CoAP 프로토콜 구현과 USN 환경 적용

민경주* · 김용운** · 유상근** · 김형준** · 정회경***

Implementation of CoAP Protocol for USN Environment

Kyoung-ju Min* · Yong-woon Kim** · Sang-keun Yoo** · Hyoung-jun Kim** · Heo-kyung Jung***

이 논문은 지식경제부의 지원을 받은 정보통신표준화 및 인증지원 사업의 연구 결과로 수행되었음

요 약

제한된 자원을 갖는 센서 노드를 효율적으로 관리하기 위해, IETF WG에서 2010년에 CoAP 프로토콜을 발표하였다. 아직 RFC로 확정 받지 않은 상태이기 때문에, 이 프로토콜의 단점을 보완하고, 논리적인 검증이 필요하다. 본 논문에서는 PC환경에서 CoAP 프로토콜을 구현하고, 프로토콜의 신속한 논리적 검증을 확인한 후, 이를 기반으로 하여 실제 센서 노드 및 네트워크 노드에 적용한다. 메모리와 컴퓨팅 능력 등이 PC 환경과 비교하여 크게 제한되기 때문에, 센서 노드에 적용하기 위해서는 하드웨어에 종속적인 환경 적용이 필요하다. 이를 위해 Cygwin 환경에 포팅하고, 하드웨어 종속성을 해결하기 위한 방법 등을 제시하고, 이를 실험을 통해 검증한다.

ABSTRACT

To manage sensing information such as temperature, humidity and so on efficiently, it is need to use special purpose protocol. In this reason, IETF WG proposed CoAP protocol, and it is on Internet draft. If it is possible to work on a specific protocol, sensor end-nodes and network devices will be managed efficiently. However, end-nodes have restricted resources, it is hard to applying to CoAP protocol directly. In this paper we analyse a CoAP protocol stack for USN. To verify this protocol quickly, at first we implemented CoAP protocol stack over PC environments. After the logical verification, we applied this protocol to the USN environment. To do this, we ported CoAP protocol to Cygwin environment, and proposed solutions for hardware dependencies, and it is verified through experiments.

키워드

CoAP 프로토콜, USN, 센서, TinyOS, Cygwin, NesC

Key word

CoAP Protocol, CoAP-Lite, USN, Sensor, TinyOS

* 준회원 : 배재대학교
** 준회원 : 한국전자통신연구원 표준연구센터
*** 종신회원 : 배재대학교 (교신저자, hkjung@pcu.ac.kr)

접수일자 : 2011. 02. 01
심사완료일자 : 2011. 03. 07

Ⅰ. 서 론

원격에 있는 소형, 저전력(Low-Powered)의 제한된 성능을 갖는 센서 노드를 이용하여 온도, 습도, 광합성도, 기울기, 오염 등과 같은 정보를 획득하는 기술 개발이 활발히 진행되고 있다. 이러한 센싱 정보의 제어를 위해 각각의 응용 서비스가 요구하는 표준화된 프로토콜 개발의 필요성이 대두됨에 따라, IETF(Internet Engineering Task Force) WG(워킹 그룹: Working Group)에서 2010년 초에 CoAP (Constrained Application Protocol)을 개발하고, 몇 차례의 수정을 거쳐, 2010년 말 최종 드래프트가 발표되었고, 2011년을 목표로 국제 표준화 작업이 진행중이다[1,2]. 이에 본 논문에서는 현재 추진 중 인 인터넷 드래프트의 논리적 검증과 센서 노드에 적용하기 위하여, CoAP 프로토콜을 구현하고, 이를 적용하는 실험을 진행하였다. 최종적으로 CoAP 프로토콜이 지향하려고 하는 센서 노드에 적용하기 위해, PC 환경에서 빠르게 드래프트에 충실하게 프로토콜을 구현하여 논리적인 검증을 거친 후, 이 프로토콜을 실제 센서 노드에 적용하였다.

본 논문의 구성은 다음과 같다. 2장에서는 CoAP 프로토콜 드래프트의 핵심 내용 및 소프트웨어 설계에 필요한 분야를 기술한다. 3장에서는 프로토콜을 PC환경에서 구현하기 위한 구조체 및 알고리즘과 이를 통해 구현된 결과를 기술한다. 4장은 PC환경에서 개발된 프로토콜을 센서 노드에 적용하기 위한 토폴로지 및 결과에 대해 다룬다. 결론 및 향후 연구 과제는 5장에서 기술한다.

Ⅱ. 관련 연구

IETF WG에서 2010년 초부터 본격적으로 개발되기 시작한 CoAP 프로토콜은, 여러 차례의 변화를 거쳐 2010년 말에 현재의 프로토콜 드래프트가 배포되었고, 이전 버전에 비해 구조적인 변화가 버전2, 3을 통해 이루어졌다. 본 장에서는 드래프트를 이용한 기존연구와 프로토콜의 핵심적인 내용을 중심으로 살펴보도록 한다.

2.1. IETF WG : CoRE

IETF WG의 드래프트의 논리적 검증을 위해 CoRE에서는 C언어 기반의 CoAP 프로토콜을 라이브러리로 제공하는 libcoap 프로젝트가 진행되고 있다[3]. RFC (Request for Comments] 이전의 드래프트 상태인 CoAP 프로토콜 스택을 직접 구현하는 사례는 없고, libcoap을 TinyOS에 포팅하는 수준에 그치고 있다[4].

2.2. CoAP 메시지 포맷

CoAP 프로토콜은 제한된 리소스를 갖는 센서 노드 등에 적용하기 위한 목적으로 개발되고 있는 프로토콜이다. 그렇기 때문에 메모리를 비롯한 컴퓨팅 능력이 PC 환경과 비교하여 빈약하기 때문에 많은 제한을 고려하여 설계되었다. 특히 CoAP 프로토콜은 데이터 전송 사이즈를 줄이려는 시도가 많은데, 이를 위해 그림1, 2에서 보는 바와 같이, 매우 단순한 메시지 헤더와 옵션 헤더를 갖는다.

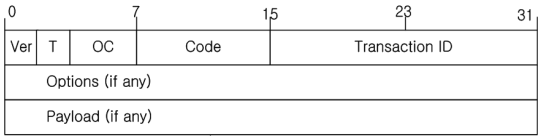

그림 1. CoAP 헤더 포맷
Fig. 1 CoAP Protocol Header Format

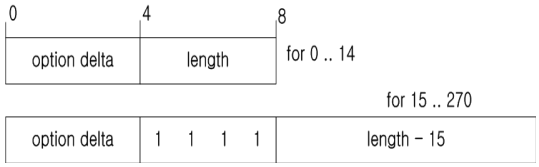

그림 2. 헤더 옵션 포맷
Fig. 2 CoAP Header Option Format

그림 1에서는 CoAP 버전 정보를 위해 2비트의 무부호 정수(unsigned integer)를 사용하고, 현재 버전을 위해 “1”로 설정한다. 유사하게, 2비트의 “T″는 ”Transaction Type″을 의미하고, 4비트의 “OC″는 ”Option Count″를 의미한다. 이는 트랜잭션 아이디 이후의 옵션 필드의 개수를 의미한다. Code 필드는 요청 메시지와 응답 메시지에 따라 그 의미를 달리 하는데, 각각 메소드와 응답 코드로 사용한다. 요청 메시지인 경우에는 GET, POST, DELETE, RESET 메소드 중 하나를 설정하고, 응

답 코드인 경우, HTTP 에러 코드와 유사한 형태의 메시지 구조를 갖는다[2]. 그림 2에서는 옵션의 길이에 따라 두 가지 형태의 메시지 타입을 갖는데, 이는 전술한 바와 같이, 전송 메시지 길이를 최소화하기 위한 용도로 구분한다.

2.3. CoAP 엘리먼트 노드

CoAP 프로토콜은 크게 서버, 프락시, 클라이언트 세 가지 형태의 네트워크 구성요소를 갖는다. 일반적으로, 서버 노드는 클라이언트로부터 요청 메시지를 수신 후, 응답메시지를 클라이언트에게 전해주는 기능을 수행하고, 클라이언트는 서버에게 요청 메시지를 전달하는 목적으로 사용되는 엘리먼트이다. 드래프트에서는 관리적인 측면의 서버-클라이언트 개념과 네트워크 측면의 개념이 혼용되고 있어서, 본 논문에서는 네트워크 모델에서의 서버-클라이언트 개념을 사용한다.

2.3.1. 클라이언트

클라이언트는 관리자 측에서 동작하는 노드로, 프락시를 통해 서버에게 요청 메시지를 전달하고, 서버 측으로부터 응답 메시지를 수신하는 기능을 수행하는 노드이다. 이 노드는 그림 1에서와 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.

2.3.2. 프락시 서버

프락시는 서버와 클라이언트 사이에서 데이터를 수신하여 메시지를 전달하는 기능을 수행하는 노드이다. 센서 노드의 자원이 취약하기 때문에, 상대적으로 자원이 풍부한 이 프락시 서버는, 재전송 기능을 수행하기에 적합한 노드이다. 주목적이 메시지의 전달이기 때문에, 프락시 서버는 수신된 메시지로 부터 정확한 목적지를 알아야 한다. 목적지를 알기 위해, URI_SCHEME 옵션 정보를 이용한다. 수신한 메시지의 URI_SCHEME 정보에서 목적지를 추출한 후, 전달한 목적지로 변경하여 메시지가 전달된다. 응답 메시지에 대해서도, 요청 메시지와 동일한 형태로 동작한다.

2.3.3. 서버

서버는 센서 노드나 정보수집 장치 측면에서 동작하는 노드를 의미한다. 프락시와 유사하게, 프락시를 통해 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.

Ⅲ. CoAP 프로토콜 구현

CoAP 프로토콜을 구현하여, 논리적인 검증을 하기 위하여, 표 1과 같은 환경을 이용하였다. 표 1에서 보는 바와 같이, 서버는 클라이언트 쪽에서 요청 메시지를 수신하는 경우, PC 환경에 센서 디바이스가 없기 때문에 실제 값이 아닌, 랜덤 값을 사용하여 프로토콜 동작을 논리적인 검증을 확인을 하였다.

3.1. 자료 구조 설계

3.1.1. 기본 정의 및 구조체 정의

CoAP 프로토콜 스택을 구현하기 위해 8, 16, 32비트 데이터에 대한 타입 정의와 2장에서 살펴본 메시지 헤더, 옵션 헤더를 위한 구조체가 필수적이다. 이러한 기본 정의가 아래의 그림 3에 나타나 있다.

표 1. 구현 환경
Table. 1 Development Environment

환 경	설 명	비 고
운영체제	CentOS 5.x Linux	서버, 클라이언트 프락시
메시지 생성기	APM	클라이언트
사용 모듈	재전송	On/Off
수집 정보	랜덤 값 사용	서버
바이트 오더링	바이트 오더링 선택	리틀 인디언

```
typedef unsigned char   uint8;
typedef unsigned short  uint16;
typedef unsigned int    uint32;
typedef struct  _option_header {
#ifdef  __BIG_ENDIAN__
      uint8    delta:4,
                        length:4;
#else
      uint8    length:4,
                        delta:4;
#endif
      uint8    length2;
} option_header_t;
typedef struct  _coap_header {
#ifdef __BIG_ENDIAN__
      uint8    ver:2,
                        type:2,
                        option_count:4;
#else
      uint8    option_count:4,
                        type:2,
                        ver:2;
#endif
      uint8    code;
      uint16  tid;
} coap_header_t;
typedef struct  _coap {
      coap_header_t   hdr;
      //option_header_t options;
      uint8    op_data[256];
} coap_t;
```

그림 3. 기본 타입 및 구조체 정의
Fig. 3 Definition of Basic Types and Structure

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 4에서 보는 바와 같이, 필드 이름을 익숙한 HTTP 에러코드 형태를 따라 정의하였다.

```
enum{
            METHOD_GET = 1,
            METHOD_POST,
            METHOD_PUT,
            METHOD_DELETE
};
enum {
      COAP_CODE_100   = 40,
            // 0x28 Continue
      COAP_CODE_200   = 80,
            // 0x50 OK
      COAP_CODE_201   = 81,
            // 0x51 Created
      COAP_CODE_304   = 124,
            // 0x7C Not Modified
      COAP_CODE_400   = 160,
            // 0xA0 Bad Request
      COAP_CODE_404   = 164,
            // 0xA4 Not Found
      // .. 이하 생략
};
```

그림 4. 메소드 및 응답 코드 정의
Fig. 4 Definition of Methods and Response Code

3.1.2. 재전송을 위한 구조체 정의

재전송 알고리즘의 사용 유무는 컴파일 시 옵션 선택 여부에 의해 결정되는 구조를 갖도록 구성하였다. 클라이언트는 서버 측으로 부터 응답메시지를 기다리지만, 정해진 시간동안 응답이 되돌아오지 않는 경우, 프락시 또는 서버 방향으로 저장된 이미 전송된 메시지를 재전송하게 된다. 이러한 재전송을 위해서는 기존에 전송한 메시지를 저장하기 위해 다음의 그림 5와 같은 형태로 정의하였다.

```
#ifdef USE_RETRANSMIT
typedef struct  _sent_info {
      struct sockaddr_in who;
      coap_t  coap;
      uint8   size;
      uint8   counter;
      uint16  tid;
} sent_info_t;
#endif
```

그림 5. 재전송을 위한 구조체 정의
Fig. 5 Structure Definition for Retransmission

3.2. CoAP 프로토콜 메시지 플로우

재전송 모듈이 활성화되어 사용되는 경우, 아래의 그림 6은 CoAP 프로토콜의 메시지 전달과정을 보여주고 있다.

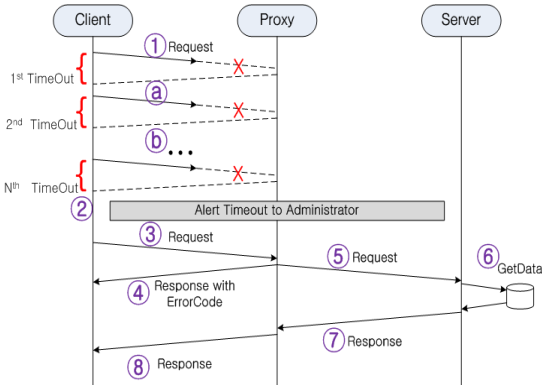

그림 6. 재전송 알고리즘 및 메시지 전달과정
Fig. 6 Retransmission Algorithm and Message Flow

클라이언트가 "confirmable" 요청 메시지를 전송한 경우, sent_info_t의 값을 설정하고, 서버 측으로 부터 응답 메시지를 기다린다. 정해진 타임아웃 시간 내에 응답 메시지가 도착하지 않는 경우, 그림 6의 ⓐ, ⓑ에서와 같이 sent_info_t의 데이터를 참조하여, 재전송을 하게 된다. 전송 목록에 없는 경우, 타임아웃이 처음 발생한 것으로 판단하여, 목록에 추가한다. 전송 또는 재전송한 데이터가 에러 없이 진행하면 그림의 ③, ⑤, ⑥, ⑦, ⑧의 절차를 거쳐 응답 메시지를 수신한다. 만약 반복적으로 응답 메시지가 되돌아오지 않는 경우, sent_info_t 리스트의 해당 데이터를 찾아 counter 필드 값만 증가시킨다. 이 전송 회수가 정해진 횟수(N)를 초과하는 경우(②), 전송 목록에서 삭제 후, 관리자에게 알리는 것과 같은 에러처리를 한다.

3.3. 데이터 처리

3.3.1. 클라이언트

그림 7에서는 클라이언트 동작을 위한 로직을 간단히 도시한 다이어그램을 보이고 있다. 효율적인 성능을 위해 Receiver(), Sender(), Retransmitter(), ProcessReceiver()의 쓰레드로 구성된다. 지속적인 서비스와 서비스 지연이나, 레이싱 컨디션으로 인한 non-blocking을 위해 Receiver() 쓰레드는 ProcessReceiver()라고 명명된 에이전트 쓰레드를 생성한다. Retransmitter() 쓰레드의 사용 유무는 컴파일 시 재전송 알고리즘 선택 유무에 따라 결정된다.

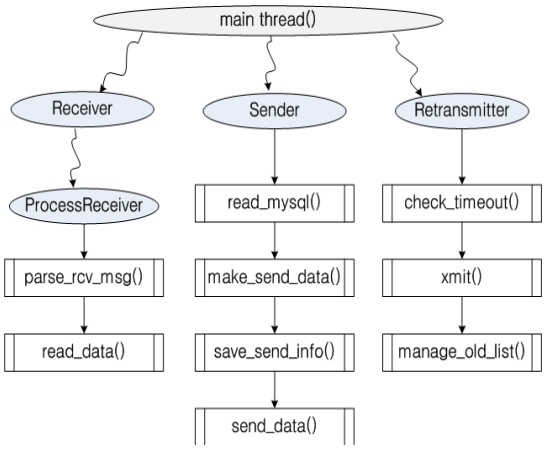

그림 7. 클라이언트 구조 및 다이어그램
Fig. 7 CoAP Client S/W Structure and Diagram

3.3.2. 프락시

ProcessReceiver() 쓰레드는 그림 8의 왼쪽 프락시에서 보는 바와 같이, 무한히 반복하면서, 데이터를 기다리는 역할을 수행한다. 데이터를 수신한 경우, 수신 데이터를 파싱한 후, 데이터를 전송할 것인지, 송신측에 에러 메시지를 전송할 것인지 결정한다. 에러가 없어 전송하려고 하는 경우, 목적지 결정을 위해, 수신 메시지의 URI_SCHEME 정보를 추출한다. 적당한 목적지를 알 수 없는 경우에는 송신자 측에 에러 메시지를 전달한다.

3.3.3. 서버

그림 8의 우측 서버에는 서버의 구조 및 로직을 간단히 도시하였다. 프락시 서버의 쓰레드의 동작과 유사하게, 수신 데이터를 파싱 후, 서버는 온도, 습도, 기울기 등 필요한 센싱 정보를 수집한다. 수집된 정보를 응답 메시지에 포함시키기 위해, 페이로드를 이용한다. 이 노드는 USN에 적용되면, 최종 센싱 노드나 데이터 수집 장치가 역할을 수행한다.

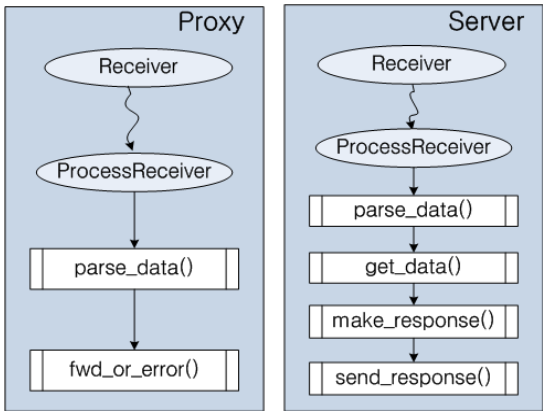

그림 8. 프락시 및 서버 구조 및 동작
Fig. 8 S/W Structure and Diagram of Proxy and Server

3.4. 프로토콜 구현 결과

클라이언트용 메시지 생성기 구현을 위해, PHP로 웹과 호환되도록 구현하였다. 그림 9는 요청메시지 생성기를 웹으로 구현한 화면의 일부를 보여주고 있다. 프로토콜 스택을 빠르게 구현하여, 프로토콜의 논리적인 검증을 위한 방법으로 사용하였다. 이는 접근성과 테스트의 용이함 및 최근 스마트폰이 보편화됨으로 인해, 관리를 보다 효율적으로 할 수 있는 장점이 있다.

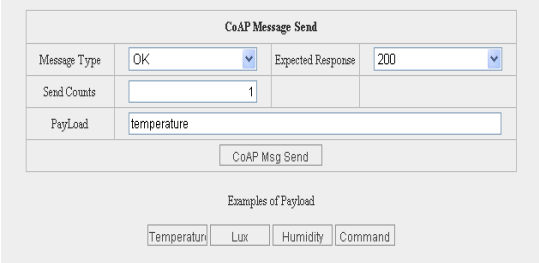

그림 9. 요청 메시지 생성기(웹 스크린 샷)
Fig. 9 Request Message Generator (Web)

그림 10, 11, 12는 각각 클라이언트, 프락시, 서버측에서의 데이터 덤프 메시지를 보이고 있다. 이는 클라이언트-프락시-서버 모델로 구성된 가장 일반적인 형태의 구조에서 데이터 플로우이다. 이는 그림 6의 ③, ⑤, ⑥, ⑦, ⑧의 절차를 통해 데이터 전송이 이루어지는 경우를 의미한다.

```
=====================================
      SND :COAP DATA :46 B
=====================================
     42(B) 01() 08) 52(R) 9B(?
     74(t) 65(e) 6D(m) 70(p) 65(e)
     72(r) 61(a) 74(t) 75(u) 72(r)
     65(e) 3F(?) 1C() 63(c) 6F(o)
     61(a) 70(p) 3A(:) 2F(/) 2F(/)
     32(2) 30(0) 33(3) 2E(.) 32(2)
     35(5) 30(0) 2E(.) 31(1) 33(3)
     33(3) 2E(.) 31(1) 37(7) 32(2)
     3A(:) 31(1) 31(1) 31(1) 31(1)
     31(1)
=====================================
       RCV FROM SERVER :40 B
=====================================
     61(a) 50(P) 08) 52(R) 3F(?)
     1C() 63(c) 6F(o) 61(a) 70(p)
     3A(:) 2F(/) 2F(/) 32(2) 30(0)
     33(3) 2E(.) 32(2) 35(5) 30(0)
     2E(.) 31(1) 33(3) 33(3) 2E(.)
     31(1) 37(7) 32(2) 3A(:) 32(2)
     32(2) 32(2) 32(2) 32(2) 34(4)
     33(3) 2E(.) 35(5) 20( ) 43(C)
```

그림 10. 클라이언트 메시지 덤프
Fig. 10 Client Message Dump

프락시 서버는 목적지 결정을 위해, URI_SCHEME을 이용하기 때문에, 클라이언트는 “coap://”로 시작하는 목적지 정보를 페이로드에 추가한다[2]. 서버 메시지 덤프의 마지막 부분에는 온도를 정보가 페이로드에 추가된 것으로 “43.5 C”라는 정보를 볼 수 있다. 이 값은 전술한 바와 같이, 랜덤으로 생성된 실험 결과 값이다.

Ⅳ. CoAP의 센서 노드 적용

4.1. 토폴로지

그림 13은 맥스포의 MTM-CMx000 센서 노드및 디바이스[5]를 이용하여 프로토콜을 적용하기 위한 토폴로지를 보이고 있다. 센서 노드 적용을 위해서 전술한 클라이언트-프락시-서버 노드가 사용되었고, 서버는 TinyOS 환경[6]에서 NesC[7]로 구현하는 실험환경을 이용하였다.

4.2. 센서노드용 CoAP 메시지 플로우

CoAP 프락시 서버는 URI_SCHEME을 이용하여, 수신된 데이터의 목적지를 결정하고, 이를 목적지로 전송한다.

```
... Receive Data From Client is Omitted
=======================================
       SND to Proxy:46 B
=======================================
     42(B) 01() 08) 52(R) 9B(?
     74(t) 65(e) 6D(m) 70(p) 65(e)
     72(r) 61(a) 74(t) 75(u) 72(r)
     65(e) 3F(?) 1C() 63(c) 6F(o)
     61(a) 70(p) 3A(:) 2F(/) 2F(/)
     32(2) 30(0) 33(3) 2E(.) 32(2)
     35(5) 30(0) 2E(.) 31(1) 33(3)
     33(3) 2E(.) 31(1) 37(7) 32(2)
     3A(:) 31(1) 31(1) 31(1) 31(1)
     31(1)
=======================================
       PROXY RECEIVE From Server :40 B
=======================================
     61(a) 50(P) 08) 52(R) 3F(?)
     1C() 63(c) 6F(o) 61(a) 70(p)
     3A(:) 2F(/) 2F(/) 32(2) 30(0)
     33(3) 2E(.) 32(2) 35(5) 30(0)
     2E(.) 31(1) 33(3) 33(3) 2E(.)
     31(1) 37(7) 32(2) 3A(:) 32(2)
     32(2) 32(2) 32(2) 32(2) 34(4)
     33(3) 2E(.) 35(5) 20( ) 43(C)
=======================================
... Receive Data from Server is Omitted
=======================================
       SND to Client :40 B
=======================================
     61(a) 50(P) 08) 52(R) 3F(?)
     1C() 63(c) 6F(o) 61(a) 70(p)
     3A(:) 2F(/) 2F(/) 32(2) 30(0)
     33(3) 2E(.) 32(2) 35(5) 30(0)
     2E(.) 31(1) 33(3) 33(3) 2E(.)
     31(1) 37(7) 32(2) 3A(:) 32(2)
     32(2) 32(2) 32(2) 32(2) 34(4)
     33(3) 2E(.) 35(5) 20( ) 43(C)
```

그림 11. 프락시 메시지 덤프
Fig. 11 Proxy Message Dump

```
=======================================
       RCV RAW DATA :46 B
=======================================
     42(B) 01() 08) 52(R) 9B(?
     74(t) 65(e) 6D(m) 70(p) 65(e)
     72(r) 61(a) 74(t) 75(u) 72(r)
     65(e) 3F(?) 1C() 63(c) 6F(o)
     61(a) 70(p) 3A(:) 2F(/) 2F(/)
     32(2) 30(0) 33(3) 2E(.) 32(2)
     35(5) 30(0) 2E(.) 31(1) 33(3)
     33(3) 2E(.) 31(1) 37(7) 32(2)
     3A(:) 31(1) 31(1) 31(1) 31(1)
     31(1)
=======================================
       SND to Proxy :40 B
=======================================
     61(a) 50(P) 08) 52(R) 3F(?)
     1C() 63(c) 6F(o) 61(a) 70(p)
     3A(:) 2F(/) 2F(/) 32(2) 30(0)
     33(3) 2E(.) 32(2) 35(5) 30(0)
     2E(.) 31(1) 33(3) 33(3) 2E(.)
     31(1) 37(7) 32(2) 3A(:) 32(2)
     32(2) 32(2) 32(2) 32(2) 34(4)
     33(3) 2E(.) 35(5) 20( ) 43(C)
```

그림 12. 서버 메시지 덤프
Fig. 12 Server Message Dump

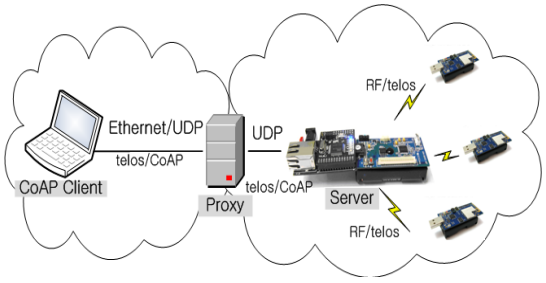

그림 13. 센서 적용을 위한 토폴로지
Fig. 13 Topology for Sensor Nodes

그림 14에서 보는 바와 같이, 센싱 정보를 수집하는 센서 노드가 주기적으로 수집 정보를 RF(Radio Frequency)로 전송하고, 이 결과를 프락시 서버로 동작하는 정보 수집 장치에 캐싱 한 후, 요청 메시지에 대한 결과를 응답하는 형태로 구현하였다.

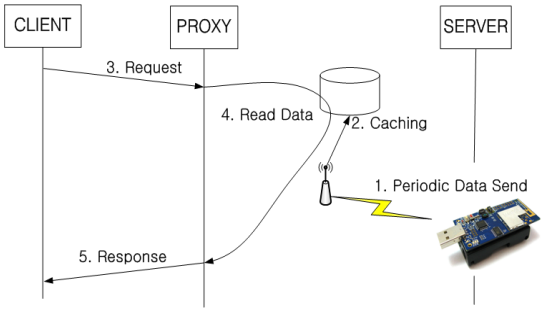

그림 14. 센서노드 CoAP 메시지 플로우
Fig. 14 CoAP Message Flow with Sensor Node

4.3. 센서노드용 CoAP 메시지 플로우

3장에서 기술했던 방법과 유사하게 덤프 데이터를 통해 메시지의 흐름을 그림 15에서 확인할 수 있다. PC환경에서 랜덤으로 생성된 값을 이용하는 것과 달리, 센서 노드를 통해 실제 값을 수집하기 위해, 클라이언트와 프락시는 Cygwin 환경에 포팅하여 PC 버전 프로그램을 하드웨어 특성에 맞도록 일부 수정하였고, MTM-CM5000은 온도, 습도를 센싱하는 장치로 사용하였고, 인터페이스 보드에 장착된 MTM-CM2000은 서버 노드에 해당한다.

그림 15는 클라이언트에서 수신한 UART(Universal Asynchronous Receiver Transmitter) 정보 및 이를 제거하여 실제 온도 값을 확인하는 화면을 보인 것이다.

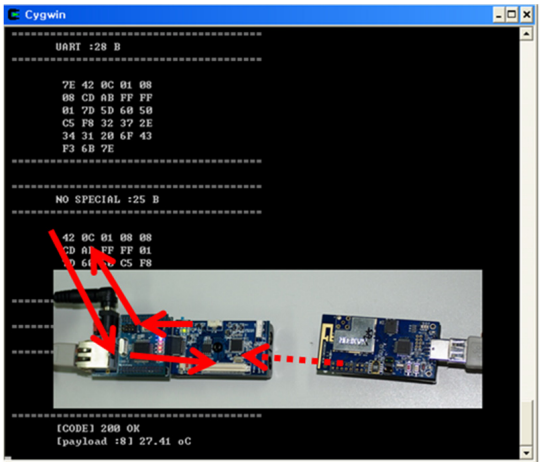

그림 15. 센서 노드 적용된 덤프 및 데이터 흐름
Fig. 15 Message Dump and Data Flow with Sensor Nodes

V. 결 론 및 향후 연구 과제

센서 노드를 효율적으로 관리하기 위해, 표준화된 프로토콜이 필요하다. 이를 위해 IETF WG에서 CoAP 프로토콜을 개발하고, 국제 표준화 작업이 진행 중이다. 아직 드래프트 단계이고, 새로운 버전이 지속적으로 발표되고 있기 때문에, 이 프로토콜에 대한 논리적 검증이 필요하다. 센서 노드는 PC 환경과 비교하여, 전력, 메모리, 컴퓨팅 파워 등 많은 부분에서, 최소한의 기능으로 제한되는 하드웨어적인 한계를 가지고 있다. 이러한 이유로 PC 환경에서와는 달리 하드웨어에 종속적인 부분이 존재하고, 이를 위한 별도의 알고리즘이 필요하다.

본 논문에서는 PC 환경에서 드래프트로 제안하는 기능을 소프트웨어적으로 구현하여 논리적인 검증을 하고, 이를 통해 센서 노드에 적용하였다. 센서 노드에 적용하기 위하여, 온도, 습도를 센싱 할 수 있는 디바이스가 정보를 수집하여, 데이터 수집 장치에 RF로 전달하고, 이를 수신한 데이터 수집 장치에 하드웨어에 종속적인 telos정보 정보에 CoAP 프로토콜을 탑재하는 형태를 취하여 CoAP 프로토콜을 확인하였다.

이는 센서 노드에 모든 프로토콜 스택을 탑재하지 않고, 기존에 개발하여 사용하는 Cygwin 환경을 최대한 이용하였다.

향후 과제로는 센서노드에 프로토콜을 탑재하기 위해 데이터 수집 장치를 서버 노드로 사용하고, 서버 노드는 주기적으로 센서 디바이스로 부터 RF 신호를 수신하였는데, 요청 메시지를 수신 하는 경우에 응답하는 형태로 구현이 필수적이다. 뿐만 아니라, 새롭게 발표되는 버전의 프로토콜 드래프트에 대해서도 발 빠른 논리적 검증이 구현을 통해 이루어져야 한다.

참고문헌

[1] 민경주, 유상근, 김용운, 김형준, 정회경, “효율적인 USN 관리를 위한 CoAP 프로토콜 분석 및 적용반안”, 한국해양정보통신학회 추계 종합학술대회 논문집 14권 2호, p507~509

[2] Z. Shelby, B. Frank, D. Sturek, “Constrained Application Protocol (CoAP)”, IETF draft-ietf-core-coap-draft 00~draft 03, 2010.

[3] “libcoap: C-Implementation of CoAP″, http://libcoap.sourceforge.net, Apr. 2011

[4] K. Kuladinithi, O. Bergmann, T. Potsch, M. Becker and C. Gorg, ″Implementation of CoAP and its Application in Transport Logistics″, http://hinrg.cs.jhu.edu/joomla/images/stories/coap-ipsn.pdf

[5] “Maxfor® USN Mote MTM-CMx00-MSP series”, http://maxfor.co.kr/sub2_1_1_1.html, Jan. 2011

[6] “TinyOS Home Page”, http://tinyos.net, Jan. 2011.

[7] E. Brewer, D. Culler, D. Gay, P. Levis, R.V. Behren, M. Welsh, “nesC: A Programming Language for Deeply Networked Systems”, http://nescc.sourceforge.net, Jan. 2011.

저자소개

민경주(Kyoung-Ju Min)

2000년 충남대학교 컴퓨터공학과(공학사)
2002년 충남대학원 컴퓨터공학과(공학석사)
2006년 충남대학교 컴퓨터공학과 박사과정 수료
2011년 ~ : 배재대학교 컴퓨터공학과 박사과정
2008년~현재: 유레카솔루션대표
※관심분야 : 초고속 네트워크, 센서 네트워크, 통신 프로토콜, AJAX & JQuery based Web Service

김용운(Yong-Woon Kim)

1989년 동아대학교 전자공학과(공학사)
1994년 포항공과대학교 정보통신공학과(공학석사)
1990년 ~ 1991년: 삼성항공
1995년 ~ 2001년: 한국전자통신연구원
2001년 ~ 2002년: ZTE 퓨처텔
2002년 ~ 2004년:이니텍
2005년 ~ 현재: 한국전자통신연구원
※관심분야:센서 네트워크,RFID 시스템, 컴퓨터 네트워크

유상근(Sang-Keun Yoo)

1997년 충남대학교 컴퓨터공학과(공학사)
1999년 충남대학교 컴퓨터공학과(공학석사)
1999년 ~ 2000년: 시그마텍
2001년 ~ 2001년: 한국전자통신연구원
※관심분야: 센서 네트워크, RFID 시스템, 정보보호 시스템, 컴퓨터 네트워크

김형준(Hyoung-Jun Kim)

1986년 광운대학교 컴퓨터공학과(공학사)
1988년 광운대학교 컴퓨터공학과(공학석사)
2007년 충남대학교 컴퓨터과학과(이학박사)
1988년 ~ 현재: 한국전자통신연구원
※관심분야:센서 네트워크,모바일 RFID 서비스, 컴퓨터 네트워크, 차세대인터넷

정희경(Hoe-Kyung Jung)

1985년 광운대학교 컴퓨터공학과(공학사)
1987년 광운대학교 컴퓨터공학과(공학석사)
1993년 광운대학교 컴퓨터공학과(공학박사)
1994년~현재 배재대학교 컴퓨터공학과 교수
※관심분야 : 멀티미디어 문서정보처리, XML, SVG, Web Services, Semantic Web, MPEG-21, Ubiquitous Computing, USN