The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 14, No. 4, pp.7-12, Aug. 31, 2014. pISSN 2289-0238, eISSN 2289-0246

http://dx.doi.org/10.7236/JIIBC.2014.14.4.7
JIIBC 2014-4-2

상호 호환성 검증을 위한 IoT 기반의 CoAP 프로토콜 구현 및 실험

Implementation and Experiment of CoAP Protocol Based on IoT for Verification of Interoperability

김문권*, 김도현**

Wen-Quan JIN*, Do-Hyeun Kim**

요 약 IETF (Internet Engineering Task Force)의 CoAP (Constrained Application Protocol) 프로토콜은 작은 용량의 메모리와 저전력 등 제한된 환경에서 센서나 구동체 노드 간에 통신을 지원한다. CoAP 프로토콜은 HTTP와 쉽게 상호 변환할 수 있으며, 사물인터넷(Internet of Thing : IoT)와 M2M (Machine-to-Machine) 환경에서 저전력 센서와 구동체 네트워크를 통한 기반 시설을 감시하거나 관리할 수 있다. IETF CoRE(Constrained RESTful environments) 워킹그룹(Working Group)에서 2010년에 CoAP 프로토콜에 대한 표준화를 시작하여 최근에 RFC(Request for Comments) 7252로 발표한다 [2]. 본 논문에서는 이질적인 운영 환경에서 CoAP 프로토콜을 설계하고 구현하여 상호 호환성을 검증한다. 이를 위해 CoAP 클라이언트에는 윈도우 기반의 CoAP 프로토콜을 실현하고, CoAP 서버에는 리눅스 기반의 CoAP 프로토콜을 구현하여 상호 연동 실험을 실시하여 동작을 확인한다.

Abstract IETF (Internet Engineering Task Force) CoAP (Constrained Application Protocol) protocol is supported communication between sensor or actuator nodes by in a constrained environment, such as small amount of memory, and low power. CoAP and HTTP protocol can convert easily, and can use to monitor or controll the infrastructure utility through low-power sensor and actuator networks in IoT (Internet of Thing) and M2M (Machine-to-Machine) environment. IETF CoRE(Constrained RESTful environments) Working Group proposed CoAP protocol in 2010, and began to standardize it. Recently, CoAP protocol is published RFC (Request for Comments) 7252. In this paper, we design and implement of CoAP protocol for testing the interoperability in heterogeneous operating environments. For this experiment, we developed the CoAP client program based on Windows environment and CoAP server program in Linux environment to test the interoperability.

Key Words : CoAP protocol, IoT, M2M, HTTP protocol, IETR CoRE

Ⅰ. 서 론

최근 사물인터넷(Internet of Thing : IoT)와 M2M (Machine-to-Machine) 환경에서 제한된 성능을 갖는 센서 노드를 이용하여 온도, 습도, 광합성도, 기울기, 오염 등과 같은 정보를 획득하는 기술 개발이 활발히 진행되고 있다 [1]. 이와 같이 소형, 저전력 사물인터넷 환경에서 상황 데이터를 수집하는 서비스를 제공하는 응용 프로토콜 요구가 증가하고 있다 [5]. 이에 IETF(Internet Engineering Task Force) CoRE(Constrained RESTful

*준회원, 제주대학교 대학원 컴퓨터공학과
**종신회원, 제주대학교 컴퓨터공학과
접수일자 : 2014년 7월 4일, 수정완료 : 2014년 8월 4일
게재확정일자 : 2014년 8월 8일

Received: 4 July, 2014 / Revised: 4 August, 2014
Accepted: 8 August, 2014
**Corresponding Author: kimdh@jejunu.ac.kr,
Dept. of Computing Engineering, Jeju National University, Korea

environments) 워킹 그룹(Working Group)은 2010년초 CoAP 프로토콜 (Constrained Application Protocol)에 대한 국제 표준화를 시작하여 최근 RFC(Request for Comments) 7252 CoAP 프로토콜을 발표한다. 이에 제한된 센서나 구동체 네트워크에서 CoAP 프로토콜을 실현하기 위한 연구와 개발이 진행중이다. 본 논문에서는 이질적인 운영 환경에서 CoAP 프로토콜의 상호 연동을 실험하기 위해 윈도우와 리눅스 기반의 CoAP 클라이언트와 서버를 설계하고 구현하여 상호 호환성을 검증한다. 이를 위해 Californium 라이브러리를 이용한 윈도우 기반 CoAP 클라이언트와, Libcoap 라이브러리를 이용한 리눅스 기반 CoAP 서버를 구현한다. 그리고 CoAP 클라이언트와 CoAP 서버 간의 통신 실험을 통해 상호 연동을 확인한다 [3, 4].

본 논문의 구성은 다음과 같다. 먼저 2장에서는 IETF RFC CoAP 프로토콜에 대해 기술한다. 그리고 3장에서는 자료 구조와 시퀀스 다이어그램 중심으로 CoAP 프로토콜 설계한 내용을 소개한다. 다음으로 4장에서는 Californium 라이브러리기반 윈도우 CoAP 클라이언트와 Libcoap 라이브러리 기반 리눅스 CoAP 서버를 구현한 내용을 기술한다. 마지막으로 5장에서는 결론을 맺는다.

II. IETF RFC CoAP 프로토콜

IETF IETF 워킹그룹에서 2010년 초부터 본격적으로 개발되기 시작한 CoAP 프로토콜은 여러 차례의 변화를 거쳐 최근 RFC 7252을 발표한다. 여기서는 RFC 7252 CoAP 프로토콜의 핵심적인 내용을 기술한다. 그림 1에서 보는 바와 같이 축약된 메시지 헤더와 옵션 헤더를 갖는다.

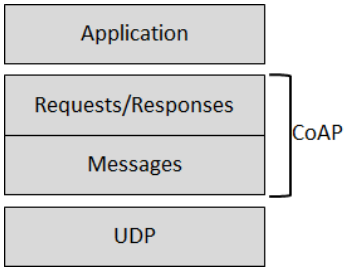


그림 1. CoAP 프로토콜 계층 구조
Fig. 1. Hierarchical Architecture of CoAP protocol

CoAP 프로토콜은 컴퓨팅 능력과 전력 소모 등의 제한된 자원을 갖는 센서와 구동체 노드를 고려하고 있다 [7]. 이를 위해 CoAP 프로토콜에서는 메모리 자원을 고려하여 데이터 전송 크기를 줄이고 있으며, 그림 2에서 보는 바와 같이 축약된 메시지 헤더와 옵션 헤더를 갖는다. CoAP 프로토콜은 UDP기반의 메시지를 이용하여 통신하며 [8], 이때 CoAP 프로토콜 메시지는 Binary format으로 인코딩된다 [10]. 그리고 CoAP 프로토콜 메시지는 4 byte 헤더로 시작되며 토큰(Token), 옵션(Option) 과 페이로드(Payload) 등이 있다.

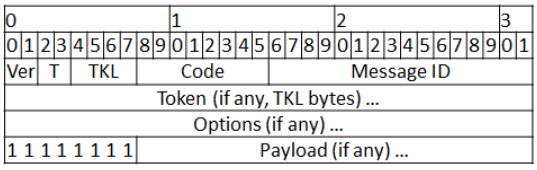


그림 2. CoAP 프로토콜 메시지 포맷
Fig. 2. Message format of CoAP protocol

CoAP 프로토콜 헤더에는 Ver (Version), T (Type), TKL, Code와 Message ID 가 있다. Ver은 버전을 의미하며 2-bit unsigned integer이고, T는 메시지타입을 의미하며 2-bit unsigned integer이다. 메시지 타입에는 Confirmable (0), Non-confirmable (1), Acknowledgement (2) or Reset (3) 가 있다. TKL은 Token 필드의 길이의 값을 의미하며, 4-bit unsigned integer이다. Code는 8-bit unsigned integer 이며, 3-bit Class와 5-bit detail로 구분한다. Code의 앞부분은 request (0), success response (2), 클라이언트 error response (4) 혹은 서버 error response (5) 등 값이며, Code값은 “C.DD” 포맷으로 되며 예를 들어 “0.00” 일 경우는 빈 메시지를 의미한다. Message ID는 메시지 중복을 맞추기 위해 Acknowledgement/Reset 타입의 메시지와 Confirmable/Nonconfirmable 타입의 메시지를 검사하며 16-bit unsigned integer이다.

헤더 뒤에는 토큰 값은 요청하고 응답을 연결을 의미하며, 0에서 8 byte로 된다. 그리고 옵션의 첫 번째 값은 이 메시지에 포함된 옵션의 개수를 의미한다. 다음 선택에 따라 페이로드 값이 있으며, 메시지에서 페이로드 앞의 1 byte의 값은 페이로드 Marker이며, 페이로드 값은 UDP 데이터그램의 마지막에 있다.

CoAP 클라이언트는 관리자 측에서 동작하는 노드로

서버에게 요청 메시지를 전달하고, 서버 측으로부터 응답 메시지를 수신하는 기능을 수행하는 노드이다. 이 노드는 그림 2에서와 같은 메시지 헤더를 생성하여 요청 메시지를 전달한다. "T"필드에 Confirmable(0), Non-Confirmable(1), Acknowledgement(2), Reset(3) 중 하나를 선택하여 메시지를 만들고, Code 필드는 GET(1),POST(2), PUT(3), DELETE(4)등을 선택한다. Content-Type, URI_SCHEME, URI-Authority 등과 같은 옵션을 필요한 경우 추가할 수 있다.

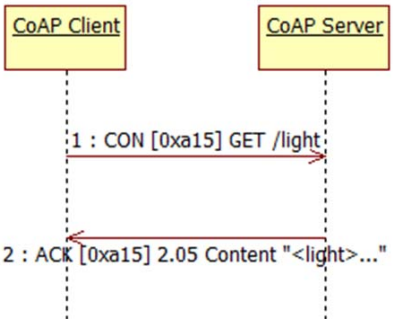


그림 3. CoAP 프로토콜 요청 및 응답
Fig. 3. Request and Response for CoAP protocol

CoAP 서버는 센서 노드나 정보수집 장치 측면에서 동작하는 노드를 의미한다. 클라이언트로부터 요청메시지를 수신한 경우, 메시지를 파싱 후 응답 메시지를 위해, 수신한 메시지를 복사하여 사용한다. 에러가 있는 경우, 현재 에러에 적합한 에러 코드를 사용하여 응답 메시지를 전송한다. 세 개의 네트워크 엘리먼트가 정상적으로 동작하기 위해서 트랜잭션 아이디가 필수적이기 때문에, 수신 메시지를 복사해서 사용하는 것이 일반적이다. 그림 3에서는 기본적인 CoAP 프로토콜 요청 및 응답 과정을 보여주고 있다.

Ⅲ. CoAP 프로토콜 분석 및 설계

서버와 리눅스 환경에서 CoAP 프로토콜을 구현하기 위하여 2개의 library를 이용하여 구현한다. 윈도우 기반 CoAP클라이언트는 Java 컴퓨터프로그램 언어 기반의 Californium 라이브러리를, 리눅스 기반 CoAP 서버는 C 컴퓨터프로그램 언어 기반의 Libcoap 라이브러리를 이용한다. Californium 라이브러리는 ETSI IoT CoAP 프로토콜이며, "3-clause BSD" license를 사용하고 있으며, Libcoap 프로토콜은 리눅스 C library를 사용하여 구현하며 "BSD" license를 사용한다. Libcoap 프로토콜은 TinyOS에도 포팅하고 있다.

CoAP 프로토콜 스택을 구현하기 위해 8, 16, 32비트 데이터에 대한 타입 정의와 2장에서 살펴본 메시지 헤더, 옵션 헤더를 위한 구조체가 필수적이다. 이러한 기본정의가 아래의 그림 4에 나타나 있다.

```
typedef struct {
  size_t max_size;
  coap_hdr_t *hdr;
  unsigned short max_delta;
  unsigned short length;
  unsigned char *data;
#ifdef WITH_LWIP
  struct pbuf *pbuf;
#endif

} coap_pdu_t;
typedef struct {
  unsigned int version:2; /* pr
  unsigned int type:2;    /* ty
  unsigned int token_length:4;
  unsigned int code:8;
  unsigned short id;     /* mess
  unsigned char token[];  /* th
} coap_hdr_t;
typedef struct {
  unsigned short delta;
  size_t length;
  unsigned char *value;
} coap_option_t;
```

그림 4. 기본 타입 및 구조체 정의
Fig. 4. Basic type and structures definition for CoAP protocol

유사한 방법으로 Code 필드에 사용할 메소드와 응답 코드를 위한 값에 대한 열거형 상수 정의가 필요하다. 이를 위해 프로토콜에서 제시한 대로 HTTP 에러 코드와 유사한 형태로 사용하기 위해 그림 5에서 보는 바와 같이, 필드 이름을 익숙한 HTTP 에러코드 형태를 따라 정의한다 [9].

클라이언트에서 서버에 메시지를 전송하면 서버에서는 클라이언트에 응답메시지를 전송한다. CoAP 프로토콜은 UDP상위 프로토콜로서 클라이언트에서 전송한 메시지가 서버에 도착하지 않을 경우가 있다. 이때 클라이언트는 서버로 부터 응답메시지를 받을 수 없다. 클라이언트에서 서버에 요청메시지를 전송하고 정해진 시간 내

에 클라이언트가 서버로 부터 응답 메시지를 받지 못하면 다시 요청메시지를 전송한다. 이와 같이 반복하여 전송하는데 전송하는 메시지의 메시지ID는 같은 것으로 전송하며 5번까지 전송하는데 계속 응답메시지를 받지 못하면 이번 통신은 실패한 것이다.

```
error_desc_t coap_error[] = {
  { COAP_RESPONSE_CODE(65),  "2.01 Created" },
  { COAP_RESPONSE_CODE(66),  "2.02 Deleted" },
  { COAP_RESPONSE_CODE(67),  "2.03 Valid" },
  { COAP_RESPONSE_CODE(68),  "2.04 Changed" },
  { COAP_RESPONSE_CODE(69),  "2.05 Content" },
  { COAP_RESPONSE_CODE(400), "Bad Request" },
  { COAP_RESPONSE_CODE(401), "Unauthorized" },
  { COAP_RESPONSE_CODE(402), "Bad Option" },
  { COAP_RESPONSE_CODE(403), "Forbidden" },
  { COAP_RESPONSE_CODE(404), "Not Found" },
  { COAP_RESPONSE_CODE(405), "Method Not Allowed" },
  { COAP_RESPONSE_CODE(408), "Request Entity Incomplete" },
  { COAP_RESPONSE_CODE(413), "Request Entity Too Large" },
  { COAP_RESPONSE_CODE(415), "Unsupported Media Type" },
  { COAP_RESPONSE_CODE(500), "Internal Server Error" },
  { COAP_RESPONSE_CODE(501), "Not Implemented" },
  { COAP_RESPONSE_CODE(502), "Bad Gateway" },
  { COAP_RESPONSE_CODE(503), "Service Unavailable" },
  { COAP_RESPONSE_CODE(504), "Gateway Timeout" },
  { COAP_RESPONSE_CODE(505), "Proxying Not Supported" },
  { 0, NULL }     /* end marker */
};
```

그림 5. 메소드 및 응답 코드 정의
Fig. 5. Method and responded codes definition

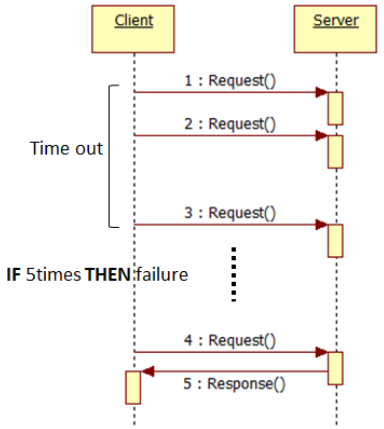


그림 6. CoAP 프로토콜 메시지 전송 시퀀스
Fig. 6. Message transfer sequence for CoAP protocol

그림 7에서는 클라이언트 동작을 위한 로직을 간단히 도시한 다이어그램을 보이고 있다. 효율적인 성능을 위해 Receiver, Sender, Retransmitter의 쓰레드로 구성된다. 지속적인 서비스와 서비스 지연이나, 레이싱 컨디션으로 인한 non-blocking을 위해 Receiver쓰레드는 waitForResponse()로 전송해오는 응답 메시지를 기다리고 메시지가 도착하면 이 메소드가 실행한다. Retransmitter쓰레드에서는 용청메시지가 전송된 때부터 응답메시지의 도착여부를 확인하는 메소드를 포함하고 있는데 그림 3과 같이 응답메시지가 전송되어 오지 않으면 이번통신이 실패한 것으로 확신된다.

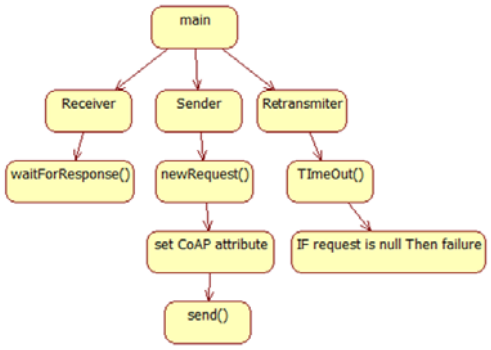


그림 7. CoAP 클라이언트 구조 및 다이어그램
Fig. 7. CoAP client structure and diagram

Ⅳ. 클라이언트와 서버에서 CoAP 프로토콜 구현 및 실험

제안한 이질적인 운영 환경에서의 CoAP 프로토콜을 구현하여 상호 호환성을 검증한다. 이를 위해 표 1은 CoAP 프로토콜을 구현하기 위한 개발환경이다. 클라이언트는 윈도우 운영체제에서 JRE(Java Runtime Environment)를 설치하여 Java 프로그램으로 작성한다. 사용한 개발도구는 eclipse를 사용한다. 서버는 우분트(Ubuntu) 리눅스 환경에서 C언어로 작성하며, 우분트가 기본으로 갖고 있는 C언어 컴파일환경인 GCC에서 실행한다.

CoAP 프로토콜은 응용계층 프로토콜로서 프로그램을 구현이 가능하다. CoAP 클라이언트는 Java로 구현하고, CoAP 서버는 C로 구현한다. CoAP 클라이언트는 CoAP 프로토콜을 통하여 CoAP 프로토콜 서버에 메시지를 전송하며 서버는 받은 메시지를 파싱하여 해당 기능을 실행하여 결과를 클라이언트에 전송한다. CoAP 클라이언트는 CoAP 프로토콜을 이용하여 상황 데이터를을 요청하며, 수신한 데이터를 처리하여 웹서비스를 통해 사용자에게 제공한다. 이때 CoAP 클라이언트는 Java 프로그래밍 언어를 사용하여 개발한다. CoAP 서버는

CoAP 프로토콜을 센서 노드에 구현하기 위해 마이크로 컴퓨터용 C 프로그래밍 언어로 개발한다. IETF 에서 제안한 CoAP 요청 메시지 포맷을 이용하며, CoAP 서버에서 파싱하고 분석한다.

표 1. CoAP 프로토콜 구현환경
Table 1. Implementation environment for CoAP protocol

개발환경		
환경	클라이언트	서버
운영체제	Windows 8.1	Ubuntu 12.4 (Linux)
실행환경	Java 7	gcc
개발도구	eclipse	gedit
프로그래밍 언어	Java	c

그림 8는 CoAP 클라이언트와 CoAP 서버 프로그램을 실행하는 환경이다. CoAP 클라이언트는 Java로 작성 되므로 JRE환경에서 실행한다. 여기서 운영체제와 관계없이 Java실행환경만 있으면 실행할 수 있다. CoAP 서버는 리눅스 C library를 사용하며 GCC 컴파일러가 있는 리눅스에서 실행할 수 있다.

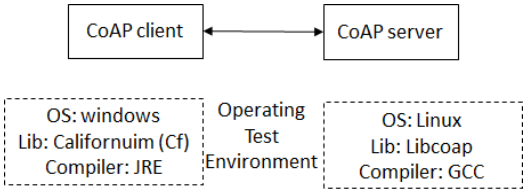


그림 8. 실험 환경
Fig. 8. Experiment environment for CoAP protocol

그림 9은 CoAP 클라이언트실행 결과이다. Californium 라이브러리를 사용한 CoAP 프로토콜 클라이언트가 CoAP 서버에 CoAP 프로토콜을 통하여 요청 메시지를 전송하면 CoAP 프로토콜 서버에서 처리하여 응답메시지를 CoAP 프로토콜 클라이언트로 전송한다. 이 그림에서는 요청한 메시지와 다시 받은 응답메시지를 보여 주고 있다. “CON”는 CoAP 프로토콜 메시지 포맷에서의 T (Type)의 값이고, “GET”은 CoAP 프로토콜의 메소드 타입이며 IETF에서는 4가지 메소드 타입 정의하고 있다. “MsgId: 10658”에서 “10658”은 메시지의 ID이고, 이 값은 자동으로 생성되거나 개발자가 정할 수 있다. “#Options: 1”는 CoAP 프로토콜메시지에 포함된 메시지 속성의 개수를 의미하는데 기본으로 “MsgId” 속성이 있다.

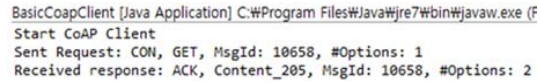


그림 9. CoAP 클라이언트 실험 결과
Fig. 9. Experiment results of CoAP client

그림 10는 CoAP 서버에서 노드 에뮬레이터의 실험 화면이다. 이 프로그램은 리눅스 환경에서 Console로 실행 되였으며 이 그림에서 보여주는 화면은 CoAP 프로토콜노드에서 메시지를 받고 가상온도환경에서 가상온도를 수집하는 상태를 Console로 보여 주고 있다.

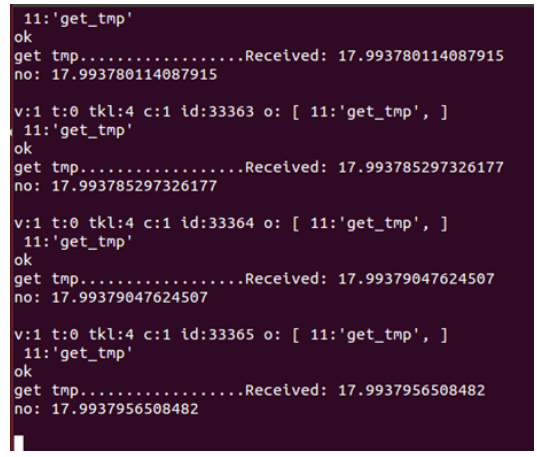


그림 10. CoAP 서버 실험 결과
Fig. 10. Experiment results of CoAP server

V. 결 론

센서 네트워크는 전력, 메모리, 컴퓨팅 처리 능력 등에서 제한된 하드웨어 한계를 극복하기 위한 연구가 진행되고 있다. 더불어 센서 노드를 이용하여 사물을 모니터링하고, 효율적으로 관리하기 위해 센서 네트워크의 표준화된 응용 프로토콜이 요구된다. 이를 위해 IETF 워킹그룹에서 CoAP 프로토콜을 국제 표준화하고 있다. 본 논문에서는 CoAP 클라이언트에는 Californium 라이브러리를 이용하여 윈도우 기반의 CoAP 프로토콜을 실현하고, CoAP 서버에는 Libcoap library를 사용하여 리눅스 기반의 CoAP 프로토콜을 구현하고, 서로 데이터 전송을 통해 상호 연동 실험을 실시하여 동작을 확인한다. 이때 CoAP 클라이언트는 현재 웹 응용 프로그램에서 사용되는 Java 컴퓨터 프로그래밍 언어로, CoAP 서버는 임베

디드 시스템에 사용되는 C 언어로 구현한다. 이질적인 운영환경에서 CoAP 프로토콜 데이터를 전송하여 상호 호환성을 검증할 수 있었다. 향후 사물인터넷 환경에서 이질적인 다양한 CoAP 프로토콜의 성능을 평가하고자 한다.

References

[1] Kyoung-Ju Min, Sang-Geun Yoo, Young-Woon Kim, Hyung-Jun Kim, Hoe-Kyung Jung, "CoAP Protocol Applying Method for Efficiency USN Management," The journal of the Korea Institute of Maritime Information & Communication Sciences Vol.14 No.2, pp. 507-509, 2010.

[2] Z. Shelby, B. Frank, D. Sturek, "Constrained Application Protocol (CoAP)", RFC 7252, June, 2014.

[3] http://libcoap.sourceforge.net," Libcoap: C-Implementation of CoAP ", 2014.

[4] https://github.com/mkovatsc/Californium, "Californium (Californium Library) CoAP framework - Java CoAP Implementation", 2014.

[5] Kyoung-ju Min, Yong-woon Kim, Sang-keun Yoo, Hyoung-jun Kim, Heo-kyung Jung, "Implementation of CoAP Protocol for USN Environment," The journal of the Korea Institute of Maritime Information & Communication Sciences, Vol. 15, No. 5, pp. 1189-1197, May, 2011.

[6] Sun-Jin Oh, "Design of a Fault-Tolerant Routing Protocol for USN," The Journal of the Institute of Internet, Broadcasting and Communication, Vol. 9 No. 2, pp. 51-57, 2009.

[7] Z. Shelby, "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August, 2012.

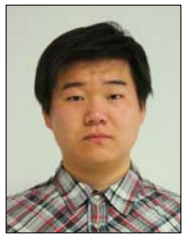
[8] J. Postel, "User Datagram Protocol", RFC 768, August, 1980.

[9] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June, 1999.

[10] K. Kuladinithi, O. Bergmann, T. P¨otsch, M. Becker, and C. G¨org. "Implementation of CoAP and its Application in Transport Logistics," In Proceedings of the 10th International Conference on Information Processing in Sensor Networks (IPSN 2011), Chicago, USA, Apr. 2011.

저자 소개

김 문 권(준회원)



- 2008년 9월 ~ 2012년 6월 : 중국 연변과학기술대학교 전산학과 학사
- 2013년 9월 ~ 현재 : 제주대학교 대학원 컴퓨터공학과 석사

김 도 현(종신회원)

- 1990년 3월 ~ 1995년 3월 : 국방과학연구소 전산망연구실 연구원
- 2004년 9월 ~ 현재 : 제주대학교 공과대학 컴퓨터공학과 교수
- 2013년 1월 ~ 현재 : 대한전자공학회 M2M/IoT 연구회 회장

"이 논문은 2014학년도 제주대학교 학술진흥연구비 지원사업에 의하여 연구되었음" 교신저자 : 김도현