP 주소와의 매칭을 통해 인터넷과 연결이 되도록 하는 게 주된 역할이다. 센서 노드들과의 통신을 위해 RF 모듈을 가지고 있고 마찬가지로 웹서버와의 통신을 위해 TCP/IP 모듈도 가지고 있어 다중통신 가능하도록 설계 되었다. 브로커는 자신에게 접속해 있는 센서 노드들에 대한 특징들을 알고 있고 웹서버를 통해 특정 명령이 내려왔을 때 가능한 명령인지 판단하여 센서 노드들에게 CoAP 패킷에 맞춰 요청 메시지를 보내거나 웹서버로 바로 특정 응답을 취하게 된다.

웹서버는 일반적인 PC에서 동작하게 되고 사용자와 브로커 들의 연결을 도와주는 역할을 한다. 변하지 않는 특정 고유의 IP 주소를 가지고 있고 각 브로커 들은 웹서버로 Broker자신과 연결되어 있는 노드들에 대한 자세한 정보를 알려주어야 한다.

HTTP Client는 쉽게 말해 웹브라우저라고 할 수 있다. 사용자들은 브라우저의 주소 명령을 통해 웹서버로 특정 요청을 하게 되고 웹서버에서는 브로커 정보를 통해 브로커 에게 명령어를 전달하거나 자신이 바로 응답 메시지를 전해준다.

1. CoAP over ZigBee

IEEE 802.15.4를 이용하여 통신하는 노드들에게 CoAP을 적용하기 위해서 일반적으로는 그림17과 같이 802.15.4 레이어 위에 6LowPAN adaptation 레이어와 IPv6, UDP 레이어를 적용한 뒤 CoAP 계층을 적용하는 방식을 사용(박영기, 김영한, 2012)한다. 이와 같은 방법을 사용하기 위해서는 기존의 ZigBee를 사용하던 노드들에게도 ZigBee 스택을 6LowPAN 스택으로 변환 시킨 후 UDP 계층을 추가로 올려 사용해야 하는 번거로움이 있다. 6LowPAN이 IoT 기술에 적용시키기에 분명 매력적인 프로토콜이긴 하지만 아직까지 단순 ZigBee만을 사용하는 제품이 많이 존재(Jen Sarto, 2013) 한다. 본 논문에서 제안된 시스템을 적용하게 되면 6LowPAN 및 UDP의 구현을 하지 않고 어플리케이션 레벨에서의 구현으로 CoAP을 적용할 수 있다는 장점이 있다.

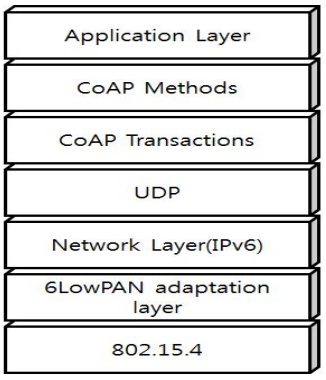

그림 17. CoAP over 6LowPAN

표 9. CoAP over ZigBee의 장단점

종류	장점	단점
ZigBee	○ 기존 ZigBee만을 사용하는 센서 노드들에게 스택 변경 없이 간편하게 적용 가능 - 6LowPAN 구현 불필요 - UDP 구현 불필요 - 어플리케이션 레벨에서 적용 - 많은 ZigBee 제품들 존재	○ 브로커라는 추가적인 디바이스 필요 - 센서 노드 고유 IP 미 할당 - 브로커를 통한 접근만 가능
6LowPAN	○ IoT 기술에 적합합 프로토콜 - IETF CORE 워킹 그룹에서 표준화 작업 중 ○ 각 노드들이 IPv6 가짐	○ 기존 ZigBee를 사용하던 노드들의 추가 작업 필요 - 6LowPAN, UDP 구현 필요 - 어플리케이션의 하위 계층의 변경 필요

2. 메시지 흐름

1) 정상적인 메시지 흐름

HTTP 클라이언트에서 GET 메시지를 요청할 때 브로커의 이름, 노드의 이름, 필요한 정보에 대한 데이터들을 요청한다. 요청을 받은 웹서버에서는 해당 브로커로 노드와 필요 정보를 전달한다. 여기까지는 HTTP 포맷에 맞추어 전달하다가 브로커 에서 센서 노드로 필요 정보를 요청할 때에는 CoAP 포맷에 맞추어 요청하게 된다. 요청을 받은 센서 노드에서는 응답 메시지를 전송하게 되고 요청 때와 역순으로 데이터를 전송해 최종적으로 HTTP 클라이언트에서 받아볼 수 있게 된다. 그림18은 단순한 온도 정보를 받아오기 위한 메시지 흐름을 나타낸 것으로 HTTP 클라이언트로부터 웹서버, 브로커, 센서 노드에 이르기까지의 흐름을 보여주고 있다.

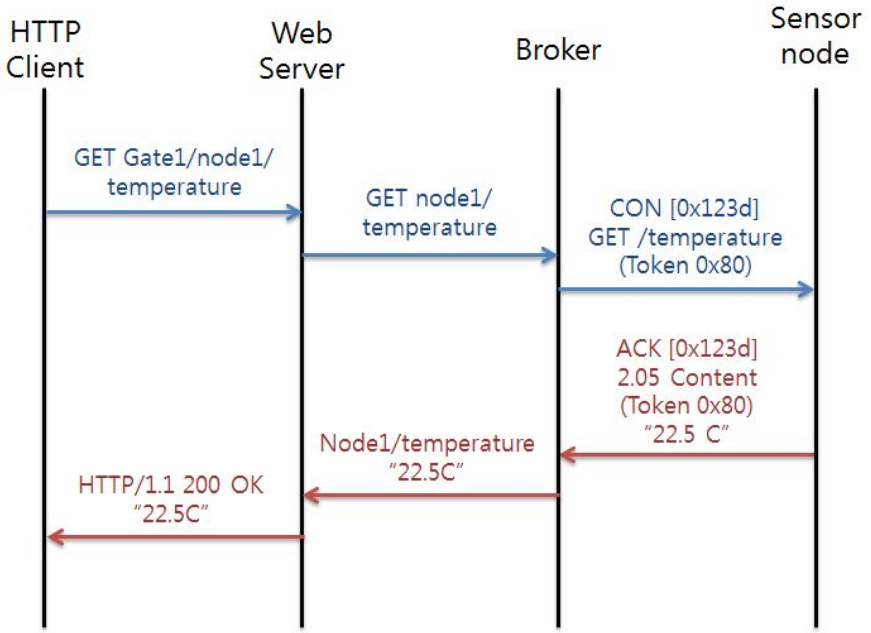

그림 18. 클라이언트로부터 센서 노드에 이르기까지의 메시지 흐름

2) 오류 발생 시 메시지 흐름

사용자가 존재하지 않는 정보를 요구하거나 잘못된 정보를 요구하는 경우 오류가 발생한다. 웹서버와 브로커, 센서 노드 모두 오류를 처리 할 수 있는 기능이 제공되어야 한다. 그림19은 온도 정보를 요구했을 때 센서 노드로부터 관련 정보를 찾을 수 없다는 오류 메시지를 전송 하는 것을 보여준다. 오류메시지 전송을 위해 브로커에서는 일반적인 메시지뿐만 아니라 이런 오류 메시지도 HTTP 프로토콜로 전환 할 수 있어야 한다.

브로커에서 센서 노드들에 대한 구체적인 정보를 모두 가지고 있는 상태에서는 센서 노드까지 요청 메시지가 갈 필요 없이 브로커에서 바로 오류 메시지로 응답 할 수도 있다.

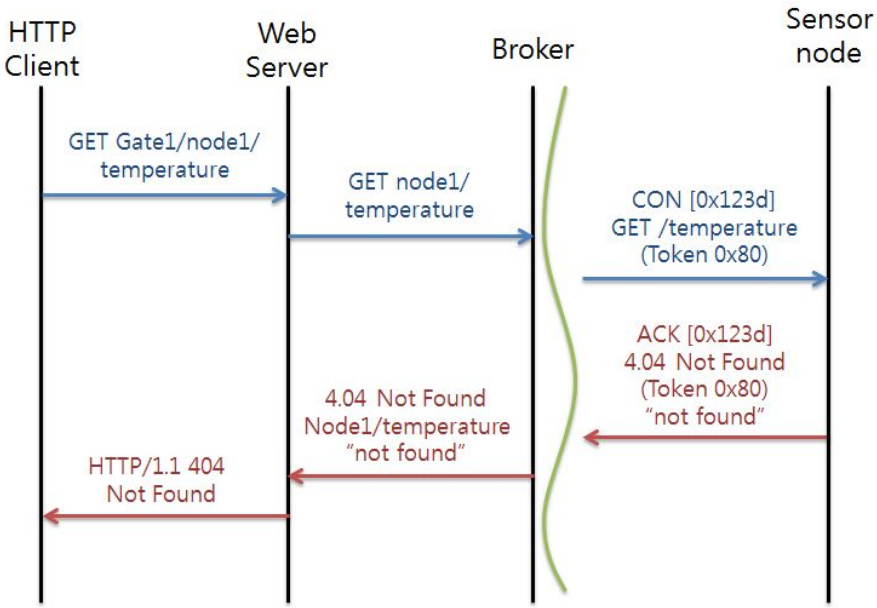

그림 19. 오류 메시지 흐름

3. CoAP 메시지 교환

1) 메시지 ID만을 이용한 데이터 전달

그림20은 브로커와 센서 노드들 간에 GET 메시지 교환의 구체적인 내용을 나타내고 그림21은 메시지 교환 시 실제 패킷 모양을 보여준다. URI의 값은 페이로드에 들어가지 않고 옵션으로 들어가게 된다. 옵션의 11이 Uri-Path를 표시하며 옵션 값으로 “temperature“ 값이 들어간다. 응답 메시지의 코드 값으로 69가 들어가는 것은 2.05(Content)를 나타내며 옵션이 아닌 페이로드에 해당 데이터가 들어간다.

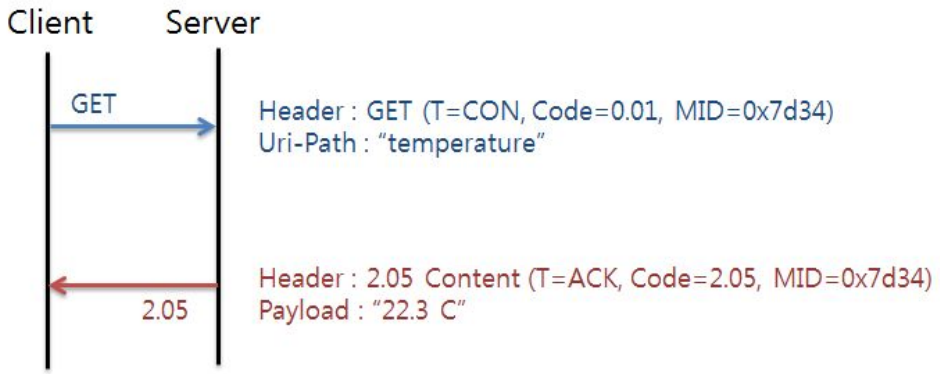

그림 20. 메시지 ID만 이용한 GET 메시지 교환 구체적인 내용

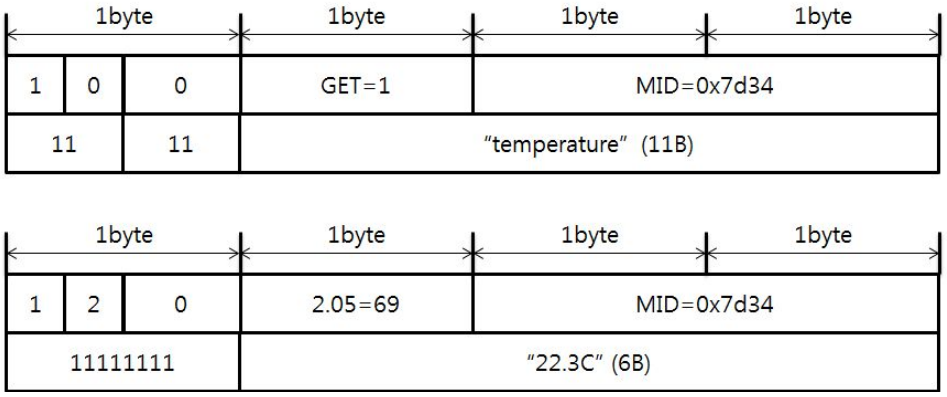

그림 21. 메시지 ID만 이용한 GET 메시지 교환 시 패킷 모양

2) 토큰을 이용한 데이터 전달

앞 절에서는 단순히 메시지 ID만을 이용하여 메시지를 전달하는 모습을 보여줬는데, 일반적으로 응답받아야 할 특정 데이터가 있는 요청을 하는 경우에는 토큰을 함께 보내주는 방식을 사용한다. 그림22가 토큰도 추가로 사용하는 경우의 GET 메시지 교환의 구체적인 내용을 나타내며 그림23은 메시지 교환 시 실제 패킷 모양을 보여준다. 이렇게 바로 피기백 응답을 하는 경우 메시지 ID와 토큰 값 둘 다 같은 값으로 응답하게 되지만 바로 응답하지 못하는 경우에는 같은 메시지 ID로 빈 응답 메시지만 보낸 후 나중에 다른 메시지 ID를 이용하고 이전의 토큰값을 실어 CON 메시지를 새로 보내는 방식을 사용하게 된다.

그림 22. 토큰을 이용한 GET 메시지 교환 구체적인 내용

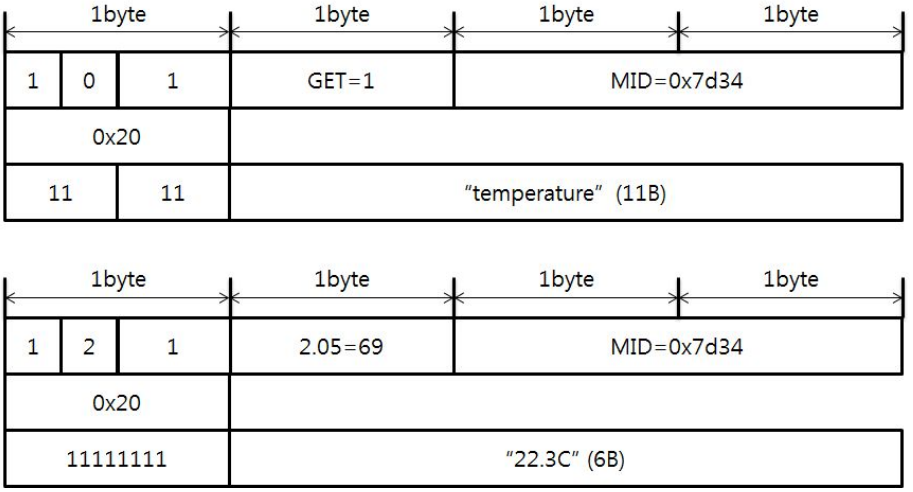

그림 23. 토큰을 이용한 GET 메시지 교환 시 패킷 모양

3) 프로토콜 스택

브로커와 센서 노드들의 프로토콜 스택은 그림24와 같다. 센서 노드는 ZigBee 프로토콜만 사용하기 때문에 ZigBee 스펙에 맞춰 구성되어 있고 거기에 CoAP 사용을 위해 ZigBee 네트워크 레이어 위에 CoAP Transactions과 CoAP Message 층이 자리하고 있다.

브로커는 ZigBee와 TCP/IP 두 가지 프로토콜을 모두 사용해야 하므로 ZigBee 하위 레이어와 CoAP 사용을 위한 UDP 레이어가 자리하고 있다. 프로토콜간의 메시지 변환은 어플리케이션 층에서 이루어진다.

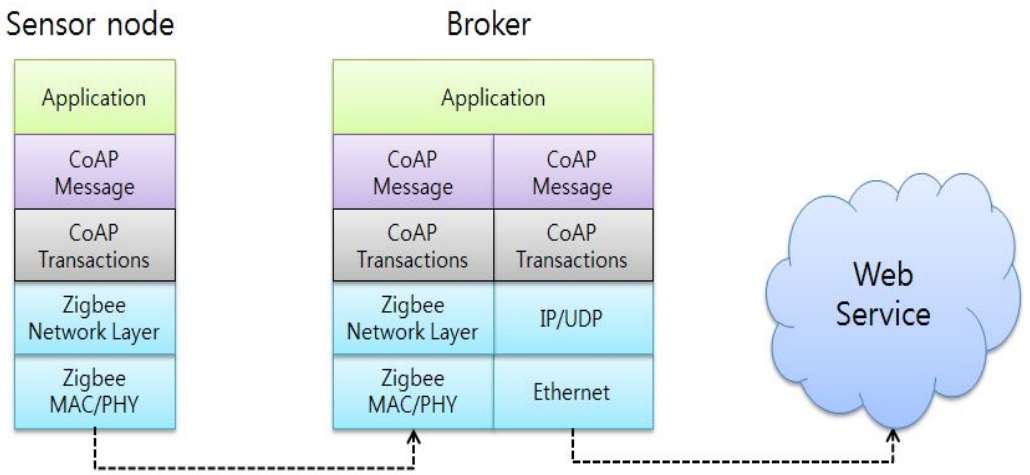

그림 24. Broker와 Sensor Node의 프로토콜 스택

4. 하드웨어 구성

1) Broker 하드웨어

Broker 디바이스는 Hardkernel사의 Odroid-Q라는 제품을 이용하였다. 해당 제품의 구조도는 아래의 그림25와 같고, 하드웨어 스펙은 표10에서 표현하고 있다. Android4.0을 사용하는 제품으로 게이트웨이 역할을 해야 하는 Broker로서 충분한 성능을 가지고 있다.

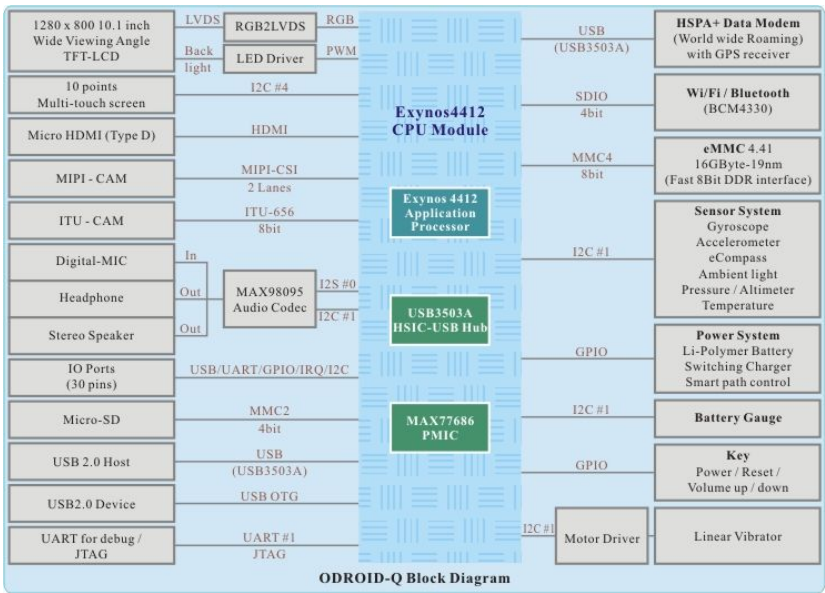

그림 25. Broker 디바이스 하드웨어 구성

표 10. Broker 디바이스의 하드웨어 스펙 (Hardkerner, 2012)

종류	설명
제품명	Odroid-Q
OS	ANdroid 4.0(구글)
MCU	Cortex-A9, 1.4GHz
메모리	1GByte
I/O	USB2.0, UART/JTAG, 30pin port
외부 메모리	16GByte

2) 센서 노드 하드웨어

그림 26는 본 설계에서 사용된 센서 노드의 하드웨어 구조도로 MCU와 각종 커넥터, 저장 장치, 전원 공급 장치로 구성되어 있다. 이 중 RF 인터페이스 커넥트에는 RF 모듈이 장착 될 수 있는데 해당 RF 모듈은 CC2420 칩셋을 사용한다.

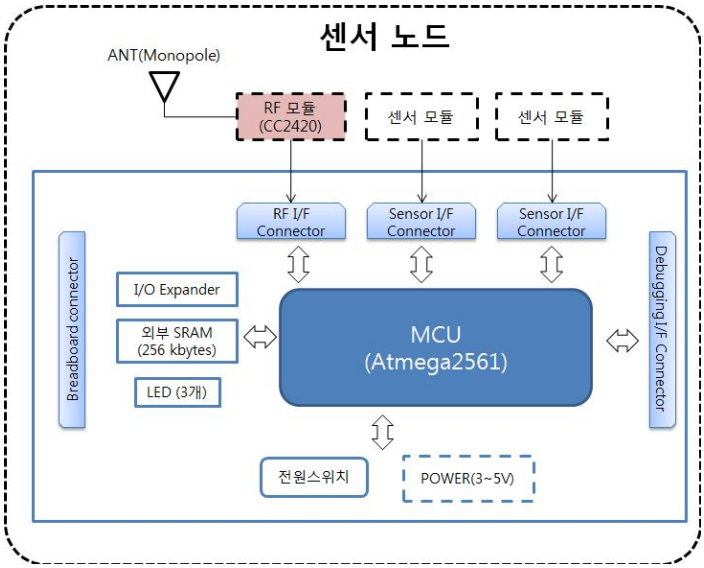

그림 26. 센서 노드 하드웨어 구성

표 11. 센서 노드의 하드웨어 스펙

종류	설명
제품명	AVR2561-DB
OS	Weaver(UTRC)
MCU	Atnega2561, 16MHz
메모리	32KByte
I/O	RF I/F, Sensor I/F, Mini USB
외부 메모리	SRAM 256 KByte

3) CC2420 칩셋

RF 모듈의 주요 칩셋인 CC2420은 TI(Texas Instruments)사에서 개발한 IEEE 802.15.4 방식의 무선 통신을 지원하는 무선 송수신 칩이다. 2.4GHz ISM 대역을 사용하며 센서 네트워크, ZigBee 등 용도로 많이 사용된다.

표 12. CC2420 특징

종 류	설 명
RF	True single-chip 2.4 GHz IEEE 802.15.4 compliant RF transceiver with baseband modem and MAC support
current consumption	RX: 19.7 mA, TX: 17.4 mA
supply voltage	(2.1 - 3.6 V) with integrated voltage regulator (1.6 - 2.0 V) with external voltage regulator
encryption	Hardware MAC encryption (AES-128)

5. 소프트웨어 구성

1) Broker 소프트웨어 구성

본 논문에서 제안된 시스템에서는 브로커가 큰 역할을 차지한다고 할 수 있다. 브로커는 여러 종류의 프로토콜을 동시에 처리할 수 있어야 하며 다수의 디바이스에서 동시 접근이 가능해야 한다. 브로커의 기능에 충실하기 위하여 그림 27과 같은 소프트웨어 구조가 필요하다.

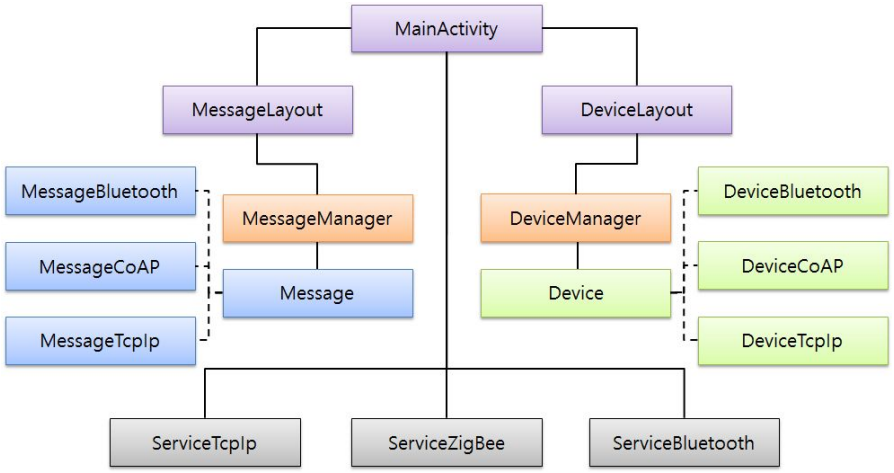

그림 27. Broker 소프트웨어 구조

2) Broker 및 노드 정보 관리

웹 서버에서는 필요한 요청들을 전달 해주고 사용자에게 정보 전달을 위해 브로커에 대한 정보들을 가지고 있을 필요가 있다. 이를 위해서 브로커는 작동이 시작되거나 자신에게 접속된 노드들에 대한 정보의 변화가 생겼을 때 그에 대한 정보를 웹서버로 전달해 줘야 한다.

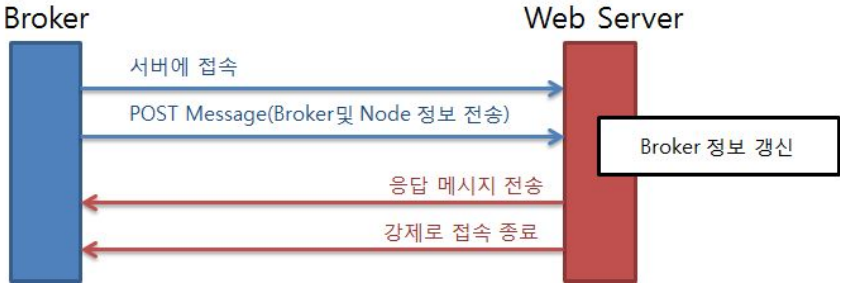

그림 28. Broker와 웹서버간의 정보 교환

IV. 구현 및 성능평가

본 논문에서 설계된 시스템의 구현을 위해 브로커는 Odroid-Q를 사용 하였고, 센서 노드들은 UTRC의 AVR2561 개발 보드(AVR2561-DB)(UTRC, 2013)를 사용 하였다. 브로커 구현을 위해서는 TCP/IP 뿐만 아니라 ZigBee도 사용 가능해야 하는데 Odroid-Q에는 ZigBee 모듈이 장착되어 있지 않았다. 그래서 ZigBee 모듈이 존재하는 AVR2561-DB 노드 하나를 Odroid-Q와 USB 인터페이스를 통해 연결 하여 통신이 가능하게 하였다. ZIgBee 통신은 AVR2561-DB 노드에서 담당하고 나머지 데이터 처리들은 Odroid-Q에서 담당하도록 구현하였다. 아래의 그림29은 그 구조를 나타내며 그림30은 실제 설치 모습을 보여준다.

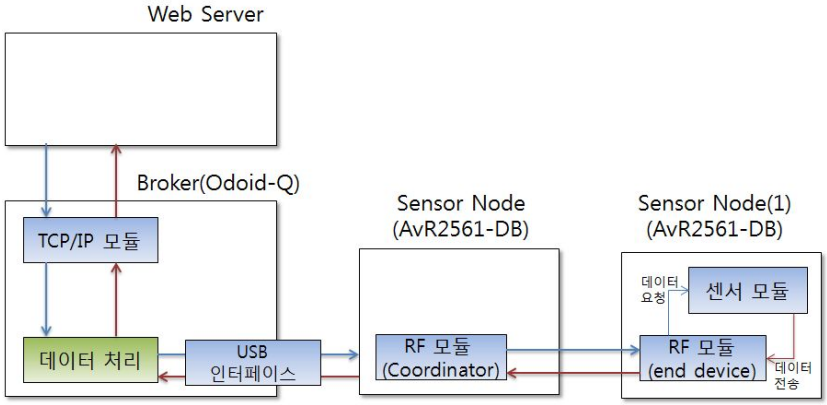

그림 29. Odroid-Q와 AVR2561-DB의 구조도

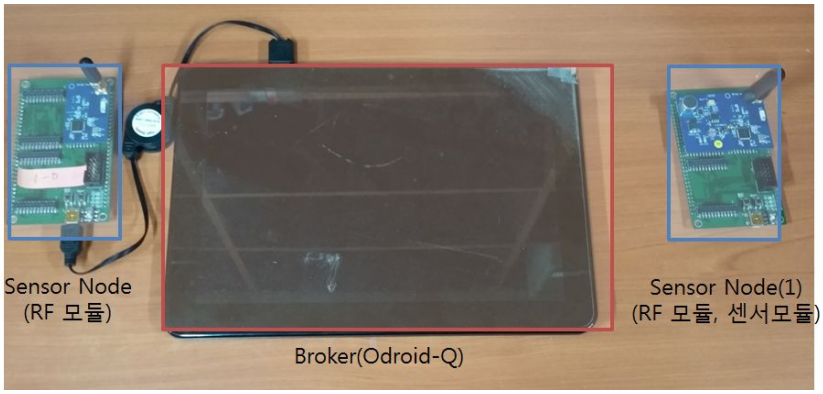

그림 30. 구현된 실제 디바이스 사진

웹서버의 구현은 일반 PC에서 JAVA를 이용하여 단순한 기능만을 수행 할 수 있도록 구현하였다.

1. 제안된 시스템의 CoAP 결과 분석

센서 노드에서 CoAP을 사용하는데 있어서 측정된 전력량은 49mA이다. 이 값은 노드 자체의 전력량인 20mA, RF 모듈의 전력량인 21mA, 센서 모듈의 전력량인 8mA와 합한 값과 같다. 단순 ZigBee으로 하드웨어 스펙 상에 나와 있는 전력량만큼 사용하며 추가로 사용하지 않는 걸 볼 수 있다.

HTTP 클라이언트로 GET 요청을 하여 응답 받을 때까지의 시간은 아래의 그림31과 같다. 3-4-2)절에서 명시한 AVR2561-DB 노드를 이용하여 기존의 6LowPAN을 이용한 방식과 본 논문에서 제시하는 ZigBee를 바로 이용하는 방식에 응답 시간을 총 50번 구하여 평균을 낸 값이다. 6LowPAN을 이용한 기존의 방식으로 CoAP을 적용한 경우 평균 1432ms의 응답시간을, ZigBee를 이용한 제안된 방식으로는 평균 1980ms의 응답시간을 가지는 것으로 측정되어 기존의 방식보다는 500ms정도 느린 결과나 나왔다. 측정된 결과에서 성능이 오히려 더 떨어진 것은 Broker라는 디바이스를 하나 더 거치므로 발생하는 차이로 인해 발생된다. 이렇게 기존의 방식보다 성능은 조금 더 떨어지긴 하지만 사용 중인 ZigBee 노드들에게 바로 적용 가능하다는 큰 장점을 가진다.

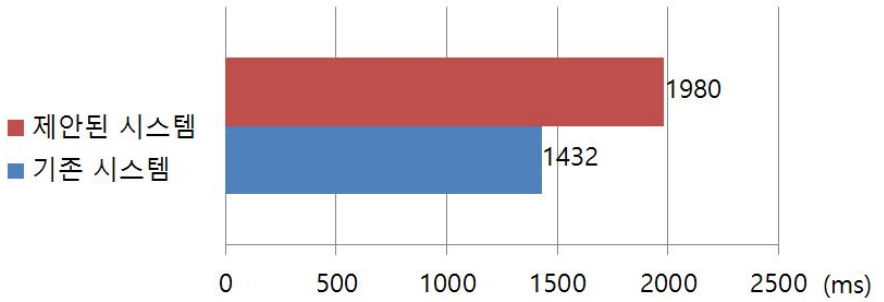

그림 31. 기존 시스템과의 응답시간 비교

V. 결론

사물 인터넷(Internet of Things)에 대한 통신 프로토콜 표준화를 위해 IETF CORE(Constrained RESTful Environments) 워킹 그룹은 UDP의 데이터그램을 이용하여 UDP와 어플리케이션 계층 사이에 새로운 레이어를 추가하여 M2M 노드 간에 통신할 수 있는 CoAP(Constrained Application Protocol)을 중심으로 프로토콜을 개발하고 있다. CoAP은 TCP/IP를 이용하던 HTTP와 달리 UDP를 이용하면서도 신뢰성을 유지시킬 수 있으며, 오버헤드 없이 단순화된 패킷을 이용하고, 복잡한 메시지 전달 절차를 줄이므로 저사양의 노드들에서 적용이 가능하도록 설계되어 있다. UDP를 위해 6LowPAN을 이용하여 구현하도록 되어 있지만 기존의 IEE802.15.4 노드들은 대부분 ZigBee를 사용하고 있어 구현에 어려움이 있다. 본 논문에서는 ZigBee도 UDP와 마찬가지로 데이터그램을 이용한 프로토콜이라는 것을 이용하여, 기존 C로 개발된 CoAP을 센서 노드에 바로 적용할 수 있도록 ZigBee 데이터그램으로 동작 할 수 있는 시스템을 설계하고 구현하였다. 6LowPAN과 달리 IP대신 자체적인 주소를 가지는 ZigBee 프로토콜에서 웹과의 연결을 위해 브로커(Broker)라고 불리는 게이트웨이를 추가적으로 사용하는 것으로 그것을 가능하게 한다.

UTRC의 AVR2561-DB 노드를 이용하여 시스템 구현 후 성능 평가를 하였을 때 전력 소모는 기존의 ZigBee 사용으로 최소한의 전력으로 사용한다는 것을 알 수 있었다. 요청 후 응답 받을 때까지의 시간을 비교해 봤을 때 6LowPAN을 사용하던 기존의 시스템보다 한 단계를 더 거치는 것으로 인해 500ms정도의 성능 저하를 보여주기는 하지만 기존의 노드들에게 쉽게 적용이 가능하다는 장점이 존재한다.

따라서 제안된 시스템을 활용하여 기존의 센서 노드들이 사물 인터넷에 쉽게 적용될 수 있을 것으로 생각된다.

참고 문헌

이대영(2013). 모든 것을 연결하는 사물 인터넷의 모든 것, IDG Tech Report

이근호(2003). 유비쿼터스 AutoID & M2M 개요 및 전망, 인터넷정보학회지 제4권 제2호, pp52-67

문용선, 배영철, 노상현(2010). CC2520 기반의 지그비 모듈 구현에 대한 연구, 한국전자통신학회, 제5권 제6호, pp664-671

박영기, 김영한(2012), CoAP 그룹기술을 이용한 빌딩 내 센서네트워크 설계, 한국컴퓨터종합학술대회 논문집, 제39권 제1호, pp1-3

최미정, 진창규, 김명섭(2011). HTTP 트래픽의 클라이언트측 어플리케이션별 분류, 한국통신학회논문지, 제36권 제11호, pp1277-1284

고석갑, 박일균, 손승철, 이병탁(2013). IETF CoAP 기반 센서 접속 프로토콜 기술 동향, 전자통신동향분석 제28권 제6호, pp133-140

오상훈(2001), URI(Uniform Resource Identifier), TTA journal, 제77호, pp81-85제2호, pp52-67

UTRC(2013). AVR2561-DB, 위버 Wmote 하드웨어 플랫폼 이해와 활용, http://www.utrc.re.kr

UTRC(2013). ZWeaver 이해와 활용, http://www.utrc.re.kr

forrester Research, http://www.forrester.com

Zach Shelby, Carsten Bormann(2009). 6loWPAN The Wireless Embedded Internet, Wiley Series in Communication Networking and Distributed Computing

Zach Shelby(2014). ARM IoT Tutorial

Dimčić, Tomislav, Srđan Krčo, and Nenad Gligorić(2012). CoAP (Constrained Application Protocol) implementation in M2M Environmental Monitoring System, E-society Journal, pp229-234

Shelby, K. Hartke, C. Borman(2013). Constrained Application Protocol (CoAP), draft-ietf-core-coap-18 (work in progress), IETF

CoRE Status Page, Constrained RESTful Environments (Active WG) http://tools.ietf.org/wg/core

CoRE WG(2010). Constrained RESTful Environments WG (core), 79th meeting slides, http://www.ietf.org/proceedings/79/slides/core-0.pdf

Tomislav Dimčić, Srđan Krčo, Nenad Gligorić(2010): ecoBus - Mobile Environment Monitoring, Proc. of ServiceWave 2010, Vol. 6481Springer, pp 189-190, Ghent, Belgium

W3C(2014). drafts -26 of the revised HTTP/1.1 specifications,

http://www.w3.org/Protocols
Hardkernel(2012). Odroid-Q Products.
http://www.hardkernel.com/main/products/prdt_info.php?g_code=G133888637376
Gartner(2013). Top 10 Strategic Technology Trends for 2014,
http://www.gartner.com/technology/research/top-10-technology-trends
Jen Sarto(2013). ZigBee VS 6LoWPAN for Sensor Networks, LS Research,
http://www.lsr.com/white-papers/zigbee-vs-6lowpan-for-sensor-networks
ZigBee Alliance Board of Directors(2008). ZigBee Specification,
http://www.zigbee.org

System Design for usage of CoAp within the Zigbee Network

Jeong, Min geun

School of Computer and Information Engineering
Graduate School, Daegu University
Gyeongbuk Korea

Supervised by prof. Kim, Chang Hoon

(Abstract)

Recently the “Internet of Things” has been an issue. CORE(Constrained RESTful Environments) Walking Group is making a web base protocol called the CoAP (Constrained Application Protocol) for their network environment's limitation in the areas of memory, energy and its performance. CoAP is a technology to support webs by using the UDP protocol's datagram to make an extra layer of level in between the UDP level and the application's level. Therefore, in order to adapt the protocol for the original sensor node, which uses the Zigbee, there must be an additional implementation for the 6LowPAN and the UDP. This paper offers a applicable system to simply adapt the CoAP by implementing application level without changing any stack for the original sensor node which uses the Zigbee.
The propped system, borker device, which acts as the gateway is additionally used to connect the web and original sensor node which uses the Zigbee. In Broker, it is design to multitask while having both Zigbee stack and TCP/IP stack; also, there may be transformation between HTTP packet and CoAP packet. By using the system for the usage of CoAP within the Zigbee Network, as it is proposed in this paper, it is estimated that it will be easier and faster to approach the “Internet of Things.”