



저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:



저작자표시. 귀하는 원저작자를 표시하여야 합니다.



비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.


이것은 [이용허락규약\(Legal Code\)](#)을 이해하기 쉽게 요약한 것입니다.

[Disclaimer](#)

공학석사 학위논문

CoAP 사용을 위한 미들웨어 API 설계

CoAP Middleware API Design



2015년 8월

서울시립대학교 대학원

전자전기컴퓨터공학부

권 학

CoAP 사용을 위한 미들웨어 API 설계

CoAP Middleware API Design

지도교수 나 영 국

이 논문을 공학석사학위 논문으로 제출함.

2015년 6월

서울시립대학교 대학원

전자전기컴퓨터공학부

권 학

권학의 공학석사 학위논문을 인준함.

심사 위원장 김기철 인

심사 위원 김한준 인

심사 위원 나영국 인

2015년 6월

서울시립대학교 대학원

국 문 초 록

TCP/IP 표준을 개발하는 IETF(Internet Engineering Task Force)의 상위 기구인 IAB(Internet Architecture Board)는 작은 사물에도 TCP/IP protocol stack을 붙이기로 결정했다. 이는 하나의 작은 사물(thing)도 하나의 통신 노드로 인정하여, 이를 다섯 개의 전 계층이 올라간 스마트 오브젝트(smart object)로 확장하겠다는 상징적인 의미를 가진다.

이러한 배경 아래서 등장한 것이 인터넷(웹)을 기반으로 사물 간 지능적으로 네트워크를 구성해 통신하는 IoT/WoT(Internet Of Thing /Web Of Thing)이다. IoT환경에서 사물(thing)은 각각 고유한 IP를 가지며, Application계층에서 많은 사람들이 사용하는 웹을 통해 통신하는 스마트 오브젝트이다. IoT에 관한 관심에 따라 사물의 인터넷 통신 프로토콜로 CoAP(Constrained Application Protocol)을 IETF에서 제정하였다.

CoAP는 제한된 환경 내에서 사용할 수 있는 프로토콜로 사물이 인터넷에 연결될 것을 예상하여 만든 표준 규약이다. 이에 따라 본 논문에서는 CoAP 네트워크가 구축된 환경에서 CoAP와 연결되어 관련 사물들의 데이터를 수집하며 관리할 수 있는 Middleware API를 제안하고자 한다.


주요어: CoAP, Constrained Application Protocol, API, Sensor, Network, Protocol, IoT, Internet of Things, Middleware

공학석사 학위논문

CoAP 사용을 위한 미들웨어 **API** 설계

CoAP Middleware API Design

2015년 8월

The seal of The University of Seoul is a circular emblem. It features a central shield with a stylized building or monument. The words "THE UNIVERSITY OF SEOUL" are written in a circular path around the top half of the shield, and the Korean text "서울시립대학교" is written around the bottom half.

서울시립대학교 대학원
전자전기컴퓨터공학부
권 학

목 차

제1장 서 론	1
제1절 연구의 배경 및 목적	1
제2절 연구의 내용	3
제3절 논문의 구성	4
제2장 관련 연구	5
제1절 CoAP	5
제2절 센서네트워크 Middleware	17
1. 미들웨어 개념과 구조	17
2. 센서네트워크 Middleware를 위한 기능 요구사항	18
3. 센서네트워크 미들웨어의 연구 동향	19
제3장 CoAP Network 기반 Middleware API 설계	21
제1절 개요	21
제2절 연구대상	24
제3절 구현	25
1. 고려사항	25
2. Client to Server 메시지 교환	27
3. Middleware API 정의	33
제4장 성능 평가	43
제5장 결론	50
참 고 문 헌	51

ABSTRACT	53
감사의 글	55



그 립 목 차

그림 2-1 CoAP Layer	6
그림 2-2 CoAP Message Format	7
그림 2-3 CoAP Option Format	9
그림 2-4 신뢰성 있는 메시지 전송	12
그림 2-5 비신뢰성 메시지 전송	13
그림 2-6 GET요청에 대한 2가지 Piggybacked 응답	14
그림 2-7 GET요청에 대한 별도응답	15
그림 2-8 NON 요청에 대한 응답	16
그림 2-9 Middleware	17
그림 3-1 CoAP to Client Architecture	21
그림 3-2 CoAP 메시지 흐름도	23
그림 3-3 함수 동작 과정	34
그림 3-4 CoAP Multicast	41
그림 4-1 Client의 웹페이지	48
그림 4-2 핵심 코드 라인 수 비교	48

표 목 차

표 3-1 Client to Proxy HTTP 헤더	27
표 3-2 Proxy to Node CoAP Header	29
표 3-3 Node to Proxy CoAP Header	30
표 3-4 Proxy to Client HTTP Header	31
표 3-5 Middleware API 분류	33
표 4-1 노드의 등록하는 과정에 대한 비교	45
표 4-2 데이터 호출에 관한 비교	46
표 4-3 데이터 결과의 사용에 대한 비교	47
표 4-4 전체 코드의 라인 수 비교	49

제1장 서론

제1절 연구의 배경 및 목적

TCP/IP 표준을 개발하는 IETF의 상위 기구인 IAB는 굵직한 시스템뿐만 아니라 센서와 같이 작은 사물에도 TCP/IP protocol stack을 붙이기로 결정했다. 이러한 정책적 배경으로 이제 센서도 CPU를 가지는 일종의 하나의 작은 컴퓨터가 된다고 생각했고, 이 작은 사물에 OS를 올리고 나서 TCP/IP의 필요성을 느끼게 되었기 때문이다. 결과적으로 하나의 작은 오브젝트에 다섯 개의 전 계층이 올라간다는 점에서 굉장히 상징적인 작업이다. 그래서 작은 오브젝트에 적합한 프로토콜들을 개발하고 있는 중이다. 다수의 장치에 고유한 네트워크 주소를 할당하기 위해서는 IPv6의 도입이 적합하다. 하지만 IPv6 기술은 아직 널리 이용되고 있지 않을뿐더러 센서네트워크에서 사용하기에는 센서가 포함된 노드에 주는 부담이 적지 않다. 이에 따라 IETF에서 센서네트워크에서 사용할 수 있도록 CoAP, 6LoWPAN, RPL등과 같은 IPv6 관련 기술의 표준화를 진행하고 있다.

매년 IT업계를 이끌 중요한 기술을 발표하는 가트너(Gartner,2015)^[7]에서도 2015년 10대 전략 기술 동향에 사물의 인터넷을 뺄 정도로 기술발전 가능성과 중요성을 강조하고 있다. 사물들이 인터넷에 연결됨에 따라 사물간의 통신을 할 수 있는 프로토콜 또한 중요성이 확대되고 있어 IETF에서 다양한 표준들의 표준화 작업을 진행하고 있다. 그중 하나가 HTTP 웹 프로토콜에 대응되는 CoAP(Constrained Application Protocol)^[8]이다. CoAP는 메모리가 작고 처리성능도 낮으며, 전력도 충분하지 않은 노드 및 손실이 있고 저 전력에 전송률이 낮은 프로토콜이다.

CoAP는 데이터 공간(RAM)이 10kbps 이하이고, 코드 공간(Flash)이 100kbps 이하인 노드로 정의되어 있으며, 이를 기준으로 표준화를 진행하고 있다.^[6] 이러한 추세를 바탕으로 libcoap, californium, jcoap 등과 같은 다양한 라이브러리들이 개발되고 있다.

여러 Library들이 제공되는 만큼 CoAP을 이용한 네트워크 환경구축이 진행되고 있다. 또한 CoAP 노드들과의 통신을 하고 직접적으로 coap://coapaddress/query 와 같은 형태의 URI로 접근할 수 있는 Firefox 브라우저의 Plug-in도 개발되어 서비스를 제공 중이다. 하지만 이러한 CoAP 노드들의 데이터를 받아와 데이터를 처리하고 각 노드들을 관리하기 위해서는 CoAP에 관한 이해가 필수적이다. 개발적인 측면에서도 Client에게 서비스를 제공할 개발자와 Network를 구성하는 개발자간의 소통을 통해서 주고받는 데이터를 정하고 필요로 하는 기능들을 정의하는 것은 인력과 시간적인 소모는 필수적이다.

결과적으로 Middleware에서 CoAP를 알지 못하는 개발자라도 새롭게 정의된 라이브러리를 통한다면 개발시간을 단축시킬 수 있을 것이다. 본 논문에서는 이러한 CoAP에 관하여 알지 못하는 개발자라도 Middleware API를 통해서 쉽게 CoAP의 노드들을 관리할 수 있도록 도움을 주는 Middleware API를 제시하고자 한다.

제2절 연구의 내용

Middleware API는 CoAP을 잘 알지 못하는 개발자라도 쉽게 CoAP의 노드들을 관리하고 이를 컨트롤 할 수 있는 Library이다. 현재 CoAP 노드들의 데이터를 수집하고 확인하기 위해서는 CoAP에 대한 이해와 함께 CoAP관련 Library들을 이해하고 사용하여야 한다. 이와 관련한 시스템을 구축하고 서비스를 제공하기 위해서는 담당 파트의 개발자와 함께 많은 시간을 들여 의견을 조율하고 필요한 기능을 구현하였다. 따라서 손쉽게 개발자가 CoAP에 대한 이해도를 낮추고 관련 기능들을 구현한다면 개발 시간을 크게 단축시킬 수 있을 것이다.

본 논문은 Middleware API 개발에 앞서 CoAP 노드들과 CoAP/HTTP Proxy, Client의 환경을 구성을 하였다. 각각의 노드들에게 맞는 역할을 주고, 특정한 상황에 대한 이벤트를 구성하여 Middleware에서 필요한 기능들을 정의하였다. 정의된 기능들과 함께 Client에서 CoAP 노드에 닿기 까지 과정을 살펴보고 Middleware에서 할 수 있는 역할에 대해 살펴보고, 결과적으로 개발자에게 필요한 API의 기능들을 정의하도록 할 것이다.

제3절 논문의 구성

본 논문은 다음과 같이 구성되어 있다.

1장은 연구의 배경과 목적에 대하여 설명한다.

2장은 CoAP의 이론적인 지식과 통신 방법에 관하여 설명하며, 센서네트워크 Middleware에 관하여 설명한다.

3장에서는 CoAP 노드와 함께 Proxy 서버를 제작하여 실제 작동하는 방식에 관해 이해하며 관련 기능들이 어떤 것들이 있는지 설명한다. 또한 이와 관련하여 필요한 기능들에 관하여 정의하고 어떤 동작을 하는지 설명한다.

4장에서는 API를 쓴 경우와 쓰지 않는 경우의 코드 라인 비교를 통하여 성능을 평가하도록 한다.

마지막으로 5장에서는 결론과 향후 방향에 대해서 제시한다.

제2장 관련 연구

제1절 CoAP

CoAP는 2010년 초부터 IETF(Internet Engineering Task Force) 워킹 그룹(Working Group)에서 본격적으로 개발되기 시작하여 여러 차례 변화를 거듭하여 최근 RFC(Request for Comments) 7252^[8]을 발표하였다.

CoAP는 작은 용량의 메모리와 저 전력의 제한된 성능을 갖는 센서나 구동체 노드간의 통신을 지원하며 비동기적인 요청/응답 디자인 구조를 지녔다. 이는 온도, 습도, 광합성도, 기울기, 오염등과 같은 정보를 획득하는 기술에 활발히 이용되고 있다.^[4] CoAP는 HTTP와 쉽게 상호변환 및 연동이 가능하며, 사물인터넷(Internet of Thing : IoT)과 M2M(Machine-to-Machine) 환경에서 저 전력 센서와 구동체 네트워크를 통한 기반시설을 감시하거나 관리할 수 있다.^[2]

또한 CoAP는 REST(Representational state transfer)구조로 설계되어 기능이 간단하고 처리에 부담이 적다. 그리고 HTTP에서 사용하는 메서드 방식과 마찬가지로 POST, GET, PUT, DELETE 메서드를 사용하여 자원을 생성하고 삭제하며 수집하는 곳에도 쉽게 활용할 수 있다.^[5]

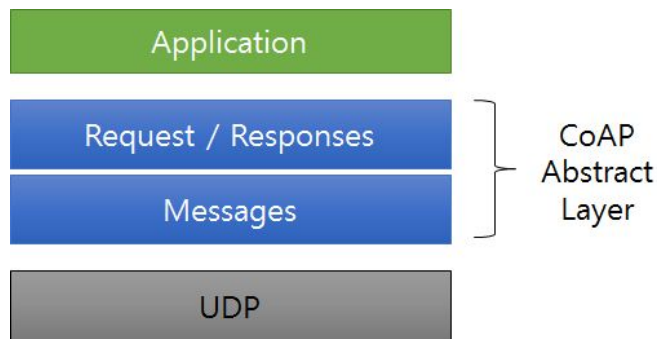


그림 2-1 CoAP Layer

CoAP는 UDP를 기반의 메시지를 이용하여 통신하며, 메시지는 Binary Format로 인코딩 된다.^[8]

기본적으로 CoAP는 메시지 요청(Request)과 응답(Response)의 형태로 작동을 하며 확인형(Confirmable), 비확인형(Non-confirmable), 응답(Acknowledgement), 재요청(Reset)의 4가지 메시지 타입을 정의하고 있다.

1. CoAP 메시지 포맷

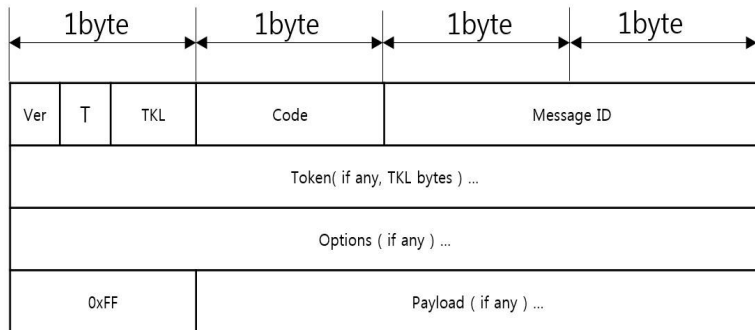


그림 2-2 CoAP Message Format

CoAP의 메시지 헤더에는 버전정보, Transaction Type, Token Length, Code, Message ID, Token, Option, Payload의 정보가 포함되어 있다.

- Ver은 CoAP의 버전정보를 위해 2비트 부호 없는 정수(unsigned integer)를 사용하고 현재의 버전을 위해 01로 설정을 한다.
- T는 2비트의 Transaction Type으로써 Confirmable(0), Non-confirmable(1), Acknowledgement(2) 또는 Reset(3) 으로 이루어져 있다.
- 4비트의 TKL은 Token Length로써 Token 영역의 가변 길이를 의미하며 0에서 8까지의 값을 사용할 수 있다. 9~15까지는 사용하지 않고 이 값이 들어올 경우에는 메시지 오류로 처리해야한다.
- Code는 8비트의 부호 없는 정수로서 3비트는 클래스를(class), 5비트

는 자세한 내용(detail)을 의미한다. 이 부분은 클래스의 값과 자세한 내용을 합하여 “c.dd”로 표현할 수 있는데 c는 0~7사이의 값이 올 수 있으며 dd는 0~31사이의 값이 온다. 클래스의 0은 요청, 2는 응답성공, 4는 클라이언트 에러 응답, 5는 서버 에러 응답의 값들을 의미한다. 0.00은 빈 메시지를 의미하고 0.01은 GET, 0.02는 POST, 0.03은 PUT, 0.04는 DELETE이다. 이러한 요청 메시지의 경우에는 GET, POST, DELETE, RESET의 메서드중 하나를 설정하고 응답코드인 경우에는 HTTP의 에러코드와 유사한 형태의 메시지 구조를 갖는다.

- MessageID는 16비트의 부호 없는 정수로서 각 메시지의 중복 및 확인성/비확인성 메시지에 대한 값으로 사용된다. 또한 신뢰성/비신뢰성 메시지에 대한 승인/리셋메시지를 보내는 데에 사용된다.
- Token은 TKL에 따라 메시지 헤더 다음으로 오는 값으로 메시지 내용의 구분식별자를 나타내는 값이다.
- Option은 토큰에 이어서 올수 있는 값으로 메시지 끝에 도달한 경우이거나 페이로드 마커인 경우이다.
- Payload marker는 옵션의 뒤에 위치하고 0xFF의 값을 가지며 이후 페이로드가 존재하는 경우에는 데이터그램의 맨 끝까지 표시된다.

2. 옵션 포맷

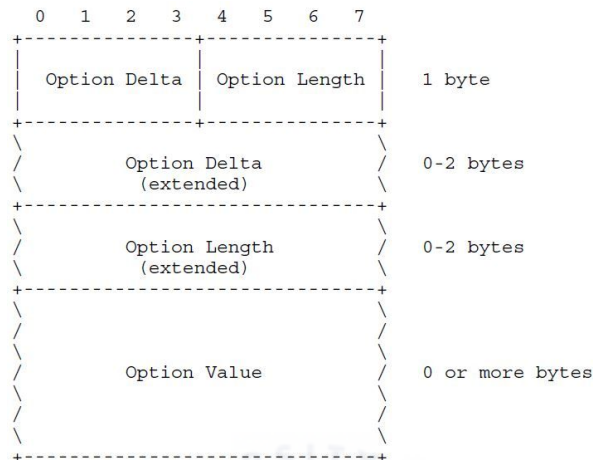


그림 2-3 CoAP Option Format

HTTP 헤더의 경우 여러 가지 옵션들이 문자열을 기반으로 이루어져 있다. 각 옵션의 구분은 ‘\r\n’ 개행문자를 통해 구분되어진다. 이에 반해 CoAP에서는 프로토콜의 최대로 경량화하기 위해서 다른 방법을 사용한다. CoAP의 옵션 부분을 살펴보면 첫 1Byte의 상위 4bit는 Option Delta, 하위 4bit는 Option Length로 구성되어 있다. 다음 0-2 Bytes는 Option Delta의 추가적인 확장 영역이고, 다음 0-2 Bytes는 Option Length의 추가적인 확장 부분이다. 그 다음으로 임의의 Byte 크기만큼 실제 Option Value가 차지하게 된다. 이런 방법은 옵션들이 차지하는 영역을 최소화하기 위한 노력이다.

초안을 살펴보면 더 자세한 정보를 알 수 있다. 만약 Option Delta 값이 0~12 사이인 경우에는 그 값 자체가 Option Delta 값이 된다. 하지만 13인 경우에는 추가적인 확장 영역 1Byte를 사용하여 표현하게 된다. 이때 확장 영역에서 13을 뺀 값이 실제 Option Delta 값이 된다. 14인 경

우에는 추가적인 확장 영역 2Bytes를 사용하여 표현하게 된다. 이때에는 확장 영역에서 269를 뺀 값이 실제 Option Delta 값이 된다. 15인 경우는 추 후 기능 확장을 위해 예약되어있는 값이다. 그리고 여러 개의 옵션이 헤더에 포함 될 경우에는 제일 값이 작은 옵션을 우선순위로 앞에 배치하며 다음으로 따라붙은 옵션들은 이전 옵션 값과의 차이 값으로 한다. Run Length Code의 방식과 유사하다. 이 것 역시 옵션의 길이를 최소화 하기 위한 노력이다.

Option Length의 경우도 Option Delta와 같은 방식이다. 만약 Option Length 값이 0~12 사이인 경우에는 그 값 자체가 Option Length 값이 된다. 하지만 13인 경우에는 추가적인 확장 영역 1Byte를 사용하여 표현하게 된다. 이때에 확장 영역에서 13을 뺀 값이 실제 Option Length 값이 된다. 14인 경우에는 추가적인 확장 영역 2Bytes를 사용하여 표현하게 된다. 이때에는 확장 영역에서 269를 뺀 값이 실제 Option Length 값이 된다. 15인 경우는 추 후 기능 확장을 위해 예약되어있는 값이므로 이 값이 설정된다면 메시지 형식 오류로 처리해주어야 한다.

몇 가지 예를 들어보면

Option Delta: 7

Option Delta extended: X

Option Length: 2

Option Length extended: X

인 경우 실제 Option Delta는 7, Option Length는 2 가 된다.

Option Delta: 13

Option Delta extended: 2

Option Length: 5

Option Length extended: X

인 경우 실제 Option Delta는 15, Option Length는 5 가 된다.

Option Delta: 14

Option Delta extended: 13

Option Length: 13

Option Length extended: 1

인 경우 실제 Option Delta는 282, Option Length는 14 가 된다.

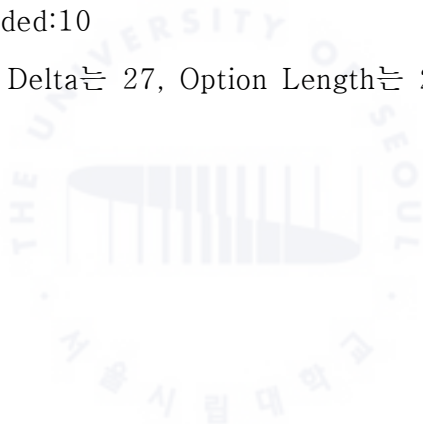
Option Delta: 13

Option Delta extended: 14

Option Length: 14

Option Length extended:10

인 경우 실제 Option Delta는 27, Option Length는 279 가 된다.



3. CoAP 동작 방식

1) 메시지 모델

CoAP 메시지 모델은 UDP를 통한 endpoint들의 메시지 교환을 기반으로 한다. 메시지 모델에는 신뢰성 있는 메시지 전달과 신뢰성이 요구되지 않는 메시지 모델이 있다. 신뢰성 있는 메시지 전달을 위해서는 확인형(CON)메시지를 전송한다. 이때 전송한 메시지 아이디로 승인(ACK)메시지에 포함하여 전송하게 된다. 수신자가 CON메시지를 처리하지 못할 때에는 리셋(RST) 메시지를 확인형(ACK) 대신에 보내게 된다.

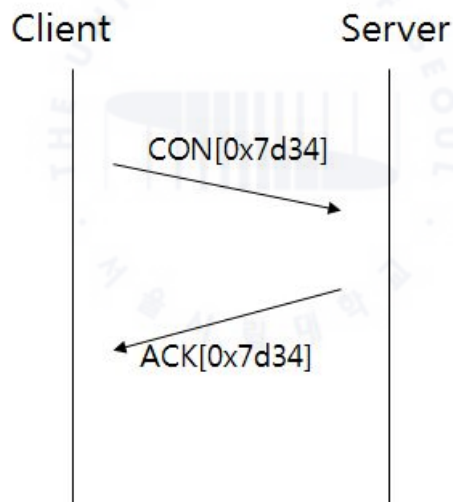


그림 2-4 신뢰성 있는 메시지 전송

신뢰성이 요구되지 않는 경우에는 비확인성 메시지(NON)를 전송할 수 있다. 메시지 ID는 중복 검출용으로 사용되고, NON메시지를 받은 endpoint는 ACK 메시지를 보낼 필요가 없다. Endpoint에서 비확인성 메시지(NON)를 처리할 수 없을 경우에는 리셋 메시지(RST)로 응답할 수

있다.

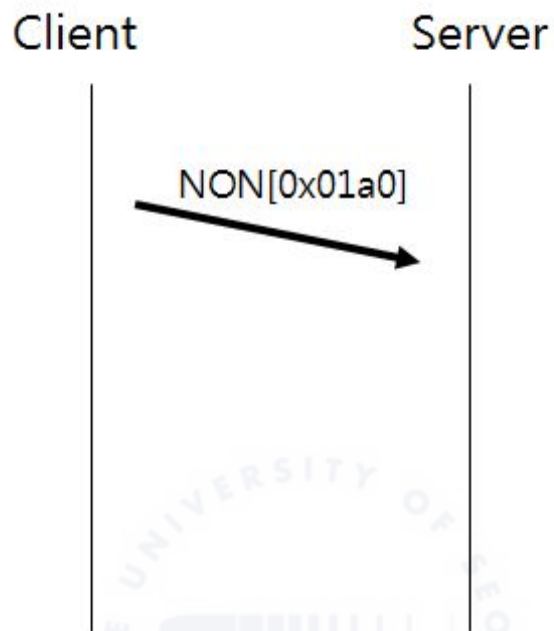


그림 2-5 비신뢰성 메시지 전송

2) 요청/응답 모델

CoAP 요청 및 응답 의미들은 메서드 코드와 응답코드 각각이 CoAP 메시지와 같이 전송된다. URI와 페이로드 미디어 타입과 같은 옵션 요청과 응답 정보는 CoAP 옵션에 포함된다. 토큰은 응답 메시지와 독립적인 요청에 대한 응답과 일치시키기 위해 사용된다. 즉, 토큰이라는 필드를 통해 짝을 이루며, 메시지 ID는 토큰과는 다르게 하나의 트랜잭션을 구분하는데 사용된다. 아래 그림에서 메시지를 주고받으며 토큰과 아이디의 쓰임새를 보여주는 과정을 설명하고 있다. GET 요청에 대해 Piggybacked 형태로 응답하는 과정을 보여주는데 이는 CON 이나 NON 메시지를 전송했을 때 즉시 응답가능 할 경우 ACK 메시지로 응답 결과를 알려주는 과정이다. 아래 그림의 좌측 트랜잭션은 성공적으로 메시지에 대한 응답을 한 결과이고, 우측은 클라이언트의 요청에 대한 답을 찾지 못한 결과를 전송하는 예를 보여준다.

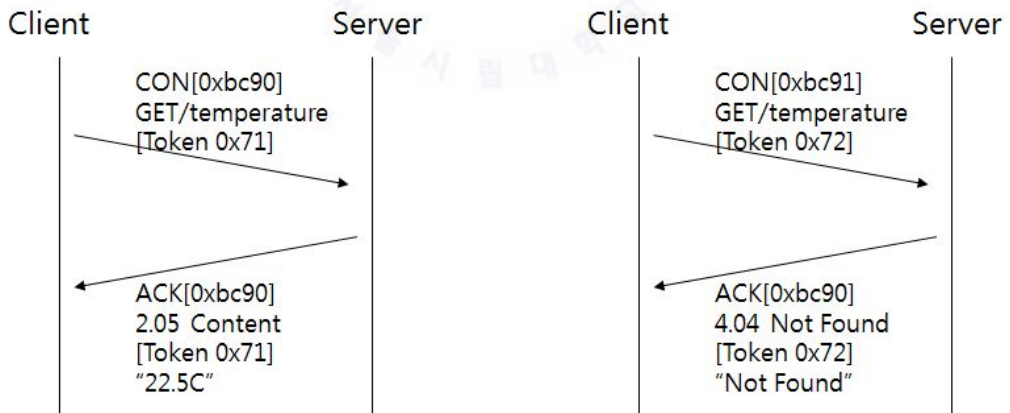


그림 2-6 GET요청에 대한 2가지 Piggybacked 응답

CoAP서버에서 CON메시지의 요구에 즉각적으로 응답할 수 없는 경우

에는 내용이 없는 ACK메시를 먼저 전송하고, 그 뒤 준비가 된 경우 다시 응답으로 CON메시지를 전송하게 된다. 이를 “별도 응답”이라고 하여 아래 그림에서 보여주고 있다.

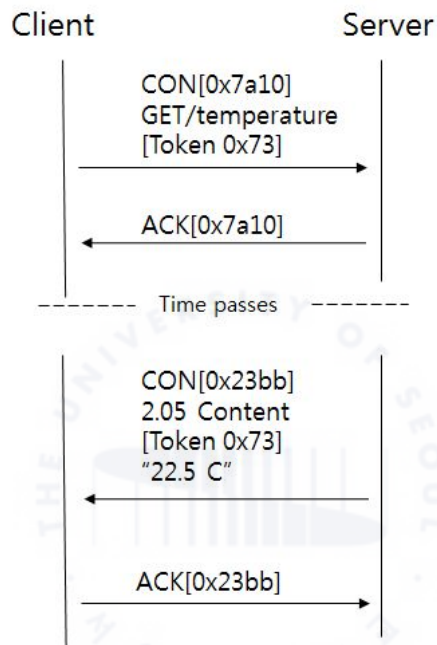


그림 2-7 GET요청에 대한 별도응답

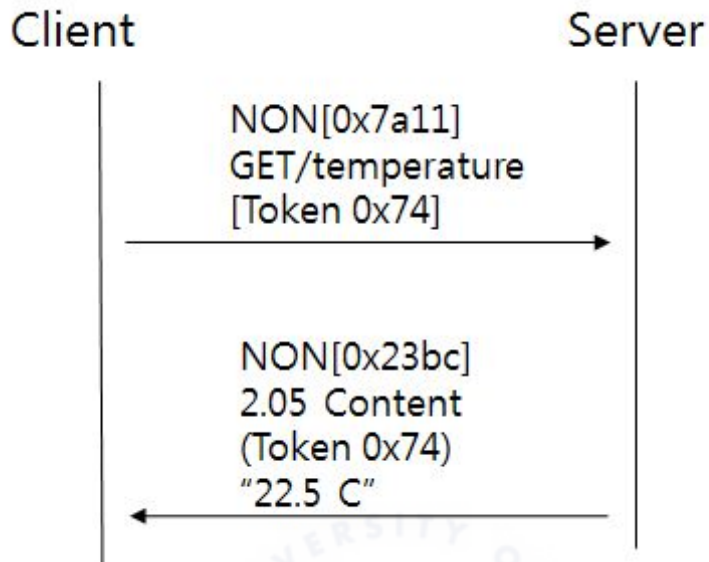


그림 2-8 NON 요청에 대한 응답

CON 메시지를 보내지 않고 NON 메시지를 통해서 NON으로 응답받는 요청을 할 수 있다.

제2절 센서네트워크 Middleware

1. 미들웨어 개념과 구조

미들웨어는 클라이언트와 서버 프로그램을 연결해주는 다리역할을 하고 있으며, 클라이언트와 서버 구조에서 존재하는 형태로서 양쪽의 데이터를 주고받으며 매개역할을 하는 소프트웨어이다.

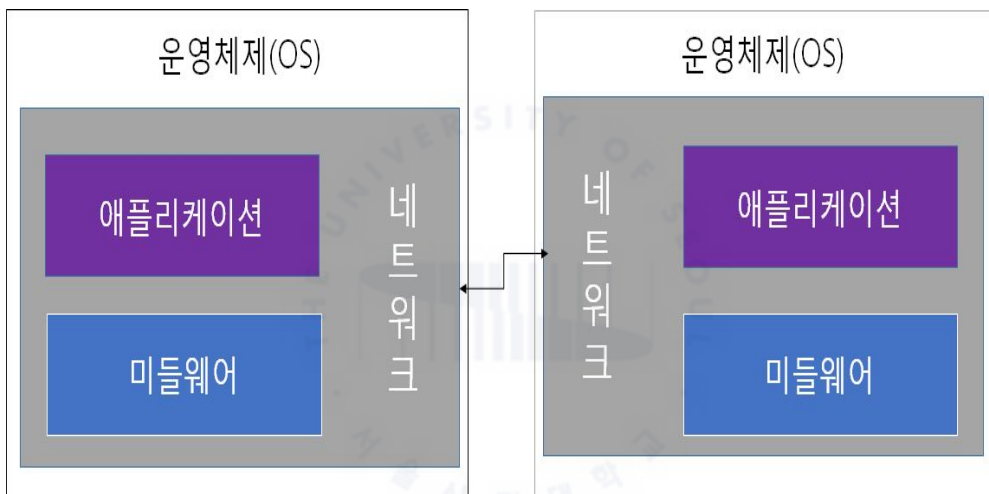


그림 2-9 Middleware

미들웨어는 크게 데이터베이스 미들웨어와 통신 미들웨어로 나눌 수 있다. 넓은 의미에서 미들웨어는 어플리케이션과 어플리케이션의 통신을 담당하는 통신미들웨어를 얘기한다.

김대영 외(2005)에 따르면 센서네트워크 미들웨어는 운영체제와 어플리케이션 사이에 존재하며 하드웨어 의존적인 구현에서 오는 복잡함과 하위 레벨 종속적인 특징들을 추상화 하여서 어플리케이션 개발자에게 API를 제공하는 계층으로 정의할 수 있다.^[1] 센서네트워크의 미들웨어가 하

는 주요한 역할은 센서네트워크의 데이터를 수집, 이벤트 처리, 전력관리와 네트워킹 기능을 사용자가 알지 못해도 쉽게 이를 사용할 수 있도록 한다. 또한 개발자가 하위 계층의 구조에 대한 이해와 이론적인 지식이 없이도 적절한 쿼리를 통한다면 센서네트워크의 기능을 상위 계층으로 제공하여 전반적인 네트워크 계층에 대한 자세한 이해 없이 사용할 수 있다.

2. 센서네트워크 Middleware를 위한 기능 요구사항

센서네트워크의 Middleware 요구사항은 센서네트워크의 요소 개발, 센서 노드의 등록 및 배포, 센서의 유지 및 관리, 데이터 수집, 감지기반 응용프로그램 수행이다.^[3] 미들웨어는 상위 태스크에게 센서네트워크로부터 통신하여 얻은 데이터를 전달하여 이에 맞는 역할을 하도록 하고, 상위 어플리케이션 태스크로부터 받은 데이터를 가공하여 각각의 노드에게 데이터를 전송해주어야 한다.

Middleware는 센서의 종류에 따라 다양한 질의를 할 수 있어야 한다. 현재 센서 노드의 데이터수집, 센서의 상태를 확인, 센서에 관한 정보수집등과 같은 질의유형을 확보해야 한다. 이러한 질의를 통해 얻은 데이터를 DB에 저장하고 관리하는 기능을 지원하도록 한다. 센서 노드에 대한 질의를 하기 위해서는 센서네트워크의 이름, 센서가 가지는 정보에 관한 종류, 그룹, 센서 노드 전체의 개수, 센서 노드의 연결 상태 등의 정보를 확인할 수 있어야 한다. 이는 주기적으로 모니터링 하여 관련 정보들을 응용어플리케이션에게 제공할 수 있도록 해야 한다. 그 외에도 센서 네트워크의 하위 노드들은 전력이 매우 제한적이기 때문에 이러한 전력을 관리하기 위한 플랫폼을 고려하여 설계해야하며 의심스러운 코드가 실행되

지 않는 보안기능 또한 필요로 한다.

3. 센서네트워크 미들웨어의 연구 동향

1) TinyDB^[11]

TinyDB는 버클리 대학에서 진행한 프로젝트로 DB기반으로 WSN을 가상의 DB로 활용하는 미들웨어이다. TinyOS 기반의 센서노드에서만 동작을 한다는 제한사항을 지니고 있다. 사용자는 SQL만 알고 있다면 센서노드로부터 데이터를 손쉽게 가져올 수 있다. 또한 전력절감에 관하여 고려하여 설계가 되어 있으며 전력소모를 위한 최적화 방법을 지닌다.

2) Cougar^[12]

Cougar는 Cornell대학에서 연구한 센서네트워크용 분산 데이터처리 시스템으로 SQL-like를 지원한다. DB 기반으로 하여 데이터에 접근하고 처리하는 형태를 모두 분산된 방법으로 수행하도록 한다. 또한 Cougar는 네트워크 환경의 변화에 대해 감지하여 적응할 수 있고, 유연성과 확장성이 높으며, 고장 감내성을 가진 시스템을 목표로 진행하고 있다.^[3]

3) SINA^[13]

SINA는 Cougar와 마찬가지로 서버사이드의 미들웨어이다. SINA는 지리적으로 가까이 있는 센서노드들을 그룹화 하여 관리하기 때문에 원하는 데이터를 얻고자 할 때는 관리하는 그룹을 통해서만 정보를 얻을 수 있다.

4) DSWare^[14]

DSWare는 서버사이드 미들웨어와 In-network의 특징을 가진 미들웨어로써 이벤트 중심으로 데이터를 처리하여 제공하는 특징을 지니고 있다. DSWare는 센서노드들의 그룹화 하여 관리할 수 있도록 지원하고 있다.

5) HYDRA^[15]

HYDRA는 IoT 기반의 플랫폼으로서 서비스 지향 미들웨어를 발전시켜 복잡도를 감소시키는 것을 목적으로 한다. HYDRA는 빌딩오토메이션, 헬스케어, 농업에서 사용되기 위해 개발되었다. HYDRA는 센서노드의 디바이스들이 변경되더라도 쉽게 알아채는 동기적 방식이며 유지보수화 확장성 면에서 뛰어난 성능을 보여준다. HYDRA는 웹서비스 인터페이스를 통하여 물리적인 장비나 센서, 시스템 등에 쉽게 적용하도록 제공하고 있다. 현재 LinkSmart라는 Open Source Middleware로 제공되고 있다.

제3장 CoAP Network 기반 Middleware API 설

계

제1절 개요

본격적으로 가상의 IoT 생활 가전 환경을 구성하여 CoAP 사용을 위한 시스템을 설계하고 구현하였다. 그리고 이와 관련하여 CoAP 센서노드들과 연결하여 데이터를 처리하고 프록시와 소통을 하는 미들웨어에 관한 API를 설계한다.

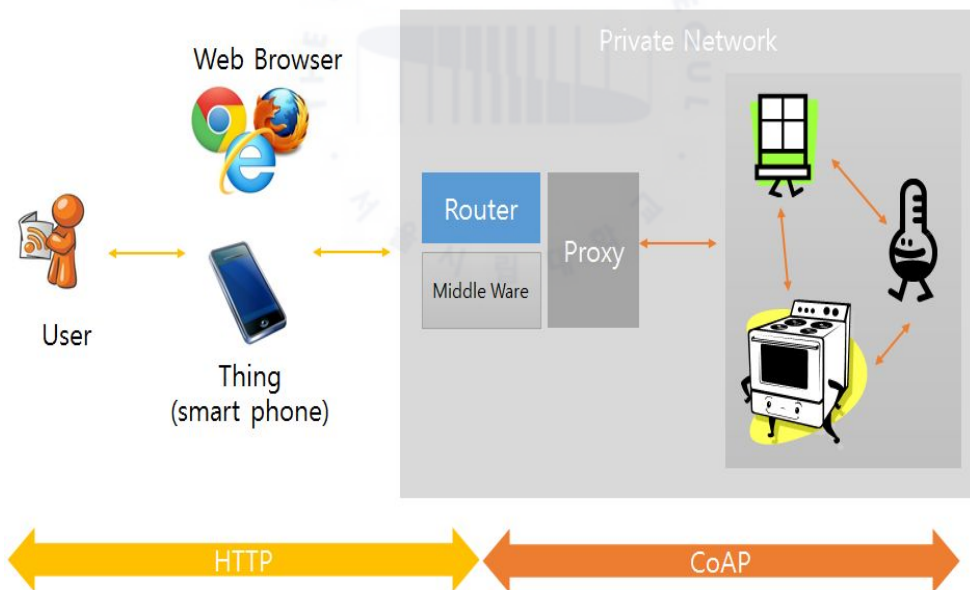


그림 3-1 CoAP to Client Architecture

먼저 CoAP과 HTTP를 이용하여 Home Network를 구성하고 이와 관련하여 Middleware에서 필요한 기능들을 정의한다.

[그림 3-1]과 같은 Home Network 상황을 가정 하였을 경우, 센서노드들은 각각 창문과 가스레인지를, 보일러라는 역할을 주어 센서의 역할을 하도록 설계 하였다. 센서노드들은 아두이노에 센서를 부착하여 센서의 데이터를 측정하도록 한다. 센서는 각각의 역할에 맞게 온도센서와 가스센서를 통해서 데이터를 수집하고 사용자가 설정한 범위 내의 값에 따른 역할을 하도록 정의하였다. 가스레인은 가스센서, 창문은 온도센서, 보일러에는 온도센서를 두어 실내의 온도가 급격히 높아지거나 가스가 누출 되었을 때의 상황에 따른 역할을 할 수 있도록 하였다. 외부에서 사용자가 직접 센서의 데이터를 보고자 할 경우를 위하여 요즘 대부분의 일반 가정 집 환경과 유사하게 네트워크의 가장 앞단에 공유기(mini Router)를 두었고 내부 네트워크에 HTTP to CoAP 프록시 서버와 CoAP 노드들을 배치하였다.

사용자가 웹 클라이언트로 프록시 서버에게 정보를 요청하게 되면, 이때 프록시는 받은 HTTP 메시지를 CoAP으로 변환하여 CoAP 노드들에게 요청을 보낸다. 요청을 받은 CoAP 노드는 자신의 Sensor 혹은 Item 의 상태나 수치를 확인하고 페이로드에 실어 프록시 서버에게 응답한다. CoAP 응답을 받은 프록시 서버는 다시 HTTP 로 변환하여 사용자의 웹 클라이언트에게 데이터를 전송하게 된다. GET, POST, PUT, DELETE 메서드를 이용해 CoAP 노드들에게 각각의 센서의 정보를 요청을 하고 응답 받을 수 있다. 사용자는 화면에 표시된 정보를 통해 가정 내부의 정보를 확인 할 수 있고, 자신이 원하는 상태로 가전기기의 상태를 변경할 수 있다. 우리가 제공한 환경에서는 사용자가 보일러의 상태 여부와 창문의 개폐 여부를 결정할 수 있고 실내의 온도, 습도를 확인할 수 있다.

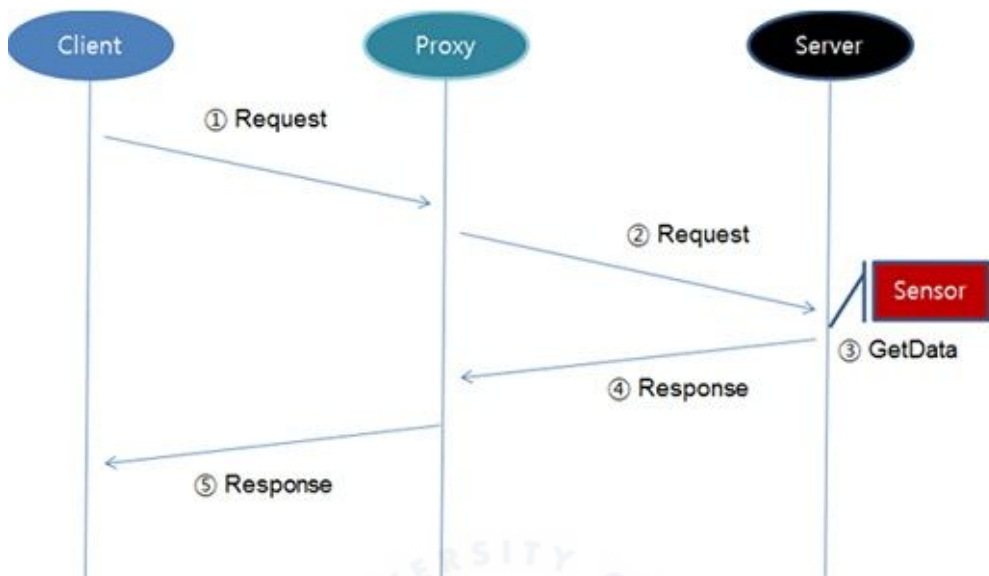


그림 3-2 CoAP 메시지 흐름도

[그림 3-2]은 센서노드의 메시지를 전송받기 위한 전체적인 메시지 흐름도이다. Client(사용자)가 센서의 데이터 값을 요청하면 이 요청은 Proxy로 전송이 되고 Proxy에서는 CoAP로 데이터를 변환하여 센서노드에 정보를 요청하게 된다. 센서노드는 자신의 데이터를 CoAP 형식에 맞춰 Proxy로 전송을 하고 Proxy는 수신한 데이터를 재가공하여 Client에게 전송하도록 한다.

제2절 연구대상

본 논문에서는 CoAP로 통신하는 환경과 HTTP 접근 환경을 구현하고 이와 관련하여 필요한 Middleware API와 관련한 연구를 진행한다. Arduino를 이용해 간단한 스마트 환경을 구현하고, Proxy에서 Sensor node와 HTTP 클라이언트에게 제공할 기능들을 정의해 이에 필요한 API들을 작성한다.

현재 CoAP 네트워크 환경을 구현하기 위해 여러 라이브러리들이 제공된다. 그중에서 Proxy에서는 Californium^[9] Library를 선택하였다. Californium은 자바 기반의 오픈소스 프로젝트로 Server와 Client를 모두 지원한다. Server측은 Arduino Uno를 사용하여 제한된 노드의 환경을 구현하도록 한다. Gas, Temperature, Humidity 센서를 Arduino에 올려서 각 센서의 역할을 하도록 한다. Server측은 libcoap^[10]을 이용하여 Arduino에서 작동하도록 구현하였다. libcoap는 C언어 기반으로 작성된 CoAP 라이브러리로 Arduino에 적용하기 간편하다.

Client가 요청하는 부분은 웹기반으로 하여 사용자가 웹페이지에 접근 시 관련 노드들에 관한 정보를 탐색 할 수 있도록 하였다. CoAP도 HTTP와 같은 메시지를 사용하기 때문에 웹기반으로 구현하여 CoAP에 관한 이해를 돕고자 했다.

제3절 구현

1. 고려사항

본 논문에서 구현한 API는 어플리케이션 계층의 S/W에서 구현하는 부분에서 사용할 함수들을 작성하는 것이다. 이를 위해 CoAP 노드들은 아두이노를 이용하여 센서들을 접합시켜 CoAP을 통해 통신할 수 있도록 구현한다. 각각의 노드는 가스 정보를 갖는 가스센서와 온도와 습도를 측정할 수 있는 보일러, 실내외 온도를 측정하는 창문이 있다. 각각의 노드는 노드가 갖는 주소에 대하여 요청을 보낼 시에 온도와 습도, 가스센서를 통한 가스의 측정 데이터를 반환해주는 역할을 한다. Proxy를 통해서 클라이언트가 요청하는 메시지를 수신할 경우, 메시지를 파싱한 후 응답 메시지의 상태를 검사한다. 수신한 메시지에 에러가 있는 경우, 그 에러에 적합한 에러코드를 사용하여 응답메시지를 전송한다. 세 개의 네트워크 엘리먼트라 정상적으로 동작하기 위해서는 트랜잭션 아이디가 필수적이기 때문에, 수신메시지를 복사하여 사용하는 것이 일반적이다.

HTTP/CoAP Proxy는 클라이언트와 서버(Arduino Sensor Node)사이에서 데이터를 수신하고 전달하는 중계기 역할을 하는 노드이다. 센서의 자원이 취약하기 때문에 자원이 풍부한 Proxy서버는 수신된 메시지에서 정확한 목적지를 알아야 한다. 정확한 목적지를 알기 위해서는 URL_SCHEME 옵션 정보를 이용한다. 수신한 메시지에서 목적지에 관한 정보를 추출한 후, 전달할 목적지로 변경하여 메시지를 전달한다. 전달한 목적지에서 받은 메시지를 다시 요청하는 노드에게 전달하는 과정도 요청 메시지와 동일한 방법으로 작동을 한다.

본 논문에서 구현한 Client 부분의 미들웨어 API는 프로그램 작성자가

CoAP에 관한 사항들을 알지 못하여도 CoAP 노드들에게 데이터를 요청하고 이를 컨트롤 할 수 있도록 해야 한다. 그에 따라 사용자가 각각의 노드의 이름을 설정하고, 설정된 노드의 이름을 통해서 이를 관리할 수 있는 구조로 설계하여야 한다.



2. Client to Server 메시지 교환

1) Client to Proxy

```
GET /proxy/192.168.100.20/gas?_=1377946214228 HTTP/1.1\r\n
HOST: 112.108.40.147\r\n
Connection: keep-alive\r\n
Accept: application/json, text/javascript, */*;q=0.01\r\n
X-Requested-with: XMLHttpRequest\r\n
user-Agent: Mozilla/5.0 (windows NT 6.2;wdw64) AppleWebKit/537.36 (KHTML,
like Gecko) chrome/29.0.1547.57 safari/537.36\r\n
Referer: http://112.108.40.147/\r\n
Accept-Encoding: gzip,deflate,sdch\r\n
Accept-Language: ko-KR, ko;q=0.8,en-US;q=0.6,en;q=0.4\r\n
Cookie: doxygen_width=300\r\n\r\n
[Full request
URI:http://112.108.40.147/proxy/192.186.100.20/gas?_=1377946214288]
[HTTP request 1/2]
[Response in frame: 38]
[Next request in frame: 71]
```

표 3-1 Client to Proxy HTTP 헤더

프록시 서버가 웹 클라이언트로부터 받은 HTTP 요청 메시지이다. 프록시 서버는 먼저 HTTP 헤더의 옵션을 ‘\r\n’ 을 기준으로 분리한다. 분리된 옵션을 CoAP 옵션 테이블을 검색하며 CoAP에 존재하는 옵션이면 변환한다. 그리고 CoAP 패킷을 빌드한다. HTTP와 CoAP을 매칭 시키기 위해 Hash 테이블에 등록 후, 패킷을 CoAP 노드에 전송한다.

위와같은 기능적인 측면은 Middleware에서 데이터를 얻어오는 함수를 통해 정의할 수 있다. getSensorData(gas)와 같은 정의를 통해 구현된 기능들이 gas라는 센서노드의 이름으로 Proxy에 쿼리를 보내 관련 데이

터를 요청한다. Client에서 Proxy로 GET 요청을 보내는 것과 같은 방법으로 Middleware에서 Proxy로 데이터를 요청하는 과정을 내부적으로 구현하여 하나의 메서드를 통하여 관련 기능을 대행하도록 한다. 이 외에도 multicast의 경우 그룹 지어진 노드들에게 각각 메시지를 보내도록 그룹 내의 노드에 관한 정보를 받아와 각 노드에게 broadcast하여 노드의 관련 정보를 받아올 수 있도록 정의해야 한다.



2) Proxy to Arduino

```

192.168.100.5      192.168.100.20    COAP    71.    CON,    MID:13239,    GET,
TKN:ab 04 /gas_=1377946214288?
01.. .... = version:1
..00 .... = Type: Confirmable (0)
.... 0010 = Token Length: 2
Code: GET (1)
Message ID: 13239
Token: ab04
Opt Name: #1: Uri-path: gas
    opt Desc: Type 11. Critical, Unsafe
    1011 .... = Opt Delta: 11
    .... 0011 = opt Length: 3
    uri-Path: gas
Opt Name: #2: Uri-Query: _=1377946214288
    opt Desc: Type 15. Critical, Unsafe
    0100 .... = Opt Delta: 4
    .... 1101 = opt Length: 13
    Opt Length extended: 2
    Uri-Query: _=1377946214288
Opt Name: #3: Accept: application/json
    opt Desc: Type 16. Elective, Safe
    1011 .... = Opt Delta: 1
    .... 0011 = opt Length: 1
    Accept: application/json

```

표 3-2 Proxy to Node CoAP Header

CoAP으로 변환된 패킷이 아두이노 노드로 향하는 모습이다. CoAP 프로토콜의 Type은 HTTP의 메서드로 정해졌다. HTTP GET 메서드의 인자로 따라 붙은 URL들은 코애플의 Uri-Path 옵션으로 분리되어 순서대로

삽입되었다. Uri-Query 옵션 역시 따로 분리되어 적용된 모습이다. 이 데이터는 가스 센서의 정보를 질의하고 있는 예시이다.

Clinet로부터 gas에 관한 정보를 요청 받은 Proxy는 gas노드로 정보를 요청한다. 이는 Proxy가 알고 있는 gas 노드의 Uri로 관련 정보를 요청한다. 이렇게 요청하는 과정은 Middleware의 사용자는 알지 못하여도 충분히 gas라는 이름 하나만으로도 그에 관한 정보를 요청하도록 한다.

3) Arduino to Proxy

```

192.168.100.20 192.168.100.5 COAP 62. ACK, MID:13239, 2.05 Content,
TKN:ab 04 (application/json)
01.. .... = version:1
..10 .... = Type: Acknowledgement (2)
.... 0010 = Token Length: 2
Code: 2.05 Content (69)
Message ID: 13239
Token: ab04
Opt Name: #1: Content-Format: application/json
  opt Desc: Type 12, Elective, Safe
  1100 .... = Opt Delta: 12
  .... 0001 = opt Length: 1
  Content-type: application/json
End of options marker: 255
Payload: Payload Content-Format: application/json, Length: 11
  Payload Desc: application/json
  JavaScript Object Notation: application/json
    Object : Member Key: "gas"
      Number value: 185
  
```

표 3-3 Node to Proxy CoAP Header

아두이노는 받은 요청에 대한 응답으로 가스에 대한 정보를 알려주고 있다. gas노드는 자신의 센서에 있는 gas에 관한 정보를 얻어와 Proxy로 전송하게 된다. gas에 관한 정보는 센서에서 제공해 주는 voltage에 따라 다르기도 하나 이는 CoAP노드에서 받은 데이터를 가공한 상태로 보내주거나 가공하지 않는 상태로 보내주는 정책에 따라 Proxy로 보내주는 값이 달라지게 된다. gas는 일반적으로 사용자가 쉽게 알 수 있는 상태인 1~1000의 값으로 전송하게 된다. 이 때 데이터 포맷은 json 형식이다. 이렇게 변환되는 데이터의 값에 관한 정보 또한 Middleware를 구현하는 사용자는 알지 못하여도 노드의 데이터를 얻어올 수 있다.

4) Proxy to Client

```
HTTP/1.1 200 OK\r\n
cache-control: max-age=60\r\n
content-type: application/json; charset=ISO-8859-1\r\n
Date: Sat, 27 Dec 2014 10:50:19 GMT\r\n
Server: Californium Http Proxy\r\n
Content-Length:11\r\n
Connection: keep-alive\r\n
\r\n
[HTTP Response 1/2]
[Time since request: 0.068103000 seconds]
[Request in frame: 9]
[Next request in frame: 71]
[Next response in frame: 97]
Javascript Object notation: application/json
  Object
    Member key: "gas"
      Number value: 185
```

표 3-4 Proxy to Client HTTP Header

응답을 받은 프록시 서버는 다시 CoAP을 HTTP로 역변환 하고 페이로드가 있다면 함께 싣는다. 그리고 변환한 정보를 요청한 Client에게 전송해 준다.

Proxy로부터 받은 데이터는 Middleware에서 String 형태로 전달받아 사용자에게 정보를 제공하도록 한다. 간단히 gas 노드에 있는 값 하나만을 받아오는 경우 String 형으로 gas의 값을 전달하여도 된다. 하지만 노드의 그룹이나 모든 노드의 이름을 사용자가 호출 하는 경우는 List의 형태로 사용자에게 제공된다.



3. Middleware API 정의

Middleware API는 크게 5가지로 나뉘어 진다. 센서노드를 등록하고 삭제하는 등록, 센서데이터에 관한 전반적인 정보를 관리하는 정보, 센서노드로부터 데이터를 받아와서 값을 리턴해주는 데이터, 사용자가 직접 관리할 수 있는 사용자모드 그리고 특정노드들을 그룹으로 묶어 그룹으로 정보를 얻도록 하는 멀티캐스트로 나뉘어 진다.

분류	함수
등록	<ul style="list-style-type: none"> • registerCoapNode() • deleteCoapNode()
정보	<ul style="list-style-type: none"> • getWellknown() • getNodeName() • nodeSleep() • startCoapNode() • confirmationCoapNode() • isValidEndpoint()
데이터	<ul style="list-style-type: none"> • getSensorData() • getNodeDataFromDatabase()
사용자모드	<ul style="list-style-type: none"> • setAllUserMode() • setUserMode() • usermodeStateController() • setAlarm()
멀티캐스트	<ul style="list-style-type: none"> • groupGenerator() • registGroupNode() • getGroupNode() • getGroupInfo() • groupGETRequest() • groupPUTRequest() • groupPOSTRequest() • groupDELETERequest()

표 3-5 Middleware API 분류

이러한 함수들은 [그림 3-3]와 같은 동작으로 이루어진다. 사용자로부터 변수를 받아와 그 변수를 통해 DB내의 정보를 확인한다. DB에 정보가 있다면 그에 관한 URL과 함께 노드 이름 등의 정보를 받아와 Proxy로 요청을 보낸다. Proxy는 하위 노드에게 Request메시지를 보내어 원하는 정보를 받아와 미들웨어로 값을 넘겨주게 된다. Proxy로부터 데이터를 수신하여 정상적인 데이터인지 확인한 뒤 정상적인 데이터라면 사용자에게 요청한 데이터의 값을 넘겨주게 된다. 정상적이지 않을 경우는 에러 메시지를 통해 사용자에게 알리도록 한다.

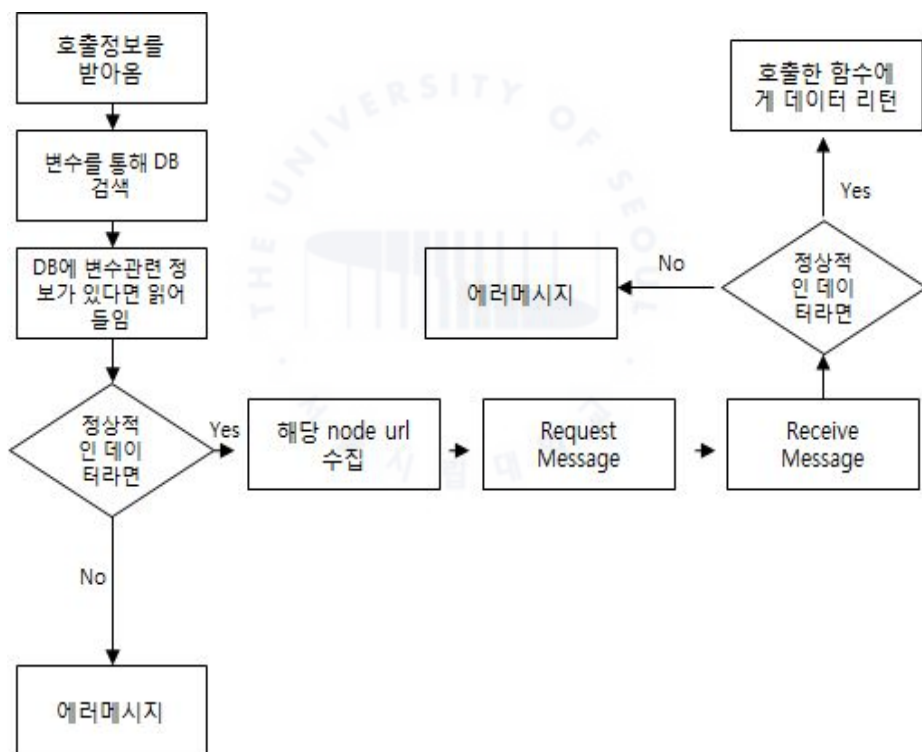


그림 3-3 함수 동작 과정

API 사용과정으로는 먼저 센서 노드의 등록하는 부분을 작성해야한다. 프록시 서버 하위에 물리적으로 네트워크를 연결한 뒤 그 노드에 관한 주소, 이름, 센서의 타입을 등록하여야 한다. 그 뒤 사용자가 센서에 관한

정보를 원할 시에는 등록된 센서의 이름을 이용하여 호출한다. 리턴 받은 데이터는 사용자가 원하는 형태로 가공하여 이용할 수 있다. 또한 사용하던 센서노드를 제거해야 하는 경우는 제거하고자 하는 센서의 이름을 통하여 함수를 호출하여 DB에서 제거하도록 한다.

1) 등록

- `public boolean registerCoapNode(String nodeName, URI uri, int type);`

CoAP 네트워크상에 한 개의 노드를 추가하고 그 노드에 관한 정보를 등록하는 함수이다. `nodeName`은 사용자가 지정할 노드의 이름이다. `uri`는 노드의 URL에 관한 정보이다. `type`는 센서의 종류에 관한 정보를 지정한다. 예를 들어 스위치 같이 on/off만을 판단하는 노드의 경우는 0번, 온도 센서와 같은 데이터 값을 측정하는 노드의 경우는 1번으로 하여 CoAP노드의 종류가 무엇인지 저장토록 한다.

- `public boolean deleteCoapNode(String nodeName);`

CoAP 네트워크상에서 CoAP 노드를 제거하는 경우 사용하는 함수이다. 제거하고자 하는 노드의 이름을 통해 그 노드를 찾아와 삭제하도록 한다.

2) 정보

- `public List<WellKnown> getWellknown();`

“./well-known/core”를 통하여 리소스에 관한 모든 정보를 받아오도록 한다. 이는 Proxy에서 연결할 수 있는 모든 링크들의 집합을 리턴 한다.

- `public List<String> getNodeName();`

`getNodeName` 함수는 DB에 저장되어 있는 노드들의 이름을 모두 불러 오도록 한다. 이렇게 불러온 노드들은 다양하기 때문에 List로 리턴을 받아 사용자가 원하는 노드의 이름을 통해 그에 관한 정보를 확인할 수 있다. 사용자가 CoAP 노드를 등록할 때 `nodeName`과 URL과 `type`를 등록하도록 하였다. 그중 사용자가 알기 쉬운 노드의 이름을 통해 이를 호출하고 그와 관련한 데이터를 얻도록 하는 것이 가장 좋은 방법이었다. 또한 사용자가 지정한 노드의 이름을 통해 질의를 할 수 있기 때문에 이와 같은 방법이 효율적이었다.

- `public void nodeSleep(String nodeName);`

CoAP는 Constrained Application Protocol인 만큼 제한된 환경에서 사용되는 네트워크 프로토콜이다. CoAP는 동기적인 방식이 아닌 비동기(Asynchronous)적인 방식인 만큼 요청 결과가 동시에 일어나지 않는다. CoAP 노드는 Sleep모드로 대기하여 이벤트가 발생할 때 그에 관한 데이터를 처리하여 노드의 디바이스 전원을 절약할 수 있다. 일정 스케줄에 따라 sleep 모드로 잠들게 되어있는 노드들에게 즉각 sleep모드로 변환되도록 한다.

- `public void startCoapNode(String nodeName);`

sleep 모드로 변환된 노드들을 깨우는 역할을 하는 함수이다. 일반적으로 센서네트워크에서 일정시간동안이 지난 후에는 슬립모드(Sleep Mode)로 전환되도록 설계를 한다. 슬립모드가 되는 경우 데이터를 신속하게 받을 수가 없고 지속적인 데이터 수집이 용이하지 않기 때문에 사용자가 직접 슬립상태의 노드를 깨워서 관리할 수 있도록 한다.

- `public boolean confirmationCoapNode(String nodeName);`

특정 노드가 정상적으로 작동중인지 확인해보는 함수이다. 특정 노드에게 다음과 같은 CON 타입의 메시지를 보내어 정상적으로 데이터가 들어오는 지 확인한다.

Header: GET (T=CON, Code=0.01, MID=0x7d34) Uri-Path: "temperature"

위와 같은 요청을 보냈을 때 다음과 같은 답이 온다면 정상작동인 것을 확인할 수 있다.

Header: 2.05 Content (T=ACK, Code=2.05, MID=0x7d34) Payload: "22.3 C"

하지만 서버 측에서 정한 시간 내에 응답이 돌아오지 않을 경우 전송했던 데이터를 참조하여 다시 메시지를 전송하게 된다. 이러한 방법으로 반복적으로 메시지를 보내 정해진 횟수를 초과하여 전송하여도 데이터를 수신할 수 없는 경우에는 사용자에게 false를 리턴 하여 사용자가 확인할 수 있도록 한다.

- `public boolean isValidEndpoint(String nodeName);`

특정 노드에 대해 정상적인 노드인지 확인하는 함수이다. 네트워크 연결이 끊어지거나 사용자가 제거한 노드임에도 데이터에서 삭제 하지 않아 남아있는 경우가 있다. 이러한 노드가 있는 경우 필요하지 않는 자원을 소모하게 된다. 혹은 잘못된 URL을 입력하여 노드의 상태를 확인할 수 없는 경우가 생기게 된다. 이러한 경우를 확인하고 사용자가 대처할 수 있도록 알리기 위한 역할을 한다.

- `public boolean isValidEndpoint(Uri uri);`

위 함수와는 다르게 이는 “coap://192.168.1.1/gas” 와 같은 URL을 통해서 유효한 노드인지 확인 하는 역할을 한다.

3) 데이터

- `public String getSensorData(String nodeName);`

`getNodeName()`을 통해 받아온 노드의 이름 혹은 사용자의 리스트에 들어있는 노드의 이름을 입력하여 노드가 제공하는 데이터를 받아오도록 한다. 노드에서 제공하는 데이터는 temperature, gas, state등과 같은 다양한 데이터를 받을 수 있도록 String형으로 리턴 하도록 하였다.

- `public List<String> getNodedataFromDatabase(String nodeName);`

노드의 전체적인 데이터를 받아와 이를 분석하고자 할 때 쓰이는 함수이다. nodeName에 따른 데이터베이스의 결과를 받아와 List로 전달해 사용자가 데이터를 분석할 수 있도록 도와준다. “SELECT PAYLOAD FROM NODENAME_TABLE WHERE = ‘NODENAME’” 와 같은 쿼리를 날려 NODE 테이블 내의 PAYLOAD 전체 값들을 불러오고 이 값들을 하나씩 저장하여 리턴 한다.

4) 사용자모드

- `public boolean setAllUserMode(boolean mode);`

CoAP 노드는 사용자가 지정하는 모드로 사용할지 아니면 노드가 자동으로 작동하게 할 것인지 결정할 수 있다. 사용자 모드일 경우 사용자가 노드의 역할을 직접 컨트롤 할 수 있다. 예를 들어 전등의 스위치라고 한다면

사용자가 전등을 직접 끄고 켤 수 있도록 명령을 내릴 수 있다. 이와 같이 모든 노드에게 사용자가 직접 명령을 내릴 수 있는 상태로 만들어주는 역할을 하는 함수이다. 사용자 모드로 바꿀 것인지 True/False로 설정을 하고 이를 POST로 전송하게 된다. True 값을 POST로 전송 할 경우 사용다 가 켜고 끌 수 있는 모드로 되며 False값을 POST로 전송할 경우 사용자 모드는 꺼지게 된다.

- `public boolean setUserMode(boolean mode, String nodeName);`

다음은 특정 노드에 대해서 사용자모드로 변경할 것인지 정보를 전달하는 함수이다. nodeName을 통해서 특정 노드에게 사용자 모드로 변환하는 경우 True로 전송하고, 사용자 모드에서 일반 모드로 변환할 경우는 False로 데이터를 전송한다.

- `public void usermodeStateController(boolean mode, String nodeName);`

사용자 모드일 경우 mode에 True를 POST할 경우 특정 노드는 켜지게 되고, False를 POST할 경우 특정노드는 꺼지게 된다. 사용자가 스위치를 컨트롤 하고자 할 때 True를 전송하면 스위치의 상태는 On으로 변하게 되고, False를 보내게 되면 스위치는 Off상태로 변하게 된다.

- `public boolean setAlarm(String nodeName, float setLow, float setHigh);`

노드의 데이터 범위를 지정하여 특정 범위를 벗어난 경우 사용자에게 알림을 주도록 하는 함수이다. 가정환경을 가정하는 경우 아날로그 Gas Sensor(DFR0049)에서 가스레인의 가스 농도는 일반적으로 100~200 사

이이다. 실제로 가스를 누출시켜 측정했던 값은 급속도로 치솟아 900이상의 값을 넘겨주었다. 가스 누출 같은 급박한 상황에 처하게 된다면 사용자가 정해놓은 수치 이상의 값을 리턴 받을 시에 사용자에게 알림을 주어 쉽게 대응하도록 할 수 있다.

- `public boolean setAlarm(String nodeName, boolean setMode);`

스위치와 같은 역할을 하는 노드의 경우 nodeName을 통해 노드에 접근하여 setMode의 상태가 사용자가 정해놓은 True/False에 따라서 알림을 리턴 받는 역할을 하는 함수이다.

5) multicast

CoAP에는 Group Communication 기능을 제공한다. 이는 여러 개의 노드를 한 대 묶어 그룹화 하여 그룹에게 일괄적으로 데이터를 보내고 받을 수 있고 그룹으로 나뉜 노드들을 조작할 수 있도록 할 수 있다.

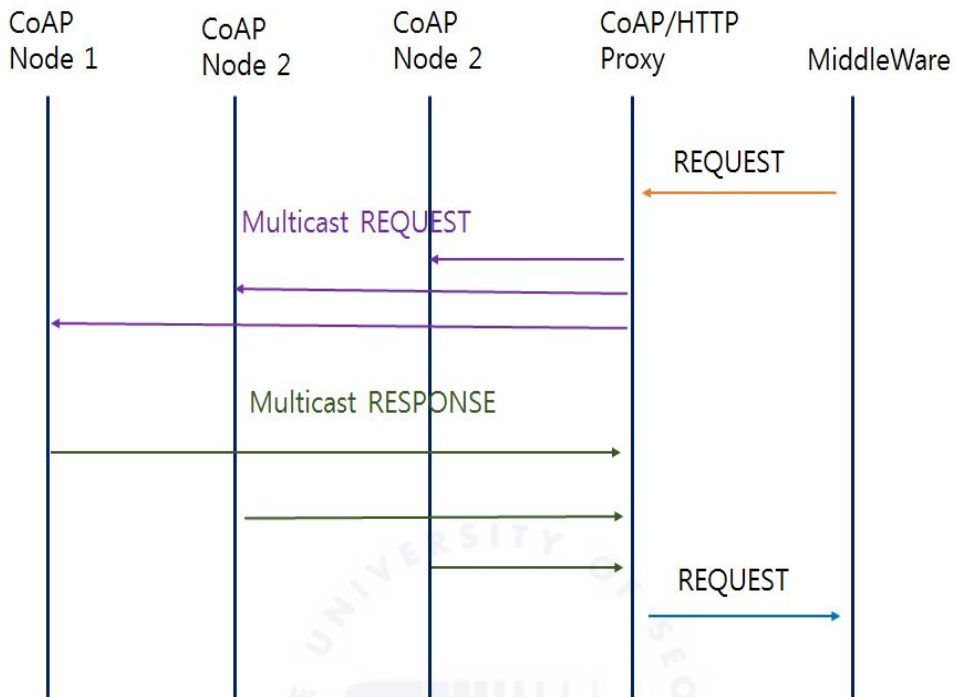


그림 3-4 CoAP Multicast

- `public void groupGenerator(String group);`

사용자가 그룹에 메시지를 보내기 위해서는 그룹을 생성해야한다. 이는 그룹을 관리하는 서버에 그룹을 생성할 수 있도록 요청을 보내는 함수이다. 그룹관리 서버에 그룹 이름을 전송하면 서버에서는 그룹의 이름으로 그룹을 생성하게 된다.

- `public void registGroupNode(String nodeName, String group);`

생성된 그룹에 노드들을 추가하는 함수이다. 그룹의 항목으로 비슷한 노드를 추가하거나 사용자의 목적에 맞는 그룹의 노드를 추가하여 관련 데이터를 수집하고 처리하는데 용이하게 할 수 있다.

- `public List<GroupNode> getGroupNode();`

사용자가 지정한 여러 가지 그룹들의 정보를 알아오는 역할을 하는 함수이다. 사용자가 생성한 그룹이 다양하게 존재하는 경우 사용자가 그 그룹의 리스트들을 받아와 확인하는 함수이다. 이는 List로 리턴을 받아 확인할 수 있다.

- `public List<String> getGroupInfo(String groupName);`

요청하는 그룹의 하위 항목으로 어떠한 노드들이 존재하는지 받아오는 함수이다. 하위항목으로 분류가 잘되었는지 확인을 하거나 하위항목의 노드를 확인하여 그에 관한 요청을 수행하도록 할 때 필요한 역할을 한다.

- `public List<String> groupGETRequest(String groupName);`
- `public List<String> groupPUTRequest(String groupName);`
- `public List<String> groupPOSTRequest(String groupName);`
- `public List<String> groupDELETERequest(String groupName);`

노드들에게 정보를 전달하거나 데이터를 전송 및 수신을 함께 있어 HTTP와 비슷한 포맷인 GET, PUT, POST, DELETE등의 메서드를 사용할 수 있다. 각각의 메서드에 따라서 CoAP 메시지의 Code 필드의 메서드를 설정하도록 한다.

제4장 성능 평가

API의 성능을 평가하기 위한 척도로 응답 속도를 측정하는 것을 첫 번째로 꼽을 것이다. 클라이언트의 요청을 받아 서버로부터 데이터를 전송받고 처리하는 것 까지 걸리는 시간은 API의 성능을 평가하는데 좋은 척도가 될 것이다. 그렇지만 서버와 클라이언트의 구현이 각각 다르기 때문에 제대로 된 비교를 수행하는 데에는 어려움이 있다. 따라서 응답 속도 항목은 평가에서 제외하였다.

두 번째 평가의 척도로 사용성에 대해 고려할 수 있다. javascript를 이용하여 각 기능의 작동에 대한 예를 구성하고 API를 사용한 경우와 사용하지 않는 경우를 서로 비교하는 방법이 있다. API를 사용하는 경우는 JSP를 이용하여 서버 내에서 동적으로 페이지를 생성하여 브라우저 내에서 각 기능에 대한 실행 결과를 보이는 방식으로 구현하였다.

센서 네트워킹을 하기 위해서는 먼저 센서를 등록한 과정을 거친다. 센서를 등록하기 위해서는 registerCoapNode함수를 이용한다. 다음으로 등록된 노드의 데이터를 얻는 과정이다. 데이터를 얻기 위해서는 getSensorData라는 함수를 이용하여 노드의 이름을 통해 관련 데이터를 얻어올 수 있다. 얻어온 데이터는 String값으로 반환되어 사용자의 요구에 따라 사용된다. 가스 센서와 같은 경우는 리턴값이 일정 수치 이상일 경우에는 사용자에게 알림 신호를 전송하고 일정수치 이하인 경우는 단순한 수치만 보여주도록 한다.

아래 표6은 새로운 센서노드를 추가할 경우 그 상황에 따른 처리 방법에 대한 비교부분이다. API를 사용하지 않는 방법으로 센서노드를 새로 등록할 경우 센서에 관한 정보를 서버로 보내어 관련 데이터를 처리하도록 해야 한다. 이 부분에서 DB와 연동하는 부분은 서버에서 새로 구현을 하고

그에 관하여 처리할 수 있는 페이지를 구성하는 추가적인 작업이 필요로 하다. 하지만 API를 이용할 경우 API에서 서버와 DB관련 정보를 입력한 뒤 원하는 정보를 함수를 통해 전달하게 된다면 손쉽게 새로운 노드를 추가할 수 있다. 센서노드를 등록하는 과정은 필수적인 세 가지 요소인 이름과 주소, 타입을 통해 손쉽게 등록이 가능하다. 반면 API를 사용하지 않을 경우는 해당 서버를 통해서 이와 관련된 데이터를 직접 전송하여 입력할 수 있도록 하며 관련 기능을 직접 구현해야하는 불편함이 있다.



Javascript	<pre> function register(){ var sensor_name = document.getElementsByName("sensorname")[0].value; var sensor_address = document.getElementsByName("sensoraddress")[0].value; var sensor_type = document.getElementsByName("sensortype")[0].value; if(sensor_name!=null && sensor_address!=null && sensor_type !=null){ jQuery.ajax({ type:"POST", url:"http://112.108.40.147/proxy/sensor/", data:{ name:sensor_name,address:sensor_address,type: sensor_type }, success:function(data){ } }); } </pre>
API	<pre> RegisterCoapNode register = new RegisterCoapNode(); String name = request.getParameter("sensorname"); String address = request.getParameter("sensoraddress"); String type = request.getParameter("sensortype"); if(sensorname!=null && sensoraddress!=null && sensortype !=null){ register.registerCoapNode(name, address, type); } </pre>

표 4-1 노드의 등록하는 과정에 대한 비교

Javascript	<pre>\$.ajax({ type: "GET", url: "http://112.108.40.147/proxy/192.168.100.20/gas", dataType: "json" })</pre>
API	<pre>String sensorData = data.getSensorData("gas");</pre>

표 4-2 데이터 호출에 관한 비교

위 테이블은 javascript와 API를 사용했을 경우 gas라는 노드에게 관련 데이터를 얻기 위해 호출을 하는 부분이다. API를 사용했을 경우 관련 노드에 관한 정보를 DB에서 호출하여 사용할 수 있기 때문에 호출하고자 하는 노드의 이름만 사용함으로써 직관적으로 이용할 수 있다. 리턴 받은 데이터는 sensorData라는 변수에 저장된다. 반면에 javascript를 이용할 경우 전송타입을 지정하고, 정보를 받아오기 위한 센서의 주소를 강제적으로 입력해야하는 번거로움이 있다. API를 사용한다면 이러한 번거로움을 줄이고 직관적이고 쉽게 원하는 노드의 정보를 받아올 수 있다.

JavaScript	<pre> .done(function(data) { \$("#g_gas").text(data.gas); if(data.gas>=700){ console.log('done gas'); \$("#gasleak .ui-btn-text").text("emergency!!!"); \$("#gasleak").attr("data-theme", "e").addClass("ui-btn-up-e"); } }); </pre>
API	<pre> if(Integer.parseInt(sensorData)<700) out.println("Safe"); else out.println("Emergency!!"); </pre>

표 4-3 데이터 결과의 사용에 대한 비교

다음으로 얻어온 데이터를 이용하여 상태에 따른 표시 정보를 적용하는 부분이다. 자바스크립트를 이용할 경우 HTML 내에 id나 name같은 부분을 지정하여 관련 데이터를 직접적으로 맵핑하여 입력하는 방식을 사용한다. 하지만 API를 사용하는 경우 HTML 내에서 원하는 부분에 대해 직접적으로 값을 입력할 수 있기 때문에 손쉽게 이용이 가능한 면이 있다.

세 번째 평가의 척도로는 코드 라인의 수를 고려할 수 있다. 코드 라인의 수를 비교하기 위하여 Web Server를 구동하여 Client에서 각각 서버노드들에게 호출하는 부분을 Ajax로 구현하여 웹페이지에서 호출하도록 하였다. 이와 비교하기 위해 같은 기능을 하는 웹페이지를 구현하고 API를 사용하여 Ajax를 이용하는 것과 같은 기능을 하는 API함수를 사용하여 클라이언트에 호출하여 결과 값을 불러오도록 하였다.

	일반 웹페이지	API를 사용한 경우
Lines of Code	61	40

표 4-4 전체 코드의 라인 수 비교

위 표의 결과는 페이지를 구성하는 전체 코드의 길이이다. 핵심 코드의 수를 비교하더라도 API를 사용한 경우가 월등히 짧다는 것을 확인할 수 있다. API를 사용한 방법이 기존의 방법보다 코드 작성에 있어 간단하고 짧고 쉽게 작성할 수 있음을 알 수 있다.



제5장 결론

IETF CORE 워킹그룹은 사물의 인터넷을 위한 통신 프로토콜을 표준화하기 위해 6LoWPAN 상위의 트랜스포트와 어플리케이션 계층의 사이에 CoAP을 위한 레이어를 두어 영역간의 통신을 하도록 하였다. 이 영역은 상위 어플리케이션 계층에서 노드들에게 이벤트를 요청하고 노드에서는 수신한 이벤트를 어떻게 전송하며, UDP 상에서 신뢰성 있고 단순한 프로토콜로서 적용이 가능하도록 설계되었다.

이와 관련하여 각 M2M 노드들 간의 통신을 연결하여 네트워크를 구성하고 그와 관련한 데이터를 가공하고 얻기 위해서는 CoAP의 동작방식과 message 내부의 부분들에 관한 이해가 필수적이다. 본 논문에서는 미들웨어를 통하여 CoAP의 동작방식과 CoAP Message에 관한 이해 없이도 사용자가 지정한 각 node의 이름을 통하여 이를 관리하고 리소스에 관한 정보를 얻어올 수 있도록 설계하였다.

앞으로 가정환경내의 모든 사물들 혹은 그 이외의 모든 사물들에게 각각의 IP가 부여되고 그 사물들이 모두 연결되어 있을 때 사물들이 가지는 고유의 이름과 주소만을 가지고 개발자가 각 노드들을 관리할 수 있는 미들웨어 표준이 제공된다면 손쉽게 이와 관련한 사물들의 기능을 통제할 수 있을 것이다. 따라서 Middleware API를 정의하고 이와 관련한 기능을 제공하는 것은 사물의 인터넷 환경에 쉽게 적용하여, IoT 관련 서비스 제공에 큰 도움을 줄 수 있을 것이다.

참 고 문 헌

- [1] 김대영 외(2005), 센서네트워크 운영체제/미들웨어 기술동향, [IITA] 정보통신연구진흥원 학술정보, pp.6-7
- [2] 김문권 외(2014), 상호 호환성 검증을 위한 IoT 기반의CoAP 프로토콜 구현 및 실험, 한국인터넷방송통신학회, pp. 7-8
- [3] 김영만(2004), 센서네트워크 미들웨어 구조 및 연구현황, 정보과학회지 제22권 12호, pp. 13-16
- [4] 민경주 외(2011), CoAP 프로토콜 구현과 USN 환경적용, 한국정보통신학회논문지 제 15권 제 5호, p.1190
- [5] 문성남(2012), 공장 내 장치 관리를 위한 센서네트워크 설계에 관한 연구, 숭실대학교, p.14
- [6] C. Bormann, M. Ersue, and A. Keranen, "Terminology for Constained-Node Networks", IETF RFC-7228, May 2014.
- [7] Gartner, "Analysts Examine Top Industry Trends at Gartner Symposium", Gartner Inc, Oct. 2014. p.1.
<http://www.gartner.com/newsroom/id/2867917>
- [8] Z. Shelby, K. Hartke, and C. Bormann, "Constrained Application Protocol(CoAP)", IETF RFC-7252, June 2014.
- [9] Matthias Kovatsch et. al, "Californium(Cf)",
<https://www.eclipse.org/californium/>
- [10] Olaf Bergmann,"libcoap: C-Implementation of CoAP",
<http://libcoap.sourceforge.net/>
- [11] SAMUEL R. MADDEN et.al, TinyDB: An Acquisitional Query

Processing System for Sensor Networks, ACM TODS, VOL.30, NO.1, 2005, pp.8-40

[12] Y. Yao and J.Gehrke. "The Cougar Approach to In-Network Query Processing in Sensor Networks," SIGMOD Record, Vol. 31, No. 3, Sept. 2002.

[13] C. Shen, C. Srisathapornphat, C. Jaikao, "Sensor Information Networking Architecture and Applications," IEEE Personal Communications, Vol.8, No.4, Aug. 2001, pp. 52-59.

[14] S. Li, S. Son, and J. Stankovic, "Event Detection Services Using Data Service Middleware in Distributed Sensor Networks," Int'l Workshop on Information Processing in Sensor Networks (IPSN'03), Palo Alto, CA, Apr. 2003.

[15] P. Eisenhauer, Markus Rosengren and P. Antolin, Hydra: A development platform for integrating wireless devices and sensors into ambient intelligence systems. Springer, 2010, pp. 367-373.

CoAP Middleware API Design

ABSTRACT

Hak Kwon

Dept. of Electronic & Electrical Computer Eng.

Graduated School of University of Seoul, Korea

Supervised by Prof. Young-gook Ra, Ph. D.

Developing the TCP/IP standard IBA(Internet Architecture Board) which is higher organization of IETF(Internet Engineering Task Force) decided to attach the TCP/IP protocol stack to small things, such as sensor. This means small object(thing) is also considered as on of the communication node, it has symbolic meaning that expand smart object has all five layer. To commercialize the smart object, the things are required own IP address on IPv6 environment and common service on application.

Under this background, IoT/WoT(Internet of Thing/Web of Thing) configure the intelligent network between objects based on the internet to communication was introduced. Things has own IP address on IoT environment and are smart object that a lot of people communicate over the internet on Application Layer. In other words, IoT is that smart object are commercialized space.

According to the interest of IoT, IETF establish CoAP(Constrained Application Protocol) for use as IoT protocol. CoAP is expected that standard protocols created by things is connected to the Internet protocol that can be used within a constrained environment. Accordingly, in this paper, we proposed Middleware API that can manage and collect the data of objects that connected CoAP network.

Keywords: CoAP, Constrained Application Protocol, Middleware, API, Sensor, Network, Protocol, IoT, Internet of Things



감사의 글

오랜 시간 대학생활을 해오면서 석사과정까지 마칠 시간이 어느덧 다가왔습니다. 짧은 기간이지만 2년 반가량 공부와 개발을 진행하면서 실력을 쌓아왔지만 여전히 부족함이 있는 것 같습니다.

대학원에 진학해서 저를 많이 지도해 주시고 가르침을 주신 나영국 교수님께 정말 감사를 드립니다. 연구실 생활을 하면서 많은 도움을 주시고 잘 챙겨주신 김미수 선배님께도 감사를 전합니다. 연구실 생활 처음부터 끝까지 함께하면서 형처럼 챙겨준 진경이, 지금의 내가 있도록 많이 가르쳐주고 롤모델이 되었던 영제, 항상 뭐든 믿음직스럽게 잘 해내는 경진이, 나의 개그를 잘 받아주며 웃음을 잃지 않았던 송현이 덕분에 연구실 생활을 잘 해나온 것 같습니다. 그리고 항상 옆에서 응원해 주고 도움을 준 세희, 같이 프로젝트를 하면서 도움을 많이 준 재홍이, 진로에 대해 고민을 들어주고 조언도 많이 해준 내친구 선태와 규태, 모두 정말 감사합니다.

무엇보다도 가장 많이 힘이 되어 주시고 응원도 기도도 많이 해 주셨으며 지금까지 그리고 앞으로도 저의 정신적인 기둥이신 부모님께 진심으로 감사를 드립니다. 항상 걱정해 주시고 염려해주시면서 경제적으로도 채워주신 형수님과 뒤에서 든든한 우군이 되었던 형 덕분에 대학생활을 잘 마무리 할 수 있었습니다. 어디서든 당당하고 자랑스러운 아들, 동생의 모습을 보여드리겠습니다.

2015년 6월 17일

권 학 올림