The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 17, No. 2, pp.119-126, Apr. 30, 2017. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2017.17.2.119
JIIBC 2017-2-18

CoAP 사용을 위한 미들웨어 API 설계

Middleware API Design for CoAP Usage

권학*, 나영국**, 조재덕*

Hak Kwon*, Younggook Ra, JaeDeok Jo***

요 약 TCP/IP 표준을 개발하는 IETF의 상위 기구인 IAB는 작은 사물에도 TCP/IP protocol stack을 붙이기로 결정했다. 이는 하나의 작은 사물도 하나의 통신 노드로 인정하여, 이를 다섯 개의 전 계층이 올라간 스마트 오브젝트로 확장하겠다는 상징적인 의미를 가진다. 이러한 배경 아래서 등장한 것이 웹을 기반으로 사물 간 지능적으로 네트워크를 구성해 통신하는 IoT/WoT이다. 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.

Abstract Developing the TCP/IP standard IBA which is higher organization of IETF decided to attach the TCP/IP protocol stack to small things, such as sensor. This means small object is also considered as on of the communication node, it has symbolic meaning that expand smart object has all five layer. Under this background, IoT/WoT 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 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.

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

Ⅰ. 서 론

1. 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 배경으론 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

*정회원, 서울시립대학교 전자전기컴퓨터 공학부
**정회원, 서울시립대학교 전자전기컴퓨터 공학부(교신저자)
접수일자: 2016년 8월 28일, 수정완료 2017년 2월 7일
게재확정일자 2017년 4월 7일

Received: 28 August, 2016 / Revised: 7 February, 2017 / Accepted: 7 April, 2017
**Corresponding Author: ygra123@gmail.com
Dept. Electronic & Electrical Compuer Eng., University of Seoul, Korea

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(Garthner,2015) [7]에서도 2015년 10대 전략 기술 동향에 사물의 인터넷을 뽑을 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/query와 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 통해 서로 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.

II. 관련 연구

1. CoAP

CoAP는 2010년 초부터 IETF(Internet Engineering Task Force) 워킹그룹(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]

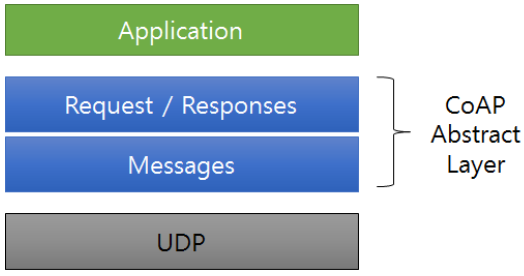

그림 1. CoAP 층
Fig. 1. CoAP layer

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.

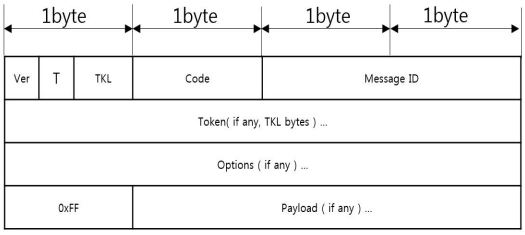

그림 2. CoAP 문자 형식
Fig. 2. CoAP Message Format

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), 5비트는 자세한 내용(detail)을 의미한다. 이 부분은 클래스의 값과 자세한 내용을 합하여 “c.dd”로 표현할 수 있는데 c는 0~7사이의 값이올 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.

Ⅲ. CoAP Network 기반 Middleware API 설계

1. 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.

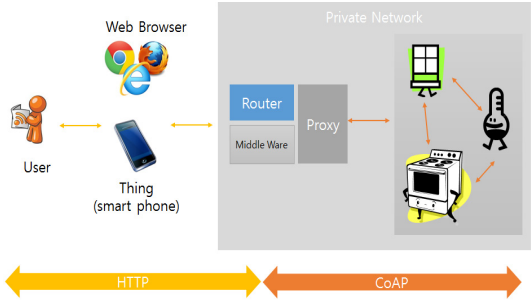

그림 3. CoAP-고객 구조도
Fig. 3. CoAP to Client Architecture

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

그림 3.과 같은 Home Network 상황을 가정 하였을 경우, 센서노드들은 각각 창문과 가스레인지, 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. 가스레인지는 가스센서, 창문은 온도센서, 보일러에는 온도센서

를 두어 실내의 온도가 급격히 높아지거나 가스가 누출되었을 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 실내외 온도, 습도를 확인할 수 있다.

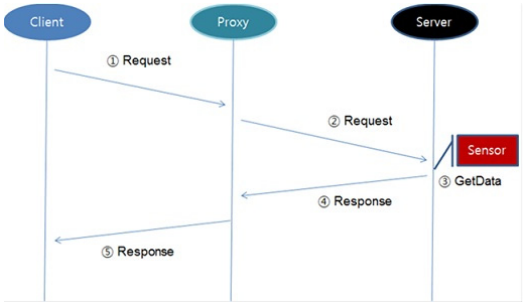

그림 4. CoAP 메시지 흐름도
Fig. 4. CoAP Message flow

그림 4.는 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.

2. 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 Server과 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 접근시 관련 노드들에 관한 정보를 탐색 할 수 있도록 하였다. CoAP도 HTTP와 같은 메서드를 사용하기 때문에 웹기반으로 구현하여 CoAP에 관한 이해를 돕고자 했다.

가. Client to Server 메시지 교환

a. Client to Proxy

표 1. 고객과 Proxy 간의 HTTP 헤더
Table 1. Client to Proxy HTTP Header

```
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: XML HttpRequest\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-Encording: gzip,deflate,sdch\r\n
Accept-Language: ko-KR, ko:q=0.8,en-US;q
=0.6,en;q=0,4\r\nCookie: 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]
```

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 데이터를 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.

b. Proxy to Arduino

표 2. Proxy 와 Node간의 CoAP 헤더
Table 2. Proxy to Node CoAP Header

```
192.168.100.5 192.168.100.20 COAP 71. CON, MID:13239,
GET, TKN:ab 04 /gas_=13779462142887
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
```

CoAP으로 변환된 패킷이 아두이노 노드로 향하는 모습이다. CoAP 프로토콜의 Type은 HTTP의 메서드로 정해졌다. HTTP GET 메서드의 인자로 따라 붙은 URL들은 코앱의 Uri-Path 옵션으로 분리되어 순서대로 삽입되었다. Uri-Query 옵션 역시 따로 분리되어 적용된 모습니다. 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.

c. Arduino to Proxy

표 3. Node와 Proxy간의 CoAP 헤더
Table 3. Node to Proxy CoAP Header

```
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
```

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.

d. Proxy to Client

표 4. Proxy 와 고객간의 HTTP 헤더
Table 4. Proxy to Client HTTP Header

```
HTTP/1.1 200 OK\r\n
cache-control: max-age=60\r\n
content-thpe:
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\t\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
```

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.

나. 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.

표 5. Middleware API 분류
Table 5. Middleware API 분류

분류	함수
등록	• registerCoapNode() • deleteCoapNode()
정보	• getWellknown() • getNodeName() • nodeSleep() • startCoapNode() • confirmationCoapNode() • isValidEndpoint()
데이터	• getSensorData() • getNodedataFromDatabase()
사용자모드	• setAllUserMode() • setUserMode() • usermodeStateController() • setAlarm()
멀티캐스트	• groupGenerator() • registGroupNode() • getGroupNode() • getGroupInfo() • groupGETRequest() • groupPUTRequese() • groupPOSTRequest() • groupDELETERequest()

이러한 함수들은 그림 5.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 에러메시지를 통해 사용자에게 알리도록 한다.

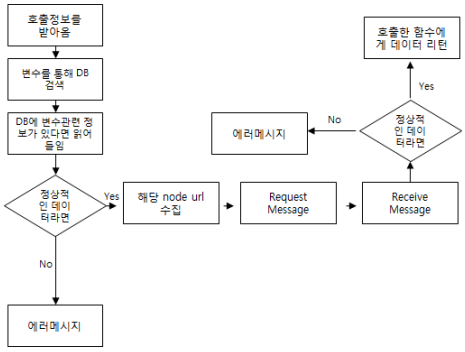

그림 5. 함수동작 과정
Fig. 5. Function State

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

Ⅳ. 결론

IETF CORE 워킹그룹은 사물의 인터넷을 위한 통신 프로토콜을 표준화 하기위해 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.

References

[1] Kim, Dae-Young et al. (2005), Sensor network operating system / middleware technology trend, [IITA] Journal of the Institute of Information Technology Advancement, pp.6-7

[2] Kim, Moon-Kwon et al. (2014), Implementation and Experiment of IoT-based CoAP Protocol for Interoperability Verification, Korea Internet Broadcasting Communication Association, pp. 7-8

[3] Kim, Young-Man (2004), Sensor Network Middleware Structure and Research Status, Journal of Information Science Vol. 22, No. 12, pp. 13-16

[4] Min, Kyung-Ju, et al. (2011), Implementation of CoAP protocol and application of USN environment, Korea Information and Communications Society Journal Vol. 15, No. 5, p.1190

[5] Moon Sung-nam (2012), A Study on Sensor Network Design for Device Management in Factory, Soongsil University, 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. Jaikeo,“Sensor Information Networking Architec-ture 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.

저자 소개

권 학(정회원)

- 2013년 서울시립대학교 격영학부 졸업
- 2015년 서울시립대학교 전자전기컴퓨터공학부 졸업
- 2015년 서울시립대학교 전자전기컴퓨터공학부 석사

<주관심분야 : CoAP, API Design>

나 영 국(정회원)

- 서울대학교 전자공학과 졸업
- 미국 Michigan Univ. 공학박사
- 서울시립대 전자전기컴퓨터 공학과 교수(현)

<주관심분야 : DB어플리케이션 제작 도구, XML과 관계형 데이터베이스 관련, 웹 어플리케이션 프레임워크>

조 재 덕(정회원)

- 서울시립대학교 전자전기컴퓨터공학부 재학중

※ 이 논문은 2015년도 서울시립대학교 학술연구조성비에 의하여 연구되었음