# 2016년 1학기 캡스톤 디자인 설계 최종 보고서

# 사물인터넷 표준 프로토콜 CoAP에서의 보안 프로토콜 Tiny-DTLS 적용

- **과목명: 캡스톤디자인**
- **과목 담당교수: 이종원 교수님**
- **제출일: 2016년 06월 04일**
- **프로젝트 팀원명단**

| 팀원명 | 학번 | 제1 전공<br>(심화전공) | 제2전공 | 담당 업무<br>팀장부터<br>기입 |
|---|---|---|---|---|
| 김진우 | 21000189 | 컴퓨터 공학 심화 | - | 팀장 |
| 김연희 | 21000131 | 생명과학 | 컴퓨터공학, 전자공학 | 팀원 |
| 한영광 | 21100758 | 컴퓨터 공학 심화 | - | 팀원 |
| 위성일 | 21300464 | 컴퓨터 공학 심화 | - | 팀원 |

- **캡스톤 설계 완성도 [지도교수가 채점]**

| **점수** | **캡스톤 작품 구현 완성도** | |
|---|---|---|
| | **설명** | **지도교수 평가<br>해당난에 체크(O)** |
| 5 | **상용 제품으로도 손색이 없는 수준** | |
| 4 | **모든 기능이 잘 작동하고 최적화도 어느 정도 됨** | |
| 3 | **전반적인 기능 원활히 작동 함.<br>성능은 기존 제품이나 논문에 비해 미흡한 편** | |
| 2 | **일부 기능만 구현하거나 동작 중 오류가 가끔 발생** | |
| 1 | **다수 기능이 미구현되었거나 또는 미작동함** | |

**날짜:** **지도교수명:** **(서명)**

<table>
<tr><th colspan="7">설계 결과 요약서</th></tr>
<tr><td>설계주제<br>(Title)</td><td colspan="6">사물인터넷 표준 프로토콜 CoAP에서의 보안 프로토콜 Tiny-DTLS 적용</td></tr>
<tr><td>주제어<br>(Keywords)</td><td colspan="6">사물인터넷, 보안, 무선 센서 네트워크, CoAP, DTLS, TinyDTLS, OpenMote-CC2538, OpenBattery, OpenBase, Raspberry Pi, Ultrasonic sensor</td></tr>
<tr><td>설계팀원명단</td><td colspan="6">21000131 김연희<br>21000189 김진우<br>21100758 한영광<br>21300464 위성일</td></tr>
<tr><td>지도교수, 산업체자문위원<br>(Advisor)</td><td colspan="6">이종원 교수님</td></tr>
<tr><td>설계 기간<br>(Period)</td><td colspan="6">2015-07-01 ~ 2016-06-01</td></tr>
<tr><td>설계문제의 정의<br>(Problem Statement)</td><td colspan="6">1) 제한된 네트워크에서 사용하는 표준 프로토콜인 CoAP은 보안에 취약하다.<br>2) CoAP은 transport layer에서 UDP를 사용하므로 보안을 위하여 DTLS 프로토콜을 고려할 수 있으나 IoT 통신의 제한적인 리소스 때문에 실질적 구현이 어렵다.</td></tr>
<tr><td rowspan="2">설계요소<br>(Design Elements)<br>(해당요소에 O표)</td><td>목표 설정</td><td>분석 및 개념설계</td><td>상세설계</td><td>구현 및 제작</td><td>시험 및 평가</td><td>기타</td></tr>
<tr><td>O</td><td>O</td><td>O</td><td></td><td></td><td></td></tr>
<tr><td rowspan="2">제한조건<br>(Constraints)<br>(해당요소에 O표)</td><td>제작 비용 및 기간</td><td>환경</td><td>사회 및 윤리</td><td>안전 및 미학</td><td>산업표준</td><td>기타</td></tr>
<tr><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td></td></tr>
<tr><td>설계결과물<br>(deliverables)</td><td colspan="6">1) 센서로부터 센서 값을 받아 올 수 있는 라즈베리파이 환경<br>2) CoAP + tinydtls 통신을 simulation 할 수 있는 서버와 클라이언트 프로그램<br>3) Client측에서 결과를 확인할 수 있는 JAVA 기반 API<br>4) CoAP + TinyDTLS 코드가 올라간 OpenMote-CC2538<br>5) 라즈베리파이에 XBee dongle을 통해 연결한 Slip-radio OpenMote</td></tr>
</table>

| | |
|---|---|
| 설계 결과의 요약 (Abstract) (100자 이내) | Client가 센서를 가진 Server에게 resource를 request하면, 암호화된 CoAP 메시지가 server로 전송되고, server는 센서 값 암호화하여 client로 response한다. |


**Summary**

**Date: 2016-06-1**

| **Course Name** | **Capstone Design** | **Year/Semester** | **2016 - 1** |
|---|---|---|---|
| **Design Title** | Simulating IoT network security Using TinyDTLS | | |
| **Keywords** | IoT, Security, WSN, CoAP, DTLS, TinyDTLS | | |
| **Designers (stdudents)** | 21000131 Kim YeonHee<br>21000189 Kim JinWoo<br>21100758 Han YoungKwang<br>21300464 Wi SungIl | | |
| **Advisor(s)** | Professor Lee JongWon | | |
| **Design Problem Definition** | 1) CoAP protocol used in constraint environment is vulnerable to Security.<br>2) CoAP is operating over UDP. So We may consider DTLS protocol to service security for CoAP. But, IoT network has limited resource. So, we can't apply DTLS protocol to CoAP. | | |
| **Design Constraints** | 1. Development Environment :<br>For ultrasonic sensor, using HC-SR04 model.<br>For Raspberry Pi, using Raspbian.<br>For OpenMote, using JLink Debugger.<br>2. Operating Environment and Stability :<br>Making encryption of Plain text flawless.<br>3. Cost :<br>All cost is within 1,500,000 won.<br>4. Aesthetic :<br>1) For 30 inch monitor, show JAVA GUI depending on response received from ultrasonic sensor value.<br>2) For 30 inch monitor, it contains ability to showing contents of packet encryption and deletion of contents.<br>5. Ethicality :<br>Make to not violate network communication law.<br>6. Etc:<br>1) Time to sense movement with ultrasonic sensor is under 5 | | |

| | |
|---|---|
| | seconds.<br>2) Type of tiny-DTLS cipher suite :TLS_PSK_WITH_AES_128_CCM_8<br>**3)** Ultrasonic sensor must be able to sense things within 20 cm. |
| **Design Objectives** | 1) IoT communication using CoAP over TinyDTLS.<br>2) Making simulation program for testing to get ultrasonic sensor value and encrypt, communicate between server and client.<br>3) Making model for implementing our protocol on hardware. |
| **Deliberables** | 1) Client: Raspberry pi connected to ultrasonic sensor.<br>2) Server: Raspberry pi<br>**3)** Simulation program to test communicating sensor value with CoAP and TinyDTLS.<br>**4)** OpenMote fused CoAP and TinyDTLS as a server.<br>**5)** OpenMote fused Slip-Radio as a router.<br>**6)** Californium-Scandium as a client program |
| **Extended Abtracts** | 1. Simulation Program<br>1) Client request distance resource to server connected with ultrasonic sensor.<br>2) Encrypted CoAP message is sent to server.<br>3) Server response to client with encrypted sensor value<br>4) If the response value is lower than 20 cm, alert screen will be showed.<br><br>2. Hardware (OpenMote-CC2538)<br>1) Client program request temperature/humidity resource to OpenMote server connected to OpenBattery.<br>2) DTLS handshake process is partially succeeded until ClientHello with Cookie. |

- **Level Descriptor (문제수준 설명)**

| 번호 | 문제의 속성 | 문제수준 설명<br>**컴퓨팅 문제란 아래 속성들 중 일부 또는 전부를 갖는 컴퓨팅 문제이다.** | 체크란 |
|---|---|---|---|
| 1 | **상충되는 요건의 범위** | 상충되는 기술적, 컴퓨팅적 요건과 그 외 다른 상충적 요건을 포함한다. | O |
| 2 | **요구되는 분석의 깊이** | 명백한 해답이 없으며, 적절한 추상적인 모델을 수립하기 위해 개념적인 사고와 독창적인 분석을 요구한다. | O |
| 3 | **요구되는 지식의 깊이** | 교과과정을 통해 습득한, 깊이 있는 컴퓨팅 지식 또는 특정 분야의 지식 그리고 확립된 이론에 근거한 분석적인 방법이 사용된다. | O |
| 4 | **논점의 대상** | 흔히 다루지 않는 논점을 포함하고 있다. | O |
| 5 | **문제의 수준** | 전문적인 컴퓨팅에 요구되는 표준적인 방식과 일반적인 실무 절차로 해결되지 않는다. | |
| 6 | **이해당사자들의 요구 수준 및 범위** | 다양한 이해당사자의 서로 다른 요구사항을 고려한다. | |
| 7 | **영향력** | 문제해결이 광범위한 분야에 중요한 영향을 미친다. | O |
| 8 | **상호의존성** | 상호 의존하는 구성요소들 또는 많은 하위요소들로 구성되어 있다. | O |
| 9 | **문제의 명확성** | 문제의 요구조건 또는 필요성이 모호하거나 명백하게 기술되어 있지 않다. | |

**0. Executive SUMMARY**

최근 사람과 사람, 사람과 사물들을 인터넷에 연결하여 초-연결 사회를 구축할 수 있는 기반 기술인 사물인터넷(IoT: Internet of Things)에 대한 관심이 높아지고 있다. 시장조사업체 IDC에 따르면, 전 세계 사물인터넷 솔루션 시장은 '13~'20년 연평균 성장률 17.5%를 기록하였고 '13년 1조 9,000억달러에서 '20년에는 7조 1,000억 달러까지 성장할 것으로 예상하고 있다. 또한 시장조사업체 가트너는 '14년 무선 커넥티드 단말 보급대수가 '14년 160억 대, '20년에는 409억대로 증가할 것으로 전망하고 있다. 한편, IoT 기술의 활성화 및 서비스 창출을 위하여 해결하여야 하는 한가지 문제는 사물인터넷 보안의 구현이다. 실제로 독일 IT 보안업체 리큐리티 랩스는 13년 해킹 실험을 통해 독일 남부에 위치한 도시의 전력 공급을 외부에서 무단으로 차단할 수 있음을 입증하였다. 따라서 본 프로젝트에서는 사물인터넷의 네트워크 환경에 보안을 적용하는 것으로 목표를 세우고 표준 프로토콜인 CoAP과 보안 프로토콜인 DTLS를 결합하여 보안이 적용된 사물인터넷 프로토콜을 구현하고자 한다.

본 프로젝트의 설계 목표는 IETF에서 IoT 표준으로 선정한 프로토콜인 CoAP과 보안 프로토콜인 TinyDTLS를 결합하여 실제 통신하는 과정 하드웨어 및 시뮬레이션으로 구현하는 것이다. 먼저, OpenMote-CC2538 hardware에 구현하는 방법은 다음과 같다. Client 측에서 센서 값을 요청하는 메시지를 암호화하여 전송하고, server 측에서는 받은 메시지를 복호화하여 요청한 센서 값을 측정한 뒤 다시 암호화하여 전송하는 암호화 통신을 하도록 한다. 그러나 TinyDTLS를 적용하는 과정에 있어서 코드 상의 문제로 Handshake 과정이 완료되지 못하였다. 따라서 본 프로젝트에서는 라즈베리파이 상에 client와 server를 모두 구현하여 시뮬레이션으로 구현하기로 결정하였다. 이 때, ultrasonic sensor에서 측정한 거리를 resource로 결정하였고, 가까운 거리에 어떠한 물체가 감지된다면 경고창을 띄워주도록 설계하였다. Wireshark와 같은 패킷 캡쳐 프로그램에서는 data가 암호화되어 있기 때문에 어떤 값을 가지는지 확인 불가능하며, 따라서 packet sniffing 등의 공격에 대한 방어가 가능하다.

본 프로젝트에서 사용한 오픈소스 라이브러리인 libcoap과 tinyDTLS이다. 본 팀은 이 두 가지 오픈 소스를 자체적으로 병합한 코드를 제안하며, Californium/Scandium 서버를 이용하여 보편성 테스트를 정상적으로 수행하여 본 프로젝트의 결과로 무선 네트워크 환경에서 데이터가 암호화 되어 전송됨을 확인한다. 기존에 사용하였던 사물인터넷에 보안 기술을 제공함으로써 단말 기간의 더 안전한 통신을 예상하고 향후 사물인터넷 환경에 적절한 보안 솔루션을 제공할 수 있을 것이라 기대한다.

## 1. PROJECT OVERVIEW (3~4 페이지 분량으로 작성)

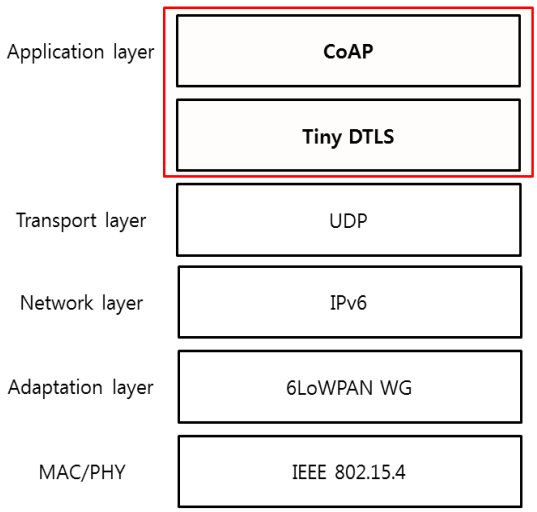
### 1.1 Introduction

사물인터넷(IoT: Internet of Things)은 사람과 사물, 서비스의 분산된 환경 요소에 사람이 개입하지 않고도 상호 협력하여 통신하는 연결망을 의미한다. 사물 간 통신하는 이동통신망 환경은 전력, 메모리 사이즈 등의 자원이 제한되어 있다. 이러한 환경에서 통신이 가능하다 할지라도 보안이 적용되지 않으면 실용화되기 어렵다. 본 프로젝트에서는 사물인터넷 프로토콜인 CoAP과 보안 프로토콜인 DTLS를 병합시켜 보안이 지원되는 사물인터넷 simulation program을 구현해보고자 한다.

최근 센서 및 각종 장치 간 인터넷을 통하여 상호 통신할 수 사물인터넷에 대한 관심이 고조됨에 따라 프로토콜의 표준화가 필요하게 되었다. 따라서 Internet Engineering Task Force (IETF)에서는 표준 사물 인터넷 프로토콜로써 Constrained Application Protocol (CoAP)을 지정하게 되었다.

CoAP은 WSN 노드와 같이 제한된 노드와 저 전력, 고 손실의 제한된 네트워크 상에서 사용하기에 특화된 웹 프로토콜이다. 노드는 주로 작은 용량의 ROM, RAM을 가진 8bit microcontrollers를 사용하며, 6LOWPAN 환경에서의 IPv6를 사용한다. UDP 기반으로 동작하며, unicast 및 재전송, 타이머 기능을 지원함으로써 신뢰성을 제공한다. 또한 RESTful 기반의 통신을 사용하기 때문에 http와 연동이 가능하다. 동기 및 비 동기 방식을 지원하고 오버헤드가 적은 바이너리 헤더를 사용함으로써 제한된 환경에서의 통신을 더 효율적으로 제공할 수 있다. 전달하는 메시지는 요청 및 응답의 2가지를 사용한다. CoAP message type은 CON, NON, RST, ACK이고, method는 GET, PUT, POST, DELETE이다.

CoAP의 계층도는 Figure 1.1과 같다. MAC/PHY layer에서는 IEEE 802.15.4 를 사용하고, Adaptation layer에서는 6LoWPAN WG를 사용하며, Network layer에서는 IPv6, Transport layer에서는 UDP, Application layer에서는 TinyDTLS, CoAP을 사용한다.




**[Figure 1.1]**

사물인터넷 환경은 장치들이 인터넷을 사용해서 통신하기 때문에, 인터넷에서 발생할 수 있는 모든 취약점이 동일하게 발생할 수 있다. 사물인터넷 활성화를 위해서는 이러한 위협 요소에 대응하는 사물인터넷 보안이 필수적이다.

본 프로젝트에서 사용한 프로토콜은 CoAP과 TinyDTLS이다.

CoAP은 UDP 기반의 애플리케이션 프로토콜이기 때문에 보안 해결책으로써 DTLS 프로토콜을 고려해볼 수 있다. 그러나 DTLS는 핸드 쉐이크 과정 등에서 발생하는 리소스가 많기 때문에 사물인터넷 환경에서 사용하기 위해서는 경량화 할 필요가 있다.

따라서 본 프로젝트에서는 DTLS를 경량화 시킨 library인 TinyDTLS를 사용하기로 결정하였다. TinyDTLS는 경량화하기 위하여 다음과 같은 방법을 사용하였다. 첫째, Raw Public Key (RPK) 모드에서 사용하는 cipher suite인 TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 와 Pre-Shared Key (PSK) 모드에서 사용하는 cipher suite인 TLS_PSK_WITH_AES_128_CCM_8 의 2가지 Cipher Suites 만을 사용하였다. 또한 AES, ECC와 같은 암호화 연산을 위한 하드웨어 가속을 사용하는 인터페이스를 제공한다.

본 프로젝트에서 사용한 open source는 libcoap과 tinydtls이다.

Libcoap은 C 기반으로 CoAP을 구현한 library이며, 임베디드 장치 및 POSIX OS를 기반으로 한 컴퓨터에서 구동되도록 설계된 open source library이다. RFC 7252 (CoAP 18) 에 명시되어 있는 기능 뿐만 아니라 observation, block-wise transfer 옵션을 제공한다. tinydtls는 C 기반으로 TinyDTLS를 구현한 library이며, 사물 인터넷 노드에 보안을 제공해 줄 수 있다. 본 프로젝트에서는 libcoap과 tinydtls를 사용하여 서버와 클라이언트 간 통신을 하고자 한다.

본 프로젝트에서 달성하고자 한 바는 다음과 같다. OpenMote-CC2538 hardware 2개를 사용하여 암호화된 온/습도 센서 값을 주고 받는다. 센서 값이 일정 값 이상이 될 경우에 알림을 보여준다. 그러나 구현 결과, TinyDTLS의 코드 상 문제로 인하여 handshake 과정이 완료되지 않았다. 문제가 생긴 지점은 ServerHello 패킷이었는데, server 상에서 메시지가 생성 된 후 전송되었으나 client 측에서 받지를 못했다.

따라서 우리 팀은 본 프로젝트의 방향을 수정하였다. 실제 hardware가 아닌 raspberry pi 상에서 server, client program을 구현했다. Server로는 raspberry pi에 ultrasonic sensor를 연결하여 사용하고, client는 raspberry pi를 사용한다. Client가 server로 request하면, 암호화된 패킷이 server로 전송된다. Server측에서는 ultrasonic sensor의 값을 5초 간격으로 읽어서 client에게 암호화하여 response한다. Client에서 받은 response 값이 20cm 이하라면, 침입자가 발생했다는 경고 API를 띄워준다.

## 1.2 Project Goal:

1.2.1 M2M 통신에서 사용되는 표준 프로토콜인 CoAP과 그 통신에서의 보안 프로토콜인 tiny-DTLS를 결합한다.

1.2.2 사물인터넷에 적합한 hardware인 OpenMote-CC2538을 client 및 server로 사용하여 암호화된 통신을 할 수 있도록 한다.

1.2.3 암호화된 통신을 확인할 수 있는 Simulation program을 구현한다.

## 1.3. Key Project Stakeholders:

1.3.1. 사물인터넷 물품을 판매하는 업체

1.3.2. 사물인터넷 보안 솔루션 개발 업체

1.3.3. 사물인터넷 보안 연구를 목표로 하는 학부생 혹은 대학원생

## 1.4. Project Requirements and Constraints:

### 1.4.1. Requirements

1.4.1.1. IoT 표준 프로토콜 CoAP과 사물인터넷 보안에 특화된 프로토콜인 TinyDTLS를 결합

1.4.1.2. OpenMote-CC2538에 slip-radio를 퓨징하여 raspberry pi에 XBee dongle을 통해 연결한 뒤 라우터로 사용

1.4.1.3. OpenMote-CC2538에 CoAP + TinyDTLS를 퓨징하여 server로 사용

1.4.1.4. 라즈베리 파이에 CoAP+TinyDTLS를 사용한 simulation program 구현

1.4.1.5. 라즈베리 파이에 통신 결과를 보여주는 GUI 구현

### 1.4.2. Constraints

| 제한조건 | 내용 |
|---|---|
| 제작비용 및 기간 | 비용은 150만원 이내로 한다.<br>기간은 2015.07.01부터 2016.06.04 로 한다. |
| 사회, 문화, 윤리 | 외부 해킹에 의한 보안 취약점과 별개로 사물인터넷에서 실시간 수집되는 정보가 사생활을 침해하지 않아야 한다. |
| 안전, 보건, 환경 | 사물인터넷 통신을 하는 데에 데이터의 기밀성, 무결성, 가용성을 보장해야 한다. |
| 산업표준 | IETF에 지정한 표준 프로토콜인 CoAP의 document인 RFC 7252를 어기지 않아야 한다.<br>UDP 보안 프로토콜인 DTLS의 document인 RFC 6347을 어기지 않아야 한다. |

| | |
|---|---|
| 기타 (미학적 요인 등) | 센서로부터 받아오는 데이터는 직관적으로 나타내야 한다. |

### 1.5. Deliverables (최종산출물) :

#### 1.5.1. Documentation

공프기 보고서, 캡스톤 보고서, 소스 코드 설명서, 사용 설명서, 특허보고서, 발표 세미나 ppt, 논문 초안

#### 1.5.2. Hardware

라즈베리파이, OpenMote-CC2538, OpenBase, OpenBattery, Ultrasonic sensor, bread borad, jumper, JLink Debugger, ARM JTag, XBee dongle, Linux PC

#### 1.5.3. Software

CoAP+TinyDTLS source code, californium, scandium, slip-radio, 6lbr-demo, 6lbr web page, JLink GDB Server, Xrdp

#### 1.5.4. Etc.

UCC, Demo, 포스터 팜플렛

## 2. PROJECT BACKGROUND

### 2.1 Literature Review

#### 2.1.1 RFC 7252 – The Constrained Application Protocol

사물인터넷은 다양한 기기가 서로 연결되어 있는 네트워크이다. 거기에는 컴퓨터와 스마트폰처럼 높은 처리 용량과 메모리를 가지고 있는 node도 있지만 세서와 같은 작은 메모리와 낮은 처리 성능을 가진 노드들도 인터넷에 연결된다. 이에 인터넷 표준화 단체인 IETF는 다양한 노드를 수용 할 수 있는 사물 인터넷 프로토콜을 만들기 시작하였고, 그 중 하나가 기존의 HTTP 웹 프로토콜에 대응되는 CoAP(Constrained Application Protocol)프로토콜이다.

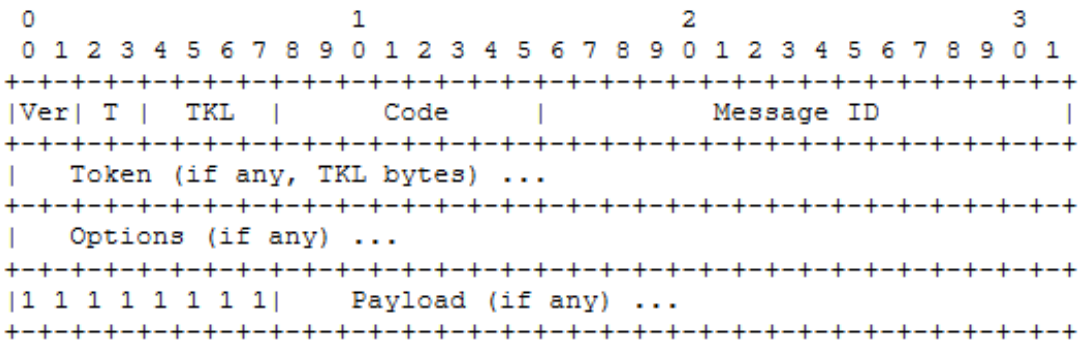
최근 사물인터넷 시대의 도래에 따라 다양한 세서 노드 및 각종 장치, 기기 간에 상호 통신할 수 있는 가벼운 공통 프로토콜이 필요하게 되었다. 사물인터넷 관련 표준화로는 ITU 및 ETSI가 서비스 모델, 서비스 연동 등 큰 그림을 담당하고 있으며, IETF에서 IP기반 프로토콜 표준화를 주도하고 있다. 그 밖에 IPSO, OMA등 기관에서도 사물인터넷 관련 표준 적용을 지원하고 있다.

IETF CoRE워킹그룹에서는 사물 인터넷 노드 사이에 통신 할 수 있는 CoAP을 개발하고 있다.

CoAP프로토콜을 기본적으로 IP계층 위에 UDP 트랜스 포트 계층을 가정하고 있지만, 하위 계층과 독립적으로 설계되어 다른 네트워크 계층 및 트랜스포트 계층에서도 동작할 수 있다.


**[Figure 2.1]**

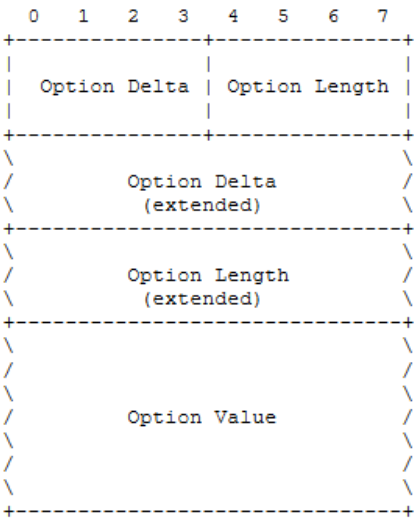
전체 프로토콜 스택 중에서 CoAP 프로토콜의 레이어의 위치는 이러하다. CoAP 프로토콜은 application protocol에 속한다. 하위로는 Transport layer에는 UDP 프로토콜, Network layer에는 IPv6 프로토콜, Adaptation layer에는 6LoWPAN WG 프로토콜, MAC/PHY layer에는 IEEE 802.15.4 프로토콜이 각각 위치하고 있다.




**[Figure 2.2] CoAP message format**

[Figure 2.2] 의 메시지 형식은 CoAP 프로토콜의 메시지 전송 포맷을 나타낸다.

CoAP Message Format에서 Header(메시지 유형과 request code, Message ID 등)와 Token(메시지 확인자), Option, Payload(데이터)로 구성되어 있다.




**[Figure 2.3] CoAP option format**

[Figure 2.3]의 형식은 Options의 포맷을 구체화시킨 것이다.

Option Format은 Option Delta(현재 Option number와 이전 Option number의 차이), Option Length(Option의 길이), Option Value(각각의 Option에 할당되어 있는 Value)로 구성되어 있다.

| | Code |
|---|---|
| Request | 0 |
| Success Response | 2 |
| Client Error Response | 4 |
| Server Error Response | 5 |

**[Figure 2.4] CoAP message code**

[Figure 2.4]의 형식은 Message Code에 담길 내용을 표로 나타낸 것이다.

송신 측이 Request를 보낼 때는 code 0.xx이고 수신 측이 Response를 성공적으로 받았을 때, 1.xx를 보낸다. 만약 Response가 실패했을 경우, Client와 Server에 따라 각각 4.xx , 5.xx를 보내게 된다.

[Figure 2.5]의 표는 각각의 option들을 표로 설명한 것이다. (구체적 목록)

| No. | C | U | N | R | Name | Format | Length | Default |
|---|---|---|---|---|---|---|---|---|
| 1 | x | | | x | If-Match | opaque | 0-8 | (none) |
| 3 | x | x | - | | Uri-Host | string | 1-255 | (see below) |
| 4 | | | | x | ETag | opaque | 1-8 | (none) |
| 5 | x | | | | If-None-Match | empty | 0 | (none) |
| 7 | x | x | - | | Uri-Port | uint | 0-2 | (see below) |
| 8 | | | | x | Location-Path | string | 0-255 | (none) |
| 11 | x | x | - | x | Uri-Path | string | 0-255 | (none) |
| 12 | | | | | Content-Format | uint | 0-2 | (none) |
| 14 | | x | - | | Max-Age | uint | 0-4 | 60 |
| 15 | x | x | - | x | Uri-Query | string | 0-255 | (none) |
| 17 | x | | | | Accept | uint | 0-2 | (none) |
| 20 | | | | x | Location-Query | string | 0-255 | (none) |
| 35 | x | x | - | | Proxy-Uri | string | 1-1034 | (none) |
| 39 | x | x | - | | Proxy-Scheme | string | 1-255 | (none) |
| 60 | | | x | | Size1 | uint | 0-4 | (none) |

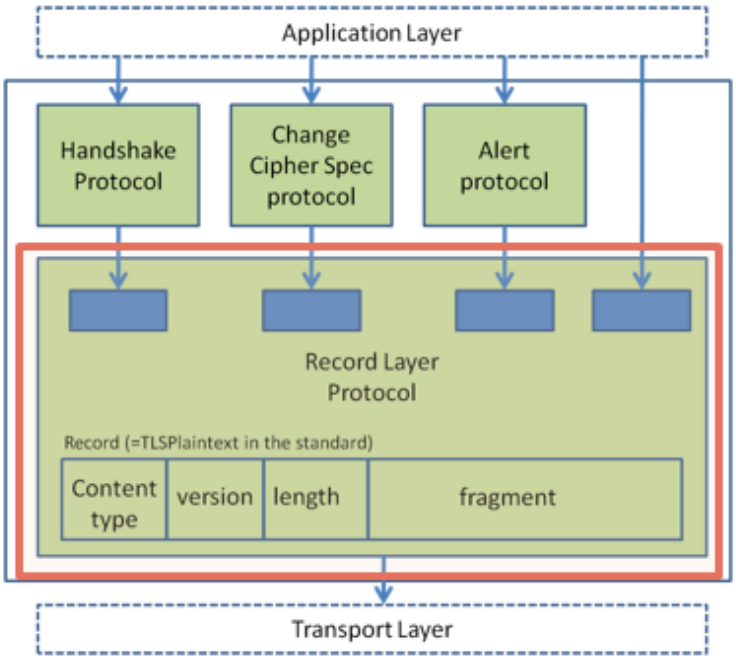
C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

**[Figure 2.5] CoAP option**

Critical/Elective(C)는 수신 측이 Option을 이해하지 못할 경우 메시지 처리를 중단하거나 무시하도록 한다. Safe/Unsafe(U)는 Client와 Server 사이에 Proxy가 Option을 처리하는 방법을 나타낸다. NoCacheKey(N)는 이전 메시지에 대한 response의 재사용 여부를 결정한다. Option의 종류로 Uri-Host, Uri-Port, Uri-Path, Uri-Query는 서버에 있는 resource의 위치를 구체화한다. ETag는 개별 리소스의 식별자로써, client는 Etag를 보내어 해당 리소스가 유효한지 확인할 수 있다. If-Match는 request에 대한 조건문으로 특정 조건이 충족되면 해당 리소스 식별자(ETag)의 존재 여부를 확인한 후 해당 리소스의 갱신 여부를 처리한다. Max-Age는 리소스가 유효할 수 있는 최대 생존 시간(lifetime)이다. Content-Format은 메시지 payload의 representation format을 명시한다. Accept 는 Client에서 지원 가능한 Content Format을 나타낸다. Location-Path와 Location-Query는 resource가 새로 생성 되었을 때, 경로를 의미한다. Size1은 리소스의 크기를 보여준다.

### 2.1.2 RFC 6347 – Datagram Transport Layer Security Version 1.2

DTLS란 Datagram Transport Layer Security의 약어로 Application layer에서 encapsulate 되는 data를 암호화하고, 정해진 크기로 자르고, 인증 헤더 등을 덧붙여서 Transport Layer로 전달하는 역할을 하는 프로토콜이다. DTLS는 Datagram에 적용되는 보안 프로토콜이기 때문에 Transport layer에서 UDP를 사용하고, CoAP도 Transport Layer에서 UDP의 service를 받는다. 따라서 CoAP에 적용 가능한 보안 프로토콜로 DTLS가 제안되었다.




**[Figure 2.6] DTLS protocol**

[Figuer 2.7]은 Handshake protocol의 동작과정이다.

Phase 1에서는 양 측에서 만든 random number, 사용할 Cipher suite, Compression method가 교환된다. Phase2에서 서버가 클라이언트에게 certificate을 통해 인증, public key를 전달하고 필요 시 Client에 대한 인증을 요청한다. Phase3에서는 양쪽에서 사용될 세션 키의 재료가 될 premaster secret을 클라이언트가 만들어 서버의 public key로 encryption하여 서버에게 전달하고 서버가 인증을 요청한 경우 클라이언트의 certificate을 보내 인증을 한다. 그리고 Phase4에서는 교환된 암호 재료들을 확인하고 change cipher spec 메시지를 통해 암호화 방법을 최종 확정한다.

```
Client                                          Server
------                                          ------

ClientHello                 -------->                           Flight 1

                            <--------    HelloVerifyRequest     Flight 2

ClientHello                 -------->                           Flight 3

                                                  ServerHello    \
                                                 Certificate*    \
                                           ServerKeyExchange*     Flight 4
                                          CertificateRequest*    /
                            <--------        ServerHelloDone    /

Certificate*                                                    \
ClientKeyExchange                                               \
CertificateVerify*                                               Flight 5
[ChangeCipherSpec]                                              /
Finished                    -------->                           /

                                          [ChangeCipherSpec]    \ Flight 6
                            <--------             Finished    /
```

Figure 1. Message Flights for Full Handshake

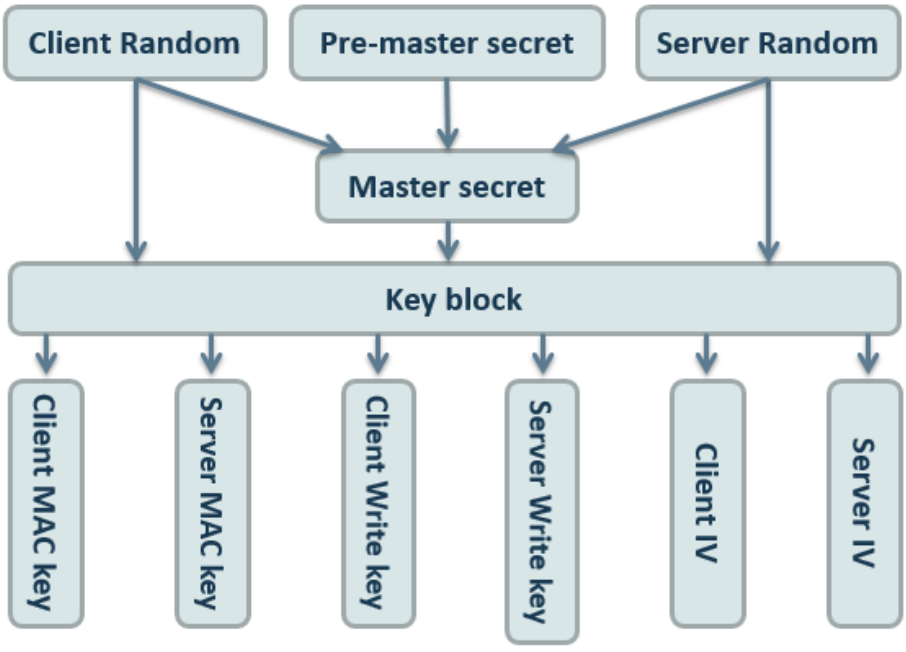
```
Client                                          Server
------                                          ------

ClientHello                 -------->                           Flight 1

                                                  ServerHello    \
                                          [ChangeCipherSpec]     Flight 2
                             <--------             Finished    /

[ChangeCipherSpec]                                              \Flight 3
Finished                     -------->                          /
```

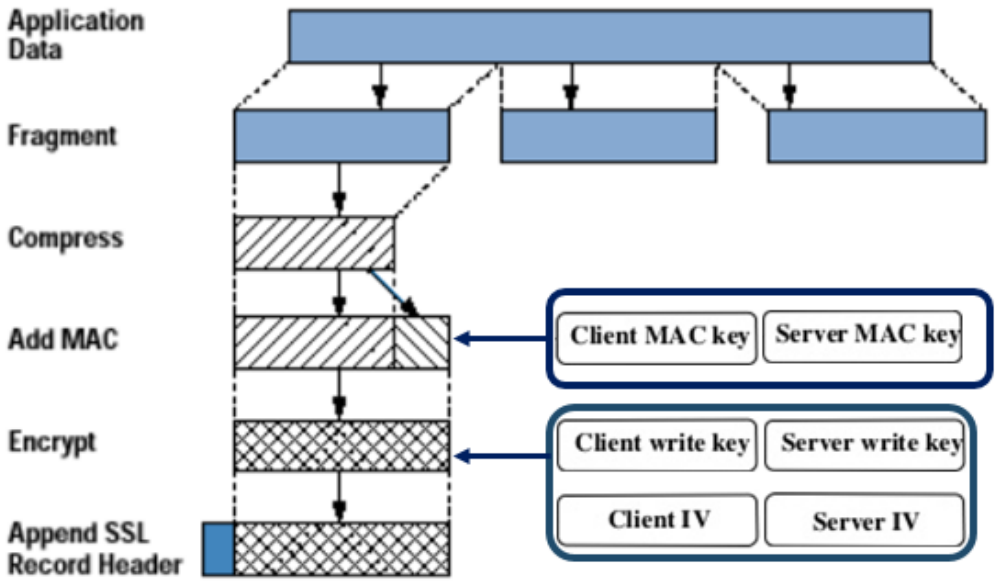
**[Figure 2.7] DTLS Handshake protocol**




**[Figure 2.8] Establishing Session Key**

[Figure 2.8]은 Session key를 결정하는 과정이다.

Handshake Protocol의 phase1에서 교환했던 Random number와 phase3에서 교환했던 Pre-master Secret 값을 hash function에 넣어 master secret 값을 얻고, 다시 이 값과 Random number을 Hash function에 넣어 Key block을 생성한다. 이 Key Block을 앞의 부분부터 차례대로 잘라서 위 그림과 같이 해당 Key로 가져간다. MAC key란 MAC을 만들 때 사용하는 key이고, Write key란 data를 암호화 할 때 사용하는 키이다. Client IV, Server IV는 특정 Cipher suite에 사용되는 값들이다.




[Figure 2.9] Record Protocol

[Figure 2.9]는 실제 메시지로 사용되는 record를 생성하는 과정을 나타낸다.

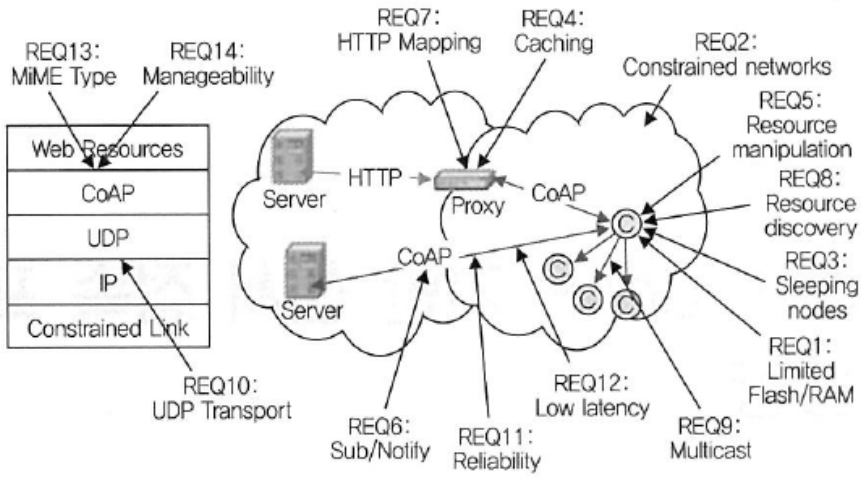
Application layer로부터 받은 data를 전송 가능 크기에 맞춰 fragment 한 뒤, compression method를 이용하여 compress를 한다. MAC key와 MAC 알고리즘을 사용하여 Integrity와 Non-repudiation을 보장해줄 MAC을 만들어 메시지의 뒤에 붙인다. 그리고 이 메시지를 Encryption 알고리즘과 write key, 필요한 경우 IV(Initialization Vector)을 이용하여 Encryption을 한 뒤, DTLS 헤더를 앞에 추가하여 UDP layer로 전달한다.

### 2.1.3 IETF CoAP 기반 센서 접속 프로토콜 기술 동향

기존의 센서 접속 기술은 그 분야에 따라 다양한 표준과 규격이 존재하였다. 빌딩 자동제어 및 제어 네트워크용으로는 BACnet이 있으며, 공장제어 및 계측제어를 위한 Modbus등이 다양한 프로토콜이 존재한다. 최근 M2M 및 IoT 시대로 접어 들음에 따라 센서 및 각종 디바이스 간에 상호 통신할 수 있는 공통 프로토콜이 필요하게 되었다. IoT 관련 표준화로는 ITU(International Telecommunication Union) 및 ETSI(European Telecommunication Standards Institute)가 주로 관련 단체 조율 등 큰 그림을 그리고 있으며, IETF

에서 IP 기반 IoT 표준화를 주도하고 있다. 또한 IPSO(Internet Protocol for Smart Objects Alhance)에서는 IETF 표준이 빨리 자리잡을 수 있도록 호환성 테스트, 사례 연구, 화이트 페이퍼 발간 등 지원을 하고 있다. 따라서, 차세대 센서 접속 프로토콜로 IETF CoAP에 많이 사용 될 것으로 보인다.

IETF CoRE 워킹그룹은 M2M 노드간에 통신할 수 있는 CoAP프로토콜을 중심으로 프로토콜을 개발하고 있다. [Figure 2.13]은 IETF 79차 회의 슬라이드에서 발췌한 것으로, CoRE 워킹그룹의 표준화 범위를 나타내고 있다.




[Figure 2.10] IETF CoRE WG 표준화 범위

CoRE 노드의 제약사항, 즉 RAM, ROM 등 메모리 크기를 비롯하여, 손실 및 전송속도, 지연 같은 네트워크의 제약사항, 노드의 전력 및 슬리핑 모드에 대한 고려, 리소스 등록 및 탐색, 신뢰성 있는 전달, 이벤트 통지 방법, 멀티캐스트, Proxy에서의 캐싱 및 HTTP 매핑, 데이터 포맷, 보안 고려 등이 표준화 범위에 해당한다.

현재 CoAP 표준화가 마무리 되어 감에 따라, 구현 업체 간 상호 연동성 시험을 위한 Plugtest가 ETSI, ProbeIT, IPSO 공조로 진행되어 왔고, 2013년 11월에 제 3차 시험이 완료 되었으며, ETRI도 이 시험에 참여하였다.

<표. 최신 정보 및 선행연구 분석>

| 기존정보조사내용 | 문헌명 및 저자 | 문헌의 핵심 내용 | 본 과제와의 공통점 | 본 과제와의 차이점 |
|---|---|---|---|---|
| | **[문헌명]** RFC 7252 – The Constrained Application Protocol | 사물인터넷 표준 프로토콜인 CoAP에 대한 문서이다. CoAP message format, message 전송 방식, request/response 동 | CoAP의 표준 규약을 준수하여 메시지를 생성하였고, 통신할 때 정해진 method와 uri를 사용하였다. | 본 과제에서는 보안계층을 별도로 두어 보안이 적용된 CoAP 통신을 하였다. |

| | | | | |
|---|---|---|---|---|
| | **[저자]**<br>Z. Shelby<br>K. Hartke<br>C. Bormann | 작 방식, CoAP method (GET, PUT, POST, DELETE) 등을 포함한다. | | |
| | **[문헌명]**<br>RFC 6347 – Datagram Transport Layer Security Version 1.2<br><br>**[저자]**<br>E. Rescorla<br>N. Modadugu | DTLS Handshake protocol, Record protocol 등을 설명한 표준 문서이다. | DTLS를 사용하여 패킷을 암호화하고, 복호화 하였다. | 본 과제에서는 application protocol로 CoAP을 사용하였으며, 보안을 적용하기 위해서는 DTLS를 경량화한 TinyDTLS를 적용시켰다. |
| | **[문헌명]**<br>IETF CoAP 기반 센서 접속 프로토콜 기술 동향<br><br>**[저자]**<br>고석갑<br>박일균<br>손승철<br>이병탁 | CoAP 표준화 및 observe, blockwise transfer과 같은 최신 동향에 대한 내용을 포함한다. | 사물인터넷 표준 프로토콜인 CoAP을 과제에 사용하였다. | 본 과제에서는 Observe, blockwise와 같은 최신 옵션을 사용하지 않는다.<br>본 과제에서는 보안 프로토콜을 사용하여 암호화된 CoAP 통신을 지원한다. |
| **활용결과** | 1) RFC 7252: CoAP에 대한 전반적인 이해를 하는 데 도움이 되었다. 구현할 때에는, server의 resource 제작 시 문서에 나와있는 format에 따라서 하였고, 그 resource를 얻기 위한 GET, PUT, POST, DELETE method 또한 문서에 나와있는 format에 따라 제작하였다.<br><br>2) RFC 6347: DTLS에 대한 전반적인 이해를 하는 데 도움이 되었다. DTLS handshake의 각 phase에서 전달되는 메시지를 정확하게 파악하여, CoAP source와 결합을 할 때에 참고하였다. 특히, 핵심적인 interface를 찾는 데에 도움이 되었다.<br><br>3) IETF CoAP 기반 센서 접속 프로토콜 기술 동향: 사물인터넷을 상용화 하기 위해서는 표준 프로토콜이 필요한데, 이는 CoAP이다. 최신 기술 동향은 이 표준 프로토콜을 사물인터넷에 특화된 hardware에 적용하여 통신하는 것 | | | |

| | |
|---|---|
| | 이다. 따라서 우리는 OpenMote-CC2538에 CoAP을 fusing함으로써 embedded system을 구현하였다. |

## 2.2 Review of Industrial Standards and Regulation

### 2.2.1 산업 표준

전 세계 사물인터넷 서비스 플랫폼 표준을 개발하기 위하여 출범된 oneM2M에서 후보 릴리즈 1 규격이 승인 및 완료되었다. 이번에 완료된 후보 릴리즈에서는 요구사항, 아키텍처, 프로토콜, 보안과 관련된 9개의 기술규격을 패키지로 제공한다.

종래 사물인터넷 표준기술 및 서비스들은 스마트홈, 스마트카, 스마트그리드 및 헬스케어와 같이 각 산업분야에 특화된 표준기술이 수직적(Vertical)으로 개발되고 서비스가 제공되었다. 따라서 해당 표준기술을 다른 산업분야에 사용할 경우 시스템의 설치, 확장 및 유지보수와 관련하여 추가적인 비용이 많이 발생하고 개발시간이 오래 소모되는 문제점을 가지고 있었다. 이에 oneM2M에서는 사물인터넷 서비스와 관련된 다양한 산업분야에서 공통적으로 사용될 수 있는 수평적(Horizontal)인 서비스 플랫폼을 지향하여, 산업분야에 특화된 서비스뿐만 아니라 산업분야 간 (Ex. 스마트홈 & 스마트카) 융합 서비스를 제공할 수 있는 기술을 개발하였다.

#### 2.2.1.1 요구사항(WG1/Requirements)

워킹그룹 1에서는 oneM2M 공통서비스 제공을 위해 [Figure 2.11]와 같이 다양한 M2M 서비스 분야의 유스케이스 33개를 분석하여 일반 요구사항, 장치 관리 요구사항, 보안 및 과금 요구사항 등 140여 개의 요구사항을 개발하였다.

#### 2.2.1.2 아키텍처(WG2/Architecture)

워킹그룹 2에서는 oneM2M에서 지원하는 네트워크 아키텍처와 아키텍처를 구성하는 엔티티 및 oneM2M 공통 서비스 기능(CSF: Common Service Function)과 이를 제공하기 위한 공통 서비스 계층에서의 레퍼런스 포인트를 정의한다.

oneM2M에서 지원하는 네트워크 아키텍쳐는 애플리케이션 전용 노드 (ADN: Application Dedicated Node), 애플리케이션 서비스 노드(ASN: Application Service Node), 중간노드(MN: Middle Node) 및 인프라스트럭처 노드(IN: Infrastructure Node)로 구성될 수 있다.

- 애플리케이션 전용 노드: 사물인터넷 애플리케이션을 포함하는 장치로써, 사물인터넷 서비스 로직만을 포함하여 제한된 기능을 가지는 제한적인 (Constrained) 장치
- 애플리케이션 서비스 노드: 사물인터넷 애플리케이션뿐만 아니라 공통 서비스를 제공하는 사물인터넷 장치
- 중간노드: 장치 노드들과 네트워크 인프라스트럭처 노드를 연결해주는 사물인터넷 게이트웨이
- 인프라스트럭처 노드: 네트워크 인프라스트럭처에 위치해 사물인터넷 서비스를 제공해주는 사물인터넷 서버

| Application | Use Cases | | | | |
|---|---|---|---|---|---|
| Smart Home | HEMS[1] | Electrical Charging Vehicles at Home | Semantic Home Control | Semantic Device Plug and Play | Event Triggered Task |
| Smart Car | Vehicle Diagnostic & Maintenance Report | Remote Maintenance services | V2V[2] | Fleet mgmt | |
| Energy | Wide Area Energy Control | Analytics for oneM2M | Smart Meter Reading | Energy Monitoring using Satelite | Oil/Gas Pipeline GW |
| Heathcare | Health care GW | Wellness services | Remote Patient care | | |
| Enterprise | Smart building | | | | |
| Public Service | Street Light Automation | Car/Bicycle Sharing Services | Smart Parking | Public Safety | |
| Other | M2M Access Network using satellite | 3GPP Interworking | Sleeppy Device Control | | |

1) Home Energy Management System, 2)Vehicle to Vehicle

**[Figure 2.11]**

#### 2.2.1.3 프로토콜(WG3/Protocols)

워킹그룹 3에서는 엔티티 간 메시지(oneM2M Primitive) 교환 프로토콜, 에러 처리를 다루는 oneM2M 코어 프로토콜 그리고 코어 프로토콜과 전송계층 프로토콜(CoAP, HTTP, MQTT 등)과의 바인딩을 정의하였다. oneM2M 코어 프로토콜은 Mca, Mcc 그리고 Mcc' 레퍼런스 포인트 상에 정의된 API를 의미한다. oneM2M 코어 프로토콜은 전송계층 프로토콜과 독립적으로 설계되어 있으며 HTTP, CoAP 및 MQTT와 같이 다양한 프로토콜과의 바인딩을 지원한다. 또한, 보안 프로토콜로는 DTLS와 TLS를 지원한다.

#### 2.2.1.4 보안(WG4/Security)

워킹그룹 4에서는 oneM2M 서비스 플랫폼의 보안 기능으로 기기 간의 인증(Authentication), 데이터의 기밀성(Confidentiality) 및 무결성(Integrity)에 대한 기술을 정의하였다. 규격에서는 프로토콜의 특성 고려 및 규격 개발 일정상 우선순위를 정해서 단대단 보안 기능보다는 Mcc와 Mca 레퍼런스 포인트를 기반으로 공통 서비스 엔티티 간 Hop by hop 보안 기능을 제공하며, 공통 서비스 엔티티 간의 보안 기능은 필수로 공통 서비스 엔티티와 애플리케이션 엔티티 간은 옵션으로 제공한다.

### 2.2.2 법령 조항

#### 2.2.2.1 전기통신기본법

제 1장 제 2조 1항 "전기통신"이라 함은 유선·무선·광선 및 기타의 전자적 방식에 의하여 부호·문언·음향 또는 영상을 송신하거나 수신하는 것을 말한다.

제 1장 제 2조 2항 "전기통신설비"라 함은 전기통신을 하기 위한 기계·기구·

선로 기타 전기통신에 필요한 설비를 말한다.

#### 2.2.2.2 정보통신기반 보호법

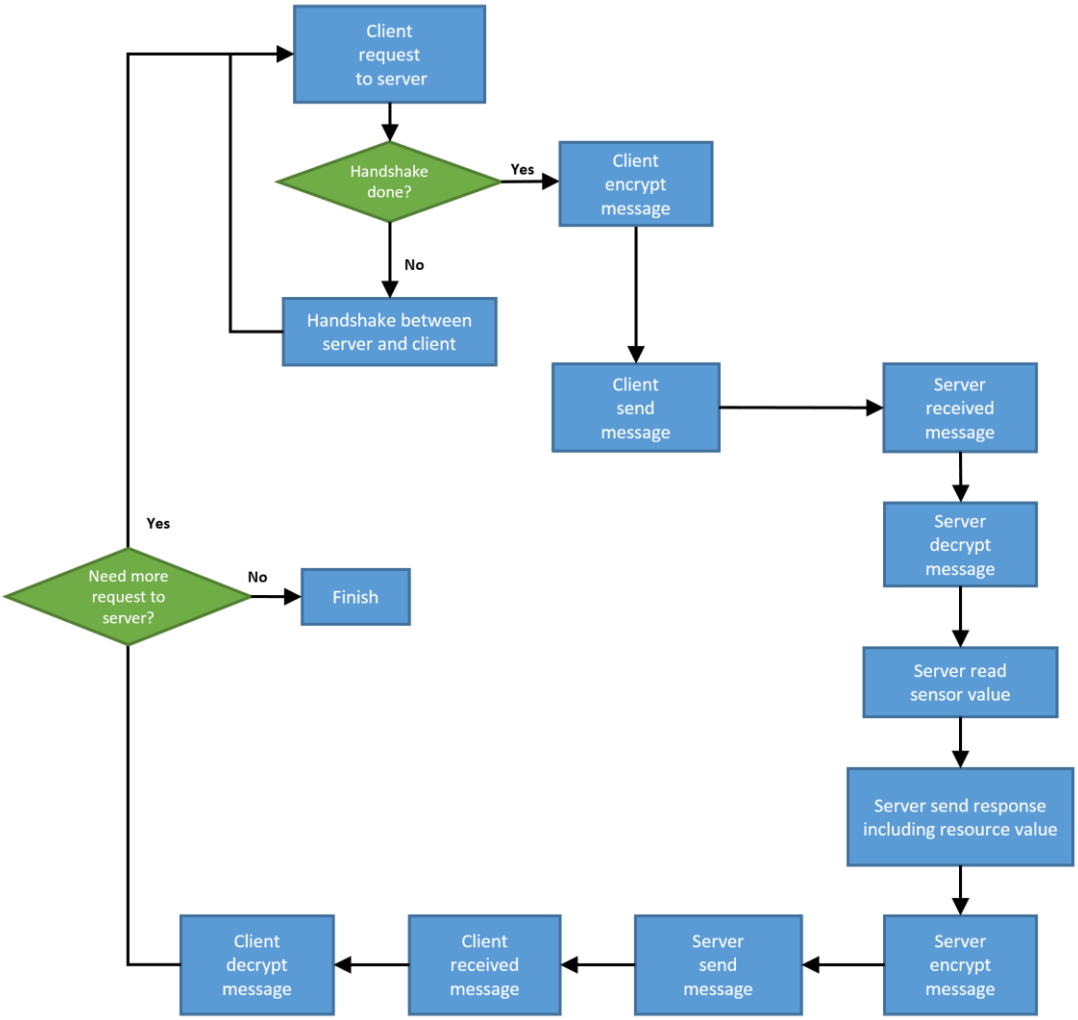
제 1장 제 1조(목적) 이 법은 전자적 침해행위에 대비하여 주요정보통신기반시설의 보호에 관한 대책을 수립·시행함으로써 동 시설을 안정적으로 운용하도록 하여 국가의 안전과 국민생활의 안정을 보장하는 것을 목적으로 한다.

제 1장 제 2조 1항 . "정보통신기반시설"이라 함은 국가안전보장·행정·국방·치안·금융·통신·운송·에너지 등의 업무와 관련된 전자적 제어·관리시스템 및「정보통신망 이용촉진 및 정보보호 등에 관한 법률」제2조 제1항 제1호의 규정에 의한 정보통신망을 말한다.

제 1장 제 2조 2항 "전자적 침해행위"라 함은 정보통신기반시설을 대상으로 해킹, 컴퓨터바이러스, 논리·메일폭탄, 서비스거부 또는 고출력 전자기파 등에 의하여 정보통신기반시설을 공격하는 행위를 말한다.

## 3. Proposed Design and Solution

### 3.1 Overall System Architecture




[Figure 3.1] 시스템 블록도

### 3.1.1 기능 블록으로 나눈 기준

**3.1.1.1** Server와 Client의 메시지 생성 과정

**3.1.1.2** 메시지 전송 과정

### 3.1.2 각 블록의 기능

**3.1.2.1** Client request to server

Client가 server에게 얻고자 하는 resource를 CoAP + TinyDTLS통신을 사용하여 request한다.

**3.1.2.2** Handshake between server and client

만약 Handshake가 되지 않은 상태라면 Handshake process를 진행한다.

Handshake process는 아래와 같다.

**3.1.2.2.1** Client Hello

**3.1.2.2.2** Server Verify Request

**3.1.2.2.3** Client Hello with Cookie

**3.1.2.2.4** Server Hello

**3.1.2.2.5** Certificate

**3.1.2.2.6** Server Key Exchange

**3.1.2.2.7** Certificate Request

**3.1.2.2.8** Server Hello Done

**3.1.2.2.9** Certificate

**3.1.2.2.10** Client Key Exchange

**3.1.2.2.11** Certificate Verity

**3.1.2.2.12** Change Cipher Spec

**3.1.2.2.13** Finished

**3.1.2.2.14** Change Cipher Spec

**3.1.2.2.15** Finished

**3.1.2.3** Client encrypt message

만약 Handshake가 완료되었다면 TinyDTLS 프로토콜에 따라 메시지를 암호화한다.

**3.1.2.4** Client send message

Client가 Server에게 암호화된 메시지를 전송한다.

**3.1.2.5** Server received message

Client가 보낸 암호화된 메시지를 Server가 수신한다.

**3.1.2.6** Server decrypt message

Server는 TinyDTLS 프로토콜에 따라 메시지를 복호화한다.

**3.1.2.7** Server read sensor value

Server는 request 받은 resource에 상응하는 sensor value를 읽는다.

**3.1.2.8** Server send response including resource value

읽은 sensor value를 포함한 response메시지를 전송한다.

**3.1.2.9** Server encrypt message

Server가 TinyDTLS 프로토콜에 따라 전송할 메시지를 암호화한다.

**3.1.2.10** Server send message

Server가 Client에게 암호화된 메시지를 전송한다.

**3.1.2.11** Client received message

Server가 보낸 암호화된 메시지를 Client가 수신한다.

**3.1.2.12** Client decrypt message

Client는 TinyDTLS 프로토콜에 따라 메시지를 복호화 한다.

**3.1.2.13** Finish

만약 client가 server로 더 요청할 데이터가 없으면 종료한다.

### 3.1.3 각 블록의 상호 작용

먼저 Client가 얻고자 하는 resource를 Server에게 "Client request to server" 블록을 통해 request한다. Handshake 여부에 따라 그 다음 행위가 결정이 된다. 만약 Handshake 과정을 거치지 않았으면 "Handshake between server and client" 블록으로 가게 된다. 그리고 다시 "Client request to server" 블록으로 돌아가서 server에게 request를 보낸다. 만약 Handshake 과정을 거친 후라면, "Client encrypt message" 블록을 통해 메시지를 암호화한다. 그리고 "Client send message"를 통하여 server에게 메시지를 전송한다. Server는 "server received message" 블록을 통하여 메시지를 수신한 뒤, "server decrypt message" 블록에서 메시지를 복호화 한다. 메시지에서 요구된 resource에 상응하는 sensor value를 "server read sensor value" 블록 과정에서 읽는다. 그리고 "server send response including resource value" 블록에서 메시지를 전송하고, "server encrypt message" 블록에서 메시지를 암호화 한다. "server send message" 블록에서는 메시지를 client에게 전송한다. "Client received message" 블록에서 client는 메시지를 수신한 후, "client decrypt message" 블록에서 메시지를 복호화 한다. 만약 client가 server에게 더 보낼 메시지가 있다면 다시 "client request to server" 블록으로 돌아가서 위 과정을 반복한다. 만약 더 보낼 메시지가 없다면 "Finish" 블록으로 가서 프로그램을 종료한다.

## 3.2 Detail Design

### 3.2.1 libcoap + tinydtls library 병합

본 프로젝트에서는 오픈 소스 라이브러리인 libcoap과 tinyDTLS를 합치기 위해, 두 개의 라이브러리를 각각 폴더에 따로 두고 부모 디렉토리에서 서버와 클라이언트 코드를 만들어 직접 정의한 Makefile을 통해 빌드 한다.

본 프로젝트에서는 함수와 변수에 대하여 합치는 부분, 한 쪽만 남기는 부분, 따로 두는 부분을 직접 정의한다. 합치는 변수는 다중 입출력을 위한 fd_set형 readfds, 조정을 위한 timeval형 tv, select기법의 결과값을 담는 int형 result, port번호를 저장하기 위한 short형 port, cmdline의 option part를 위한 int형 opt, log level을 위한 coap_log_t형 log_level 등이 있다. 합치는 함수는 구조가 비슷한 resolve_address()가 있다. 두 코드를 합치는 데에 구심적 역할을 하는 socket함수 와 그 binding 요소에 대해서는 coap과 tinyDTLS 양쪽 중 coap 쪽만 남긴다.

그 외의 부분은 따로 두도록 한다.

특별히, sendqueue, message_id등 보내는 coap 메시지의 전체 정보가 들어가 있는 coap_context_t 구조체 와 cookie_secret, peers등 dtls의 세션유지및, 암호화 정보가 들어가 있는 dtls_context_t 구조체를 따로 두어 구현의 편리성을 더한다.

Send와 Receive함수에 대해서는 각각 API의 호출 순서를 종속시킴으로써 구현한다.

#### 3.2.1.1 Client

[병합한 library의 main 함수에서 기존 libcoap에서는 없던 handshake 과정이 추가 되었다. 그리고 handshake 과정과 CoAP 메시지 전송 및 암호화 과정을 분리 시키기 위해 'isHandshakeFinished' 라는 변수를 추가해 주었다.

```
int fd = 0;     //for copy socketFD
int isHandshakeFinished = 0;
```

**[Figure 3.2]**

그리고 dtls_handle_read함수에는 dtls에 관련된 여러 메시지들을 종류별로 처리해주는데, 이 때 handshake 관련 메시지 이외의 메시지도 처리해준다. 그러므로 핸드쉐이크가 정상적으로 종료되기 위해선 dlts_handle_read 함수가 handshake의 마지막에 해당하는 메시지를 받았을 때 handshake 과정이 종료 되도록 해야한다.

이를 위해, dtls_handle_read 함수에 isHandshakeFinished 변수를 넘겨주어 handshake의 마지막 과정이 이루어질 때, isHandshakeFinished값을 1로 바꾸어주어 while문이 종료되게 하여 handshake 과정을 마치도록 한다.

```
while (isHandshakeFinished == 0) {
   FD_ZERO(&readfds);

   FD_SET(fd, &readfds);
   /* FD_SET(fd, &wfds); */

   tv.tv_sec = 5;
   tv.tv_usec = 0;

   result = select(fd+1, &readfds, 0, 0, &tv);

   if (result < 0) {   /* error */
     if (errno != EINTR)
 perror("select");
   } else if (result == 0) { /* timeout */
   } else {       /* ok */

     if (FD_ISSET(fd, &readfds))
       dtls_handle_read(dtls_context, &isHandshakeFinished);
   }
 } ? end while isHandshakeFinished==0 ?
```

**[Figure 3.3]**

#### 3.2.1.2 Server

먼저 client와 비슷한 메커니즘으로 libcoap에서 request를 받고 response를 처리하는 과정 전에 socket file descriptor를 coap_context_t구조체의 sockfd field에서 가져

온후 dtls_new_context 함수의 파라미터에 집어 넣음으로써, libcoap과 tinydtls의 context를 만들어 주고, handler를 활성화 시켜 준다. 또한, handshake과정과 그 이후 과정 (CoAP request를 decrypt하고, 그에 따라 encrypt된 response를 만드는 과정)을 다로 분리 시키기 위하여 isHandshakeFinished 변수를 준비한다.

```
dtls_handle_read(the_context, &isHandshakeFinished);
```

**[Figure 3.4]**

우선 handshake과정을 모두 처리 하기 위하여 [Figure 3.4]와 같이 dtls_handle_read를 server코드의 while문 안에서 호출하고, while의 종료 조건을 isHandshakeFinished가 1일 경우로 두면 된다.

dtls_handle_read함수 안에서는 dtls_handle_message라는 함수를 호출하게 되는데 dtls와 관련된 메시지에 대하여 처리 하는 과정이 나와 있다. 특별히 받은 메시지가 change cipher spec / alert / handshake / application data일 경우에 따라 각각 처리 하는 부분이 나와 있다. 특별히 handshake일 경우에는 거기에 따른 response에 대한 부분도 이 dtls_handle_message함수에서 다 처리하게 된다. 따라서 dtls_handle_read를 while문 안에서 호출하면 handshake가 모두 끝나게 된다.(dtls_handle_message함수는 후에 coap message를 encrypt하는 데에도 쓰인다.)

이후 다른 while문을 통하여 coap message를 decrypt, encrypt 하게 된다. 여기서 coap_read, coap_dispatch 함수가 사용된다. 이에 대한 자세한 내용은 3.2.1.3, 3.2.1.4를 통하여 자세히 설명하도록 한다.

#### 3.2.1.3 Read

Receive의 경우, main함수에서 coap의 read함수를 호출함으로써 진행이 된다. 후에 dtls_handle_message() 를 호출할 때 필요한 파라미터인 dtls_context를 위하여 [Figure 3.5]과 같이 API를 수정시켜 준다. coap_read()함 수안에서 recvfrom()함수를 호출하게 되는데, 이 recvfrom()부분을 지우고 dtls_handle_message()함수를 집어 넣어 준다. 여기서 복호화된 메시지가 들어가는 clear라는 uint8 array의 주소 값을 parameter로 넣을 수 있게 끔 API를 수정시켰다. 함수 호출 후 clear를 coap_hdr_t* 형으로 변환시켜서 pdu에 넣음으로써 이후에 coap메시지에 대한 정보를 parsing할 수 있게끔 한다[Figure 3.6]

```
coap_read( ctx );
coap_dispatch( ctx );
        ↓
coap_read( ctx ,dtls_context);
coap_dispatch( ctx ,dtls_context, &dst);
```

**[Figure 3.5] ReadAPI modification**

```
dtls_handle_message(dtls_ctx, &session, clear, buf,
bytes_read, &isHandshakeMessage);
pdu = (coap_hdr_t *)clear;
```

**[Figure 3.6] CoAP 메시지 복호화 과정**

coap_read()에서 parsing 한 후 coap_dispatch()함수를 통하여 메시지를 해석하고 그에 따른 동작을 수행한다. 이 함수에서 response를 처리하는 과정 중 예상 치 못한 response로 인하여 retransmission을 하는 부분이 있는데, 이때 coap_send() 함수 호출에 필요한 파라미터를 위하여 coap_dispatch() 자체의 API를 [Figure 3.5]와 같이 수정시켜준다.

#### 3.2.1.4 Write

Handshake 과정 이후에는 CoAP 메시지 생성 및 CoAP 메시지 암호화, 전송 과정이 이루어진다. CoAP 메시지 생성 및 전송은 con 메시지의 경우에는 coap_send_confirmed 함수에서 이루어지고 나머지 메시지의 경우에는 coap_send 함수에서 이루어진다. 그러나 암호화에 관한 작업은 이 함수들 안에는 없기 때문에 tiny-dtls 라이브러리에서 암호화를 해주는 부분을 찾아 이들 함수 내부에 추가해 주어야 한다.

coap_send 함수 내부에 있는 coap_send_impl 함수 안의 sendto 함수에서 만들어진 coap 메시지가 전송 되는데 sendto 함수 대신 tiny-dtls 코드 내부의 암호화 및 전송 함수를 사용한다.

TinyDTLS의 dtls_write 함수가 이 역할을 하므로 sendto 함수가 있는 위치에 dtls_write 함수를 배치한다.

```
if (pdu->hdr->type == COAP_MESSAGE_CON)
    tid = coap_send_confirmed(ctx, dtls_context, &dst_coap, &dst, pdu);
else
  tid = coap_send(ctx, dtls_context, &dst_coap, &dst, pdu);
```

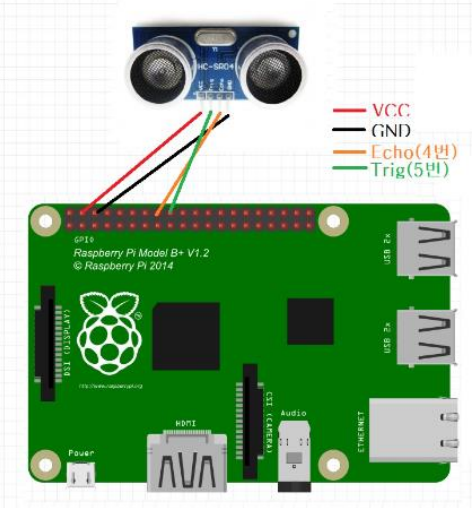
```
res = dtls_write(context_dtls, dst_dtls, (uint8 *)ptr_pdu_dtls->hdr, pdu->length);
```

**[Figure 3.7]**

### 3.2.2 libcoap resource 제작

#### 3.2.2.1 Ultrasonic sensor 연결

라즈베리파이에 Ultrasonic sensor를 연결시키기 위해서는 빵판과 초음파센서(HC-SR04), 점퍼선이 필요했다. 본 프로젝트에서는 HC-SR04센서를 빵판에 연결시키고 점퍼선을 라즈베리파이 J8 header에 직접 연결(GPIO연결)시키는 방식을 취하게 된다.




**[Figure 3.8]**

이러한 방식으로 연결하되 HC-SR04센서에 직접 연결하는 것이 아니라, bread board를 통하여 연결한다.

GPIO 4,5번을 이용하게 되는데 GPIO4에 ECHO를 연결, GPIO5에 TRIG를 연결해 준다.

본 센서는 wiringPi를 통하여 제어하게 된다 라즈베리파이에 wiringPi패키지를 설치한 이후에 센서를 이용하는 libcoap의 server코드에 wiringPi.h 헤더를 추가 시켜 준다. 개발의 편의성을 위하여 TRIGI와 ECHO에 대하여 5와 4로 server의 헤더부분에 define 해준다. 본 센서에는 client에서 request가 올 때만 값을 읽도록 하는 메커니즘을 적용시킨다. 이를 위하여 server의 코드상에서 hnd_get_distance 함수 내에서 get request에 대한 payload를 준비시키는 과정에 거리를 측정하는 메커니즘을 추가시키도록 한다.

HC-SR04데이터 시트에 의하면 이 센서의 원리는 TRIG핀을 통해 특정 파형을 계속해서 만들고, 일정 파형을 만들면 ECHO핀의 상태가 HIGH로 변경되게 되는 것이다. HIGH상태에서 초음파를 감지하면 LOW상태로 변경이 된다. 거리를 구하기 위해서는 이 때 ECHO핀이 HIGH에서 LOW로 변경되는 시간을 측정하고 이 micors로 측정한 시간을 58로 나누면 된다.

Centimeters = (HIGH가 끝나는 시간 – LOW가 끝나는 시간) / 58

```
while(digitalRead(ECHO1) == LOW);
    startTime1 = micros();
while(digitalRead(ECHO1) == HIGH);
    travelTime1 = micros() - startTime1;

distance1 = travelTime1 / 58;
```

**[Figure 3.9]**

따라서 루프를 통하여 ECHO1이 HIGH가 될 때 가지 기다리고, 바뀌는 순간의 시간을 기록한다. 여기서 digitalRead라는 함수는 ECHO핀에 전압이 들어

올 때는 HIGH를 return하고, 전압이 들어오고 있지 않을 경우에는 LOW를 return하게 된다.

이후 ECHO1에 대하여 LOW가 될 때 까지를 기다려서 그 사이의 시간을 travelTime1변수에 할당한 후 58로 나눠서 distance1에 대입하면 centimeters 단위의 거리가 distance1에 저장되게 된다.

```
sprintf(buf, "%d", distance1);
strcat(buf, "\0");
coap_add_data(response, len, buf);
```

**[Figure 3.10]**

이후 sprintf함수를 통해 buf string 변수에 거리 value를 복사해 주고, coap_add_data함수를 통하여 payload에 값을 추가함으로써 거리 value를 distance resource와 연동시키게 된다.

또한, 이 wiringPi.h가 포함된 server.c를 컴파일하기 위하여 make file의 server.c 부분에 –lwiringPi 라이브러리 추가 명령어를 포함시켜준다.

#### 3.2.2.2 Resource 추가 및 GET method 등록

본 프로젝트에서는 ultrasonic sensor의 거리를 cm 단위로 측정하는 것을 resource로 정하였으며, resource의 이름은 distance라 하였다.

```
int distance1 = 0;
```

**[Figure 3.11]**

[Figure 3.12]는 response에 distance resource의 value를 추가하는 과정이다.

```
coap_add_data(response, len, buf);
```

**[Figure 3.12]**

### 3.2.3 JAVA GUI 제작

Java로 distance에 따라 다른 화면을 보여줄 수 있도록 구현하였다. 기본 그림은 아래 [Figure 3.13]와 같다.


**[Figure 3.13]**

만약 거리가 20cm 이하가 된다면 아래 [Figure 3.14]의 그림으로 바뀐다.


**[Figure 3.14]**

핵심 부분의 코드만 pseudo code로 설명하면 아래 [Figure 3.15]와 같다.

```
while(1)
{
    while(1)
    {
      distance = getDistance();
      if (distance <= 20) break;
      showOffImage();
      sleep(0.8sec)
    }
    showOnImage();
    sleep(0.8sec)
}
```

**[Figure 3.15]**

이중 while문을 사용해서 [Figure 3.13]에 있는 경보 OFF 이미지를 보여주고 있다가, 만약 거리가 20 cm 이하가 되면 inner while 문을 탈출한 뒤 [Figure 3.14]에 있는 경보 ON 이미지를 보여준다.

### 3.2.4 OpenMote-CC2538 (진우)

Openmote-CC2538을 위한 작업 환경은 리눅스 PC이다. 리눅스 PC에 설치된 JLink-GDBServer를 통해 코드를 크로스 컴파일 한 후, Openmote-CC2538 위에 업로드를 한다. Openmote-CC2638 위에서 실행시키는 코드는 github에서 제공하는 https://github.com/cetic/6lbr를 사용하였다. 코드의 버전으로 인한 충돌을 방지하기 위해 'develop' branch 항목에서 가장 최신 버전으로 먼저 다운을 받는다. 6LBR은 임베디드 하드웨어를 지원하기 위한 코드를 제공하는데, 하드웨어가 무선 네트워크 환경에서 통신할 수 있는 네트워크 및 운영체제 구조를 구현하고 있다. 본 프로젝트에서는 6LBR 오픈 소스를 활용하되, 6LBR에 포함되어 있는 6lbr-demo(CoAP)와 6LBR에서 제공하지 않는 TinyDTLS 보안 프로토콜을 결합하여 실제 무선 네트워크 통신에서 resource가 암호화 되도록 디자인을 한다

#### 3.2.4.1 Build & Fusing

-Slip-radio : Openmote가 클라이언트로 동작시키기 위해 올려지는 소스코드이다. 이 Openmote-CC2538은 Xbee Explore Dongle을 통해 라즈베리 파이에 연결된다. Slip-radio 코드는 네트워크 스택에서 Mac layer(2계층)의 일부 계층과 1계층을 담당하고 상위 네크워크는 라즈베리파이에 설치된 6LBR이 지원한다. Slip-radio를 Build 하기 위한 실행 명령어는 다음과 같다. (make 명령 시, 퓨징하게 될 하드웨어의 플랫폼을 TARGET="" 구조로 명령을 해주어야 한다.)

```
cd $CONTIKI_HOME/examples/ipv6/slip-radio
make TARGET=openmote
```

**[Figure 3.16]**

-6lbr-demo : Openmote가 서버로 동작하기 위한 소스코드이다. 6lbr-demo는 UDP와 CoAP 계층이 하드웨어 상으로 동작시키게 한다. 본 프로젝트에서는 CoAP과 Tiny DTLS의 결합을 위해 두 코드를 같은 디렉토리 안에 포함시키고 패킷이 전달 될 때, 충돌을 방지하기 위해 두 계층간의 코드상의 인터페이스 부분을 수정한다. 이 과정이 마치면, 6lbr-demo 를 Build 하기 위한 실행 명령어는 다음과 같다. (make 명령을 실행하기 전, TinyDTLS는 별도의 auto configuration 명령을 해 주어야 configure script가 생성되어 6lbr-demo와 같이 build될 수 있다.)

```
cd $CONTIKI_HOME/examples/6lbr-dem
make TARGET=openmote WITH TINYDTLS=1 WITH COAPSERVER=1
```

**[Figure 3.17]**

-Fusing : 위의 make 과정을 통해 생성된 실행파일을 크로스컴파일을 하여 Openmote-CC2538에 퓨징하기 위해서는 먼저 JLinkGDBServer를 실행한 후, 컴파일과 퓨징 과정에 필요한 speed와 host 등 을 지정해 주어야 한다. 전체 명령어는 다음과 같다.

```
$ arm-none-eabi-gdb
$ target remote localhost:2331
$ monitor speed 2000
$ monitor endian little
$ monitor flash download = 1
$ monitor reset
$ load
$ break;
```

**[Figure 3.18]**

### 3.2.5 설계 문제의 해결책을 최적화 하는 과정과 최적의 해결책임을 보여라

#### 3.2.5.1 Hardware 선정

본 프로젝트에서는 server로 OpenMote-CC2538을 사용하였고, client 및 라우터로 Raspberry pi 2 B+를 사용하였다. 또한 시뮬레이션 프로그램을 만들기 위해 Raspberry pi 상에 client와 server를 모두 구현하였다.

##### 3.2.5.1.1 OpenMote-CC2538

사물인터넷 환경은 자원이 부족하기에 이 문제를 해결하기 위하여 자원 사용 및 기타 사물인터넷 환경을 지원하는 hardware를 선택해야 한다. 본 팀은 OpenMote-CC2538을 선택하였다. 왜냐하면 Contiki, FreeRTOS, OpenWSN, Riot과 같은 사물인터넷 대표 프로젝트의 S/W를(open-source) 지원하는 특수 Hardware이기 때문이다. 또한, 기존 블루투스 모듈인 XBee와 핀 호환이 되도록 만들어 졌으며, OpenMote 모듈과 OpenBase를 결합하면 이더넷 게이트웨이 역할을 수행할 수 있다. 특히 OpenMote는 현재 사물인터넷의 표준 프로토콜 스택인 IETF 6TiSCH, IEEE802.15.4e TSCH 등을 구현하기에 적합하도록 설계 되었으며, 본 프로젝트의 풀 스택을 fusing하기에 최적화된 Hardware이다.

##### 3.2.5.1.2 Raspberry pi 2 B+

본 프로젝트에서의 사물인터넷 구현 모델에서 라즈베리파이는 WSN과 유선 인터넷망 사이의 Border Router 역할을 하고 있다. 현재 라즈베리파이와 호환 가능한 여러 센서들이 많이 제작되고 있는 상황이며, Microsoft 회사 역시 사물인터넷 시대에 맞추어 기존에 있던 라즈비안 OS의 IoT에 대한 한계점을 인식하고 라즈베리파이3를 위한 '윈도우 10 IoT OS'를 제공하는 등 라즈베리파이가 사물인터넷 플랫폼에 적합하게 쓰일 수 있는 하드웨어 임을 인식하고 있는 추세이다. 또한, 6LBR이라는 6loWPAN Border Router Solution은 라즈베리파이를 Border Router로 쉽게 사용할 수 있게 하도록 지속적인 업데이트를 하고 있다. 이렇듯 본 프로젝트에서 라즈베리파이를 사물인터넷 모델에서 Client 혹은 Border Router의 역할로 사용한 것은 하드웨어 설계 해결책으로써 적합하다고 할 수 있다.

#### 3.2.5.2 TinyDTLS 사용

사물인터넷과 같이 저 사양 device로 이루어진 네트워크는 각 디바이스의 리소스 제약으로 인해 일반적인 네트워크에 비해 제한된 성능을 가진다. 따라서 기존의 무거운 보안 프로토콜 대신 본 프로젝트에서는 저 사양 노드 네트워크를 위한 보안 프로토콜 TinyDTLS를 제안한다. TinyDTLS는 저 사양 노드에도 고수준의 보안을 제공할 수 있도록 한다. TinyDTLS의 특징은 첫번째, 사물인터넷 주소체계로 IPv6를 지원함으로써 IPv4와 비교하여 옵션 필드 및 헤더 길이 필드가 없는 등 비교적으로 축소된 고정 크기의 header를 사용한다. 두번째, DTLS와 비교하여 최소한의 Cipher suite, 즉 TLS_PSK_WITH_AES_128_CCM_8, TLS_ECDHE_ECDSA_WITH_AES_AES_128_CCM_8 만을 적용하여 CoAP 메시지와의 통합성과 비밀성을 제공한다. 세번째, "AES_CCM" 모드를 사용하여 device들간의 CoAP 메시지 암호화를 선택적으로 적용할 수 있다. 이는 모든 디바이스의 메시지를 암호화함으로써 발생할 수 있는 리소스 낭비를 줄여준다.

## 4. Implementation and Evaluation

### 4.1 Implementation(구현)

#### 4.1.1 구현에 사용된 개발 도구

##### 4.1.1.1 Hardware

1) OpenMote-CC2538 client side: 라즈베리파이와 센서쪽 openmote과의 무선통신을 위하여 slip-radio code를 openmote에 fusing한 후 XBee Dongle을 통해 라즈베리파이와 연결한다. Slip-radio를 통해 라즈베리파이가 라우터로 동작할 수 있게 된다. 본 프로젝트에서는 channel를 25로 설정하였으며, MAC 프로토콜은 nullmac driver, RDC 프로토콜을 contikimac driver를 사용하였다.
2) OpenMote-CC2538 server side: 통신모듈에서 서버쪽 역할을 한다. 센서로부터 정보를 받아들여 클라이언트를 통해 다른 서버 (센서)에게 정보를 전달한다.
3) OpenBase: OpenMote에서 실행하는 프로그램에 대한 디버깅환경을 제공한다.
4) Open Battery: OpenMote에 Power를 공급하고 내장되어 있는 센서 중 하나인 SHT21 온/습도 센서를 resource로 사용한다.
5) Raspberry Pi 2 B+: 기본적으로 통신 모듈에서 Client역할을 한다. 센서 쪽의 정보를 서로 주고 받도록 하는 중간다리 역할을 한다
6) JLink Debugger: OpenMote에 소스코드를 크로스-컴파일할 때 사용한다.
7) XBee Explorer Dongle: 라즈베리파이에 센서 쪽과 ZigBee 통신환경을 제공하기 위해 ZigBee 모듈을 부착하고 ZigBee 모듈 위에 OpenMote를 부착한다.
8) SmartRF06 Evaluation board: OpenMote의 메모리를 flash할 때 사용한다.

##### 4.1.1.2 Software

1) cetic/6lbr: 라즈베리파이를 IPv6라우터로 동작 가능하게 해주는 프로그램이다.
2) CoAP – libcoap : Open Source로 공개된 CoAP의 c라이브러리인 libcoap을 라즈베리파이의 App Layer에 위치시킨다.
3) TinyDTLS: eclipse/kapua/tinydtls: Dtls를 CoAP 프로토콜에 최적화 되도록 수정한 TinyDTLS를 libcoap사이의 API쪽 코드수정을 통해 결합한다.
4) wheezy: 라즈베리파이에 설치한 OS이다.
5) SEGGER/JLinkGDBServer: OpenMote에 소스코드를 크로스-컴파일할 때 사용한다.
6) Flash programmer 2: OpenMote의 메모리를 flash할 때 사용한다.
7) putty: 라즈베리파이 terminal 원격조정 프로그램이다. 학교 안 어디서든지 로컬네트워크망에 연결되었을 때 라즈베리파이를 원격으로 조정하기 위하여 사용하였다.
8) xrdp: 라즈베리파이 GUI 원격조정 프로그램이다. 학교 안 어디서든지 로컬네트워크망에 연결되었을 때 라즈베리파이를 원격으로 조정하기 위하여 사용하였다.(특별히 GUI적 요소가 필요한 경우)
9) wireshark: 통신 패킷을 캡쳐하는 프로그램이다. 개발한 CoAP + TinyDTLS 코드가 실행될 때 적절한 Handshake이후에 CoAP메시지를 잘 암호화하는지 테스트하기 위하여 wireshark를 사용하였다.
10) eclipse: 자바 GUI 프로그램에 대하여 그 코드에 대한 작업과 결과 테스트를 위하여 자바

의 개발 툴인 eclipse를 사용하였다.
11) californium-scandium: californium은 CoAP 프로토콜과 그와 관련된 여러 라이브러리를 java로 구현한 프로젝트이다. californium의 subproject중 하나인 scandium은 DTLS1.2가 구현되어 CoAP를 암호화 해주는 역할을 한다. 본 프로젝트에서 구현한 CoAP + TinyDTLS 코드가 표준에 맞게 잘 구현 되었는지 판단하기 위하여 일반 상용화된 라이브러리인 californium-scandium의 코드를 이용하여 테스트하였다. 또한, 서버인 OpenMote와 통신하는 client program으로 사용하였다.

#### 4.1.1.3 팀원들의 역할

**App**

**1) 김진우:**

Centos 설치하여 개발환경 구축
라즈베리파이에 6lbr 환경 설정을 하였다.
OpenMote, OpenBattery, OpenBase 세팅
JLink Debbuger, JLink GDB Server 설치 및 구동
SmartRF06 Evaluation Board를 사용한 Flash programmer 2 사용

**2) 김연희:**

Centos 설치하여 개발환경 구축
OpenMote, OpenBattery, OpenBase 세팅
JLink Debbuger, JLink GDB Server 설치 및 구동
라즈베리파이에 6lbr 설치
SmartRF06 Evaluation Board를 사용한 Flash programmer 2 사용
CoAP resource 제작
JAVA GUI 제작
라즈베리파이 환경설정
6lbr+tinydtls 디버깅

**3) 한영광:**

libcoap + tinydtls 병합
Wireshark를 통한 libcoap + tinydtls 병합 코드 자체 평가
라즈베리파이에 6lbr 설치
라즈베리파이 환경설정

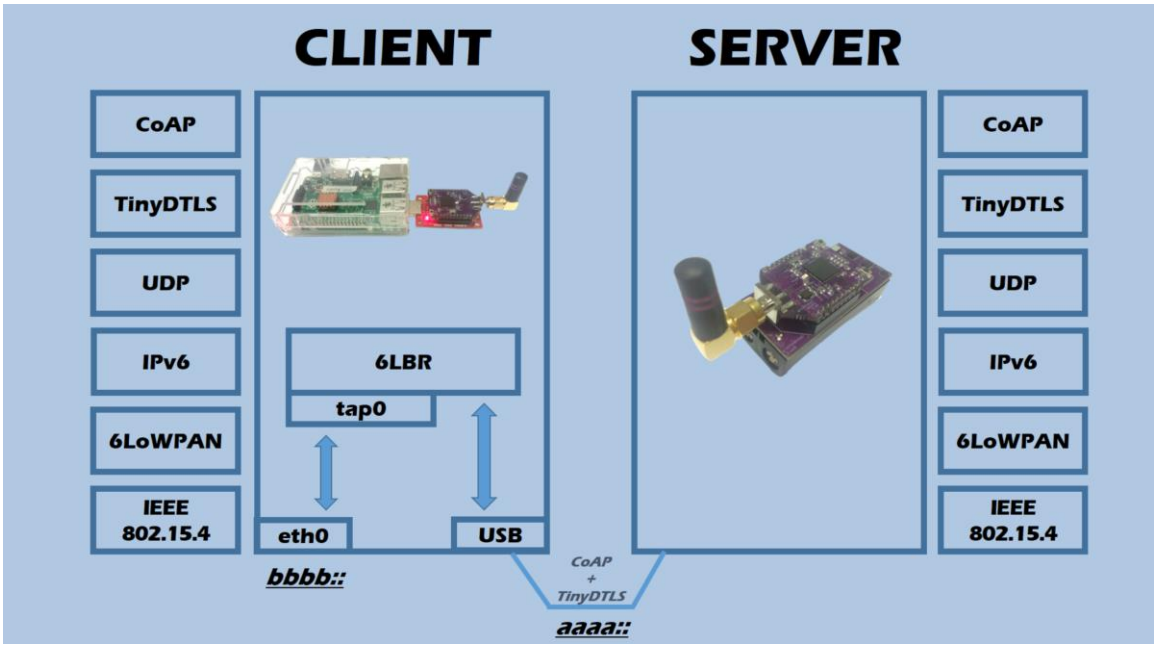
**4) 위성일:**

libcoap + tinydtls 병합
Wireshark를 통한 libcoap + tinydtls 병합 코드 자체 평가
Californium/Scandium 보편성 테스트
라즈베리파이에 6lbr 설치

라즈베리파이 환경설정
OpenMote, OpenBattery, OpenBase 세팅
JLink Debbuger, JLink GDB Server 설치 및 구동
Ultrasonic sensor 연결
CoAP resource 제작
JAVA GUI 제작
6lbr+tinydtls 디버깅 및 코드 수정

### 4.1.2 구현 환경

#### 4.1.2.1 전체 그림




**[Figure 4.1]**

라즈베리파이 위에 운영체제인 RASPBIAN을 설치하고 6LoWPAN Border Router로 동작하게 하기 위하여 Slip-radio를 퓨징한 OpenMote를 XBee dongle을 통해 라즈베리파이와 연결시킨다. Server로는 OpenMote에 CoAP과 TinyDTLS 코드를 퓨징한 후, 배터리에 연결시켜 사용한다. Client로는 Californium/Scandium에서 제공하는 client program을 이용한다. 6lbr로 동작하는 라즈베리파이는 aaaa:: 네트워크를 통하여 server와 CoAP 통신을 할 수 있다. 6lbr은 자신의 네트워크 영역에 있는 모든 노드를 인식하여, 6lbr sensor page에 나타낸다.

통신순서는 먼저, client program에서 server인 OpenMote로 resource를 요청한다. [Figure 4.1]에 나타난 것과 같이 CoAP부터 하위 계층으로 encapsulation을 하여 암호화된 메시지를 생성한다. 그리고 aaaa:: 네트워크로 패킷을 전송한다. Server는 패킷을 수신한 후 decapsulation하여 복호화 된 메시지를 읽고, 요구한 resource를 반환한다.

#### 4.1.2.2 OpenMote

OpenMote 쪽에 필요한 장비는 OpenMote와 OpenBase, OpenBattery, J Link Debugger, usb 케이블, XBee Explorer Dongle이다. OpenMote에는 CC2538이라는 SoC가 부착 돼있고 OpenMote에는 각종 interface와 라디오 송수신기, LED, BUTTON 등이 있다. OpenBase는 OpenMote 위에 소스 코드를 퓨징할 수 있도록 해준다. OpenBattery는 OpenMote에 power를 제공하며 온/습도, 조도, 기울기 센서를 제공한다. XBee Explorer Dongle은 OpenMote와 라즈베리파이를 연결하여 라우터로 동작할 수 있게 해준다. 라우터로 설정을 한 후에, 라우팅 테이블에 서버와 통신할 경로를 스태틱하게 설정하여 준다. 또한 6lbr이 정상적으로 동작하게 하기 위하여, 채널은 25로 설정하고 PAN ID를 0xABCD, MAC protocol로는 nullmac driver, RDC protocol로는 contikimac driver를 사용한다. 그리고 router로 동작할 OpenMote에는 slip-radio 소스 코드를, server로 동작할 OpenMote에는 6lbr-demo 소스 코드를 CoAP과 DTLS 옵션을 주어 퓨징한다.

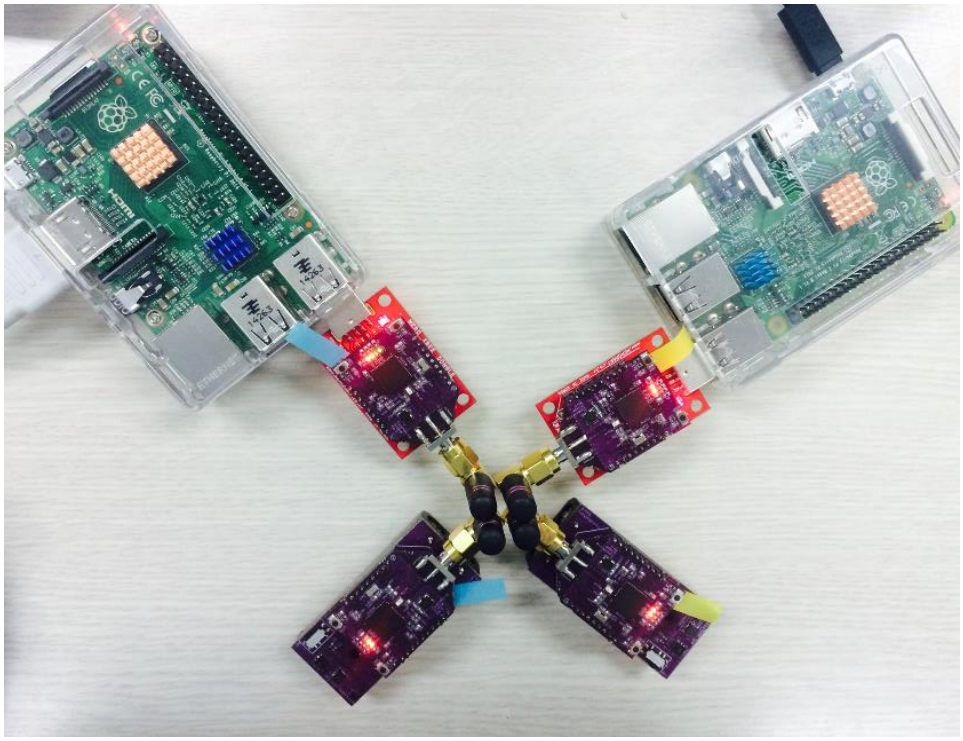
#### 4.1.2.3 Raspberry Pi

4.1.2.1과 4.1.2.2에서 OpenMote에 구현하려고 하였던 계획이 TinyDTLS handshake 과정의 코드 상의 문제로 완료되지 못하였다. 따라서 본 팀에서는 라즈베리파이 상에서 loopback 통신을 하도록 계획을 수정하였다. 라즈베리파이에 ultrasonic sensor를 GPIO에 연결하여 데이터를 전송하도록 하였고, client와 server를 같은 host 상에 구현하였다. 이 때 사용한 코드는 libcoap과 tinydtls이며, 본 팀에서 자체적으로 병합한 코드를 사용하였다. 또한 ultrasonic에서 측정하는 distance를 resource로 제작하였으며, 이 resource를 handling할 수 있는 GET, PUT, POST, DELETE method도 구현하였다. 시나리오는 다음과 같다. Client가 server에게 distance resource를 요청하면, server와의 handshake 과정 이후에 5초 간격으로 계속하여 server에게 요청을 하게 된다. Server는 암호화된 distance value를 요청을 받는 5초 마다 즉각 response 해준다. Client 상에서 만약 distance value가 20cm 이내라면, 침입자가 발생했다는 경고창을 띄워준다.

### 4.1.3 구현 결과

#### 4.1.3.1 최종 구현 결과물

##### 4.1.3.1.1 OpenMote-CC2538 / Raspberry pi



**[Figure 4.2]**

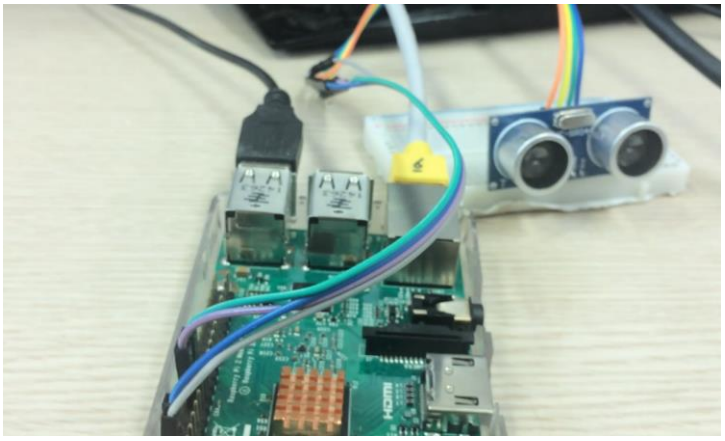
OpenMote-CC2538에는 Slip-radio 코드를 퓨징하여 XBee dongle을 통해 raspberry pi에 연결하여 router로 동작하도록 한다. 또 다른 OpenMote에는 6lbr-demo의 옵션을 CoAP server, TinyDTLS 로 설정하여 CoAP-TinyDTLS 통신 server로 사용한다. Client는 raspberry pi 상에 Californium/Scandium client program을 설치하여 사용한다.

Server로 동작하는 OpenMote를 Raspberry pi 상의 6 LoWPAN Border Router가 노드를 인식하고 ping 통신을 정상적으로 수행함을 확인하였다. 그러나 Client와의 CoAP-TinyDTLS 통신 과정 중 일부분만 진행되었다. TinyDTLS의 Handshake 과정 중에서 ClientHello → ServerHelloVerifyRequest → ClientHelloWithCookie 과정까지 정상적으로 진행되었으나, server가 ServerHello를 보내는 과정부터 진행되지 않았다. TinyDTLS 코드 상 security parameter initialization 과정에서 문제가 있어서, 코드를 수정하여 메시지를 생성하는 것까지 완료하였으나, server가 ServerHello 메시지를 전송하였음에도 불구하고 client에서 수신하지 못하였다. 이를 6lbr-dev mailing service에 문의하고, 6lbr 개발자에게 답변도 받아서 apps/tinydtls/platform-specific/platform.h에 있는 소스 코드인 #define DTLS_PEERS_NOHASH 1을 주석처리 하면 정상적으로 동작한다고 하였으나 이를 처리하니 소스코드가 빌드 되지 않았다. 빌드 되지 않은 이유는 uthash.h 코드를 컴파일 하기 위해서는 gcc version이 높아야 했는데, 본 팀이 가지고 있는 gcc compiler의 버전이 낮아서 컴파일이 되지 않았다. 또한 6lbr의 TinyDTLS 코드 부분이 캡스톤 프로젝트 완료 5일 전에 수정되는 등 bug fix 과정이 아직 완료되지 않아서 본 팀이 파악하지 못한 여러 버그가 있었을 것이라 예상된다. 또한 초기에 디버깅 환경을 구축하는 부분이 미흡했던 점도 있다. 따라서 본 팀이 병합한 libcoap-tinydtls 코드를 raspberry pi 상에 구축하여 simulation 할 수 있도록 program을 구현하였다.

#### 4.1.3.1.2 Raspberry pi 2 B+ / Ultrasonic sensor

Raspberry pi 2 B+ 에 libcoap+tinydtls 병합 코드를 컴파일하고, server와 client가 CoAP+TinyDTLS loopback 통신을 하도록 결정하였다. 이를 확인하기 위한 simulation program으로는 server가 ultrasonic sensor resource를 가지고 있고, client가 이를 요청하여 그 결과에 따라 침입자가 있는 지 없는 지 경고창을 띄워 알려주도록 구현하였다.

[Figure 4.3]은 라즈베리파이에 ultrasonic sensor를 ECHO를 GPIO PIN4, TRIG를 GPIO PIN5에, GND를 GND, VCC를 VCC에 연결한 것이다.



**[Figure 4.3]**

프로그램을 실행하기 위한 과정은 아래와 같다.

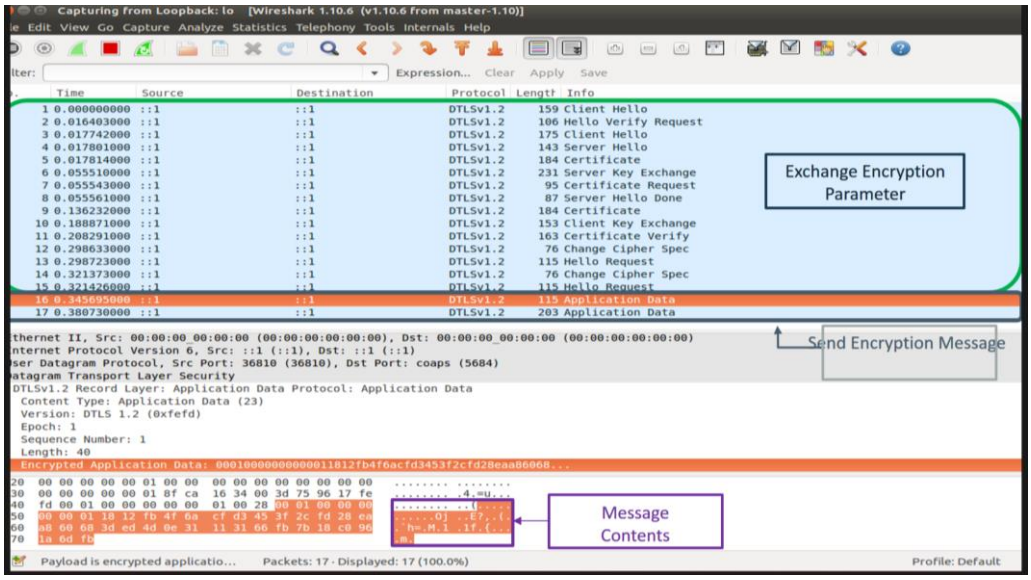
먼저 아래 명령어를 사용하여 server를 켠다.

```
./sudo server
```

그리고 client가 server에게 ultrasonic distance resoure를 다음과 같은 명령어를 사용하여 GET 한다. 의미는 GET method를 사용하여 CoAP loopback, tinydtls port 번호인 20220 번으로 distance resource를 요청한다는 것이다.

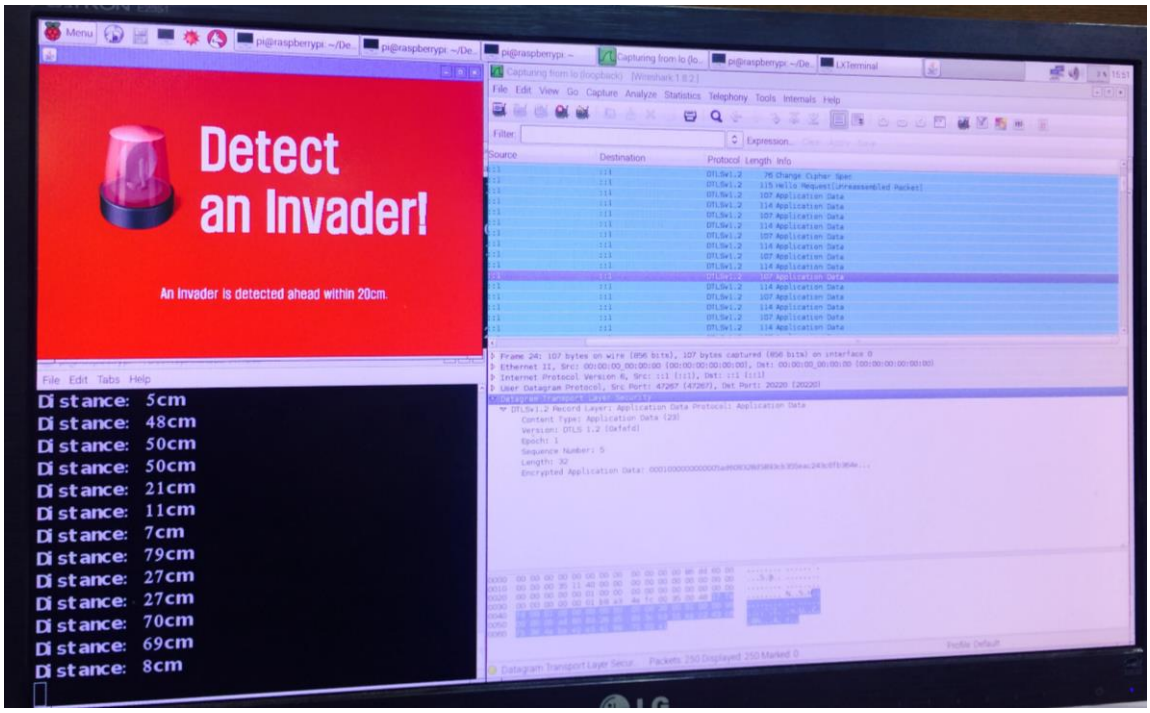
```
./client -m get coaps://[::1]:20220/distance
```

그러면 server와 client 간의 [Figure 4.4]와 같은 Handshake 과정이 일어난다.

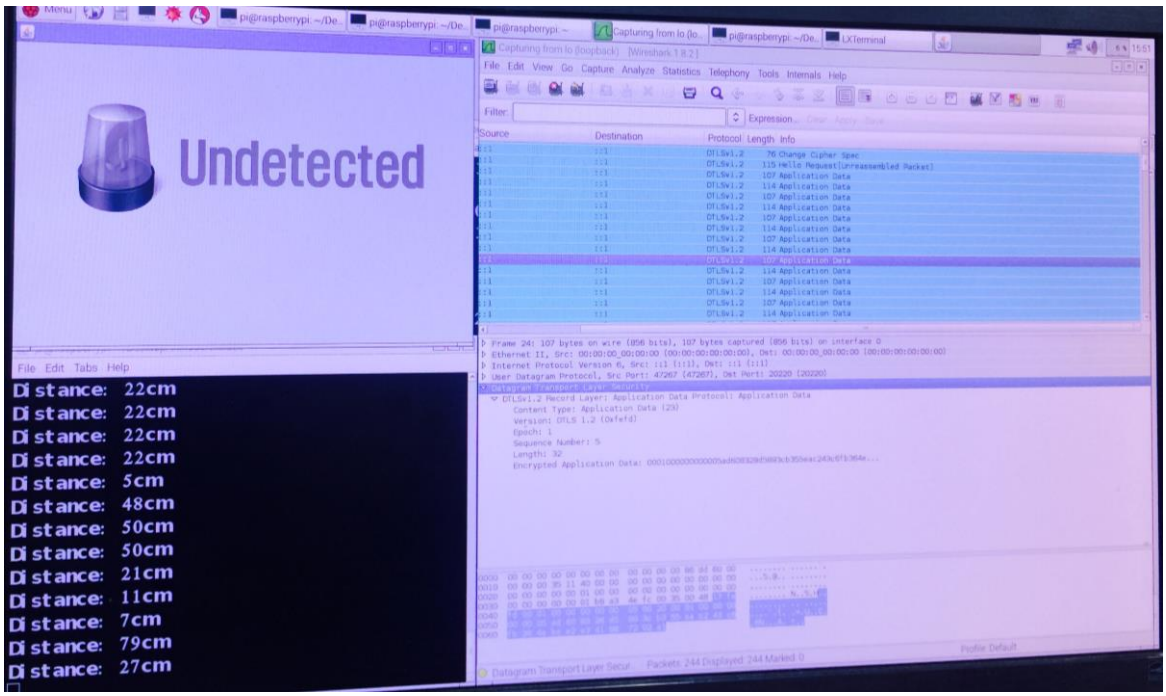



**[Figure 4.4]**

Handshake 과정 이후에 server와 client간 5초 간격으로 암호화된 distance value를 주고 받게 된다. 아래 [Figure 4.5]는 client가 받은 distance resource 값이 20 cm 이상일 때의 화면이고, [Figure 4.6]은 값이 20 cm 미만일 때의 화면이다. Wireshark 메시지에서는 application data가 암호화되어 어떤 값인지 알아볼 수 없다.




**[Figure 4.5]**




**[Figure 4.6]**

### 4.2 Experimental Setup

본 프로젝트에서 사용한 오픈 소스 버전은 아래와 같다.

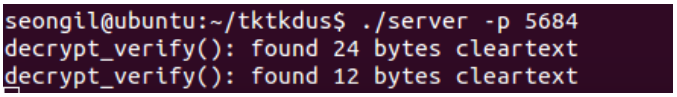
1) tinydtls - 0.8.2 version
2) libcoap - 4.1.1 version
3) 6lbr : (committed on Apr 28)-laurentderu
4) tinydtls : (committed on Mar 8 )-laurentderu
5) californium: (committed on Nov 20. 2015)-mkovatsc

### 4.3 Evaluation (설계평가)

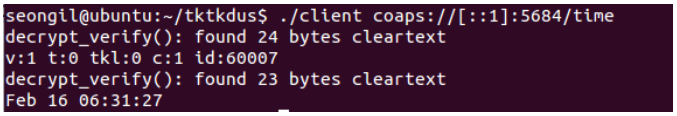
#### 4.3.1 성능 테스트

##### 4.3.3.1 자체 코드

본 프로젝트에서는 libcoap과 tinydtls 라이브러리를 합친 코드에 대한 테스트를 위하여 직접 만든 서버와 클라이언트에 대하여 통신을 시도하였다.

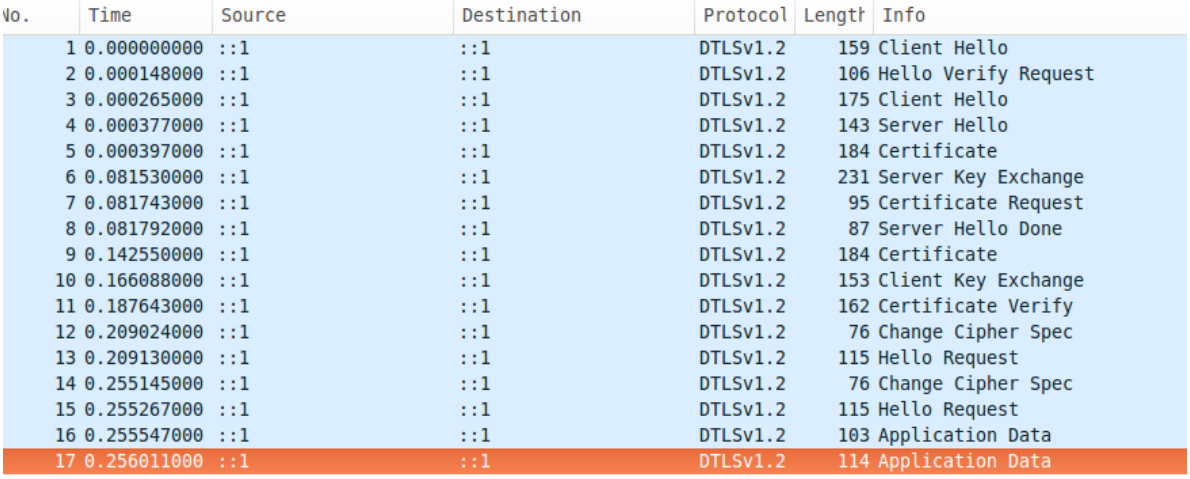



**[Figure 4.6] CoAP-TinyDTLS server side**




**[Figure 4.7] CoAP-TinyDTLS client side**

[Figure 4.6], [Figure 4.7]은 libcoap과 tinydtls를 합친 클라이언트와 서버 실행파일을 각각 실행함으로써 서버에 있는 time resource를 클라이언트에서 GET method를 통해 얻어오는 과정을 보여주고 있다. 실행 결과 CON, ACK 메시지가 정상적으로 암호화, 복호화 되어 response가 오는 것을 볼 수 있다.



| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 1 | 0.000000000 | ::1 | ::1 | DTLSv1.2 | 159 | Client Hello |
| 2 | 0.000148000 | ::1 | ::1 | DTLSv1.2 | 106 | Hello Verify Request |
| 3 | 0.000265000 | ::1 | ::1 | DTLSv1.2 | 175 | Client Hello |
| 4 | 0.000377000 | ::1 | ::1 | DTLSv1.2 | 143 | Server Hello |
| 5 | 0.000397000 | ::1 | ::1 | DTLSv1.2 | 184 | Certificate |
| 6 | 0.081530000 | ::1 | ::1 | DTLSv1.2 | 231 | Server Key Exchange |
| 7 | 0.081743000 | ::1 | ::1 | DTLSv1.2 | 95 | Certificate Request |
| 8 | 0.081792000 | ::1 | ::1 | DTLSv1.2 | 87 | Server Hello Done |
| 9 | 0.142550000 | ::1 | ::1 | DTLSv1.2 | 184 | Certificate |
| 10 | 0.166088000 | ::1 | ::1 | DTLSv1.2 | 153 | Client Key Exchange |
| 11 | 0.187643000 | ::1 | ::1 | DTLSv1.2 | 162 | Certificate Verify |
| 12 | 0.209024000 | ::1 | ::1 | DTLSv1.2 | 76 | Change Cipher Spec |
| 13 | 0.209130000 | ::1 | ::1 | DTLSv1.2 | 115 | Hello Request |
| 14 | 0.255145000 | ::1 | ::1 | DTLSv1.2 | 76 | Change Cipher Spec |
| 15 | 0.255267000 | ::1 | ::1 | DTLSv1.2 | 115 | Hello Request |
| 16 | 0.255547000 | ::1 | ::1 | DTLSv1.2 | 103 | Application Data |
| 17 | 0.256011000 | ::1 | ::1 | DTLSv1.2 | 114 | Application Data |

**[Figure 4.8] Wireshark Capture**

[Figure 4.8]는 [Figure 4.6]와 [Figure 4.7]의 통신에 대한 패킷 캡쳐 사진이다. 정상적인 handshake 과정이 끝나고 Application Data로 복호화된 CON, ACK의 CoAP 메시지가 전송됨을 확인 할 수 있다.

#### 4.3.3.2 보편성 테스트

본 프로젝트에서 만든 소스코드에 대한 보편성 테스트를 위하여 공식적으로 CoAP을 자바 소스코드로 구현한 Californium과 이에 대한 sub-project로써 DTLS 1.2가 구현된 Scandium을 서버로 이용한다. TinyDTLS와의 통신이기에 오직 TLS_PSK_WITH_AES_128_CM_8 ciphersuite 만이 Scandium과 상호 운용이 가능하다. 본 테스트에서는 californium에서 scandium 라이브러리를 이용하는 cf-secure의 SecureServer 클래스를 server로 사용한다.

```
seongil@ubuntu:~/tktkdus$ ./client coaps://[::1]:5684/.well-known/core
decrypt_verify(): found 24 bytes cleartext
v:1 t:0 tkl:0 c:1 id:13393
decrypt_verify(): found 36 bytes cleartext
</.well-known/core>,</secure>
```

**[Figure 4.9] libcoap + tinydtls client side**

```
12 FINE [ServerHandshaker]: DTLS Message processed (/0:0:0:0:0:0:0:1:49974):
==[ DTLS Message  ]=========================================
Content Type: Handshake (22)
Version: 254, 253
Epoch: 1
Sequence Number: 0
Length: 40
        Handshake Protocol
        Type: Finished (20)
        Message Sequence: 3
        Fragment Offset: 0
        Fragment Length: 12
        Length: 12
                Verify Data: C1 2F 40 35 E7 BA 3B 45 E7 0D 74 24
============================================================ - (ch.ethz.inf.vs.scan....
------------------------------------------------------------
/0:0:0:0:0:0:0:1:49974 ==> req CON-GET    MID=13393, Token=[], OptionSet=[URI-Port=5684, URI-Path=[.well-known, core]],
------------------------------------------------------------
------------------------------------------------------------
/0:0:0:0:0:0:0:1:49974 <== res ACK-2.05   MID=13393, Token=[], OptionSet=[Content-Format=40], "</.well-known/core>,"...
------------------------------------------------------------
11 FINE [DTLSConnector]: Sending message to /0:0:0:0:0:0:0:1:49974 - (ch.ethz.inf.vs.scandium.DTLSConnector.java:.....
```

**[Figure 4.10] Scandium cf-secure server side**

테스트 결과 client에서 well-known core로 request를 했을 때 적절한 response로 resource의 list가 전송되는 것을 확인할 수 있다.

### 4.3.2 성능 평가

#### 4.3.2.1 Objective evaluation

| 목표 | 평가 |
|---|---|
| IoT 표준 프로토콜 CoAP과 사물인터넷 보안에 특화된 프로토콜인 TinyDTLS를 결합 | O |
| OpenMote-CC2538에 slip-radio를 퓨징하여 raspberry pi에 XBee dongle을 통해 연결한 뒤 라우터로 사용 | O |
| OpenMote-CC2538에 CoAP + TinyDTLS를 퓨징하여 server로 사용 | X |
| 라즈베리 파이에 CoAP+TinyDTLS를 사용한 simulation program 구현 | O |
| 라즈베리 파이에 통신 결과를 보여주는 GUI 구현 | O |

#### 4.3.2.2 Constraint evaluation

| 제한조건 | 내용 | 평가 |
|---|---|---|
| 제작비용 및 기간 | 비용은 150만원 이내로 한다.<br>기간은 2015.07.01부터 2016.06.04 로 한다. | O |
| 사회, 문화, 윤리 | 외부 해킹에 의한 보안 취약점과 별개로 사물인터넷에서 실시간 수집되는 정보가 사생활을 침해하지 않아야 한다. | O |
| 안전, 보건, 환경 | 사물인터넷 통신을 하는 데에 데이터의 기밀성, 무결성, 가용성을 보장해야 한다. | O |
| 산업표준 | IETF에 지정한 표준 프로토콜인 CoAP의 document인 RFC 7252를 어기지 않아야 한다.<br>UDP 보안 프로토콜인 DTLS의 document인 RFC 6347을 어기지 않아야 한다. | O |
| 기타 (미학적 요인 등) | 센서로부터 받아오는 데이터는 직관적으로 나타내야 한다. | O |

## 5. CONCLUSION

본 프로젝트의 목적은 M2M 통신에서 사용되는 표준 프로토콜인 CoAP과 그 통신에서의 보안 프로토콜인 Tiny-DTLS를 결합하는 것이다. 또한, 사물인터넷에 적합한 hardware인 OpenMote-CC2538을 client 및 server로 사용하여 암호화된 통신을 가능하도록 하며, 그에 대한 결과를 확인할 수 있는 Simulation program을 구현한다

libcoap과 tinydtls 오픈소스를 병합한 소스 코드를 Raspberry pi 상에서 컴파일한다. 그리고 server 및 client를 구동하여 CoAP + TinyDTLS 통신을 하도록 하였다. 시뮬레이션 프로그램은 server가 ultrasonic sensor 값을 resource로 가지고 있고, 이를 client가 요청한 뒤 서로 DTLS handshake 과정을 통해 암호화 파라미터 등을 결정하여 보안이 적용된 메시지를 주고받고, 그 결과를 요청한 client가 확인할 수 있도록 구현하였다.

libcoap과 tinydtls 병합 코드의 보편성을 평가하기 위하여 본 팀은 2가지의 테스트를 완료하였다. 먼저, 자체적으로 구현한 client와 server를 실행하여 loopback 통신을 하도록 한다. 그 결과를 Wireshark로 캡처하여 정상적으로 DTLS의 handshake 과정 및 암호화 과정이 진행되었는지 확인한다. 테스트 결과, handshake 과정이 정상적으로 일어났으며 그 이후의 메시지는 암호화되어 전송되는 것을 확인할 수 있었다. 둘째, CoAP-18과 DTLS 1.2가 구현되어 있는 Californium/Scandium library의 server를 사용하여 본 프로젝트에서 구현한 client와 정

상적으로 통신이 진행되는지 확인하였다. Wireshark로 캡처한 결과 DTLS, CoAP 통신이 정상적으로 일어났음을 확인하였다.

본 프로젝트의 의의는 사람과 사람, 사람과 사물들을 인터넷에 연결하여 초-연결 사회를 구축할 수 있는 기반인 사물인터넷(IoT: Internet of Things) 기술의 활성화 및 서비스 창출을 위하여 해결하여야 하는 한가지 문제인 보안 솔루션을 연구 및 적용해 보는 데에 있다. 본 프로젝트에서 제안한 솔루션은 사물인터넷 노드에 적합한 DTLS 프로토콜인 TinyDTLS와 사물인터넷 표준 프로토콜인 CoAP을 결합함으로써 end device의 기밀성, 무결성, 가용성을 지원하는 것이다. 사물인터넷 표준 프로토콜인 CoAP(Constraint Application Protocol)에 저전력 노드에 적합한 보안 기술인 TinyDTLS를 제공함으로써 더 안전한 통신을 할 수 있을 것이라 기대한다.

본 과제에서 완료하지 못했던 부분인 DTLS handshake 과정의 오류를 수정하여 OpenMote server와 raspberry pi상의 client가 정상적인 CoAP + TinyDTLS 통신을 완료시킬 필요가 있다. 그리고 OpenMote server의 sensor resource를 OpenBattery에 있는 센서 등으로 설정하여 통신에 사용하도록 한다. 또한 TinyDTLS의 취약점을 연구하여 보완할 수 있도록 한다. 그리고 libcoap + tinydtls 병합 코드를 open source 개발자 경진대회에 게시하여 적절한 코드임을 인증하도록 한다.

## 6. REFERENCES

### 6.1 기술표준문서, 특허 및 참고논문


6.1.1 RFC 7252 – The Constrained Application Protocol, Z.Shelby, K.Hartke, C.Bormann, IETF, June 2014.

6.1.2 RFC 6347 – Datagram Transport Layer Security Version 1.2, E.Rescorla, N.Modadugu, IETF, January 2012.

6.1.3 draft-ietf-core-block-17 – Block-wise transfers in CoAP, C.Bormann, Z.Shelby, CoRE Working Group, March 2015.

6.1.4 draft-ietf-core-observe-16 – Observing Resources in CoAP, K.Hartke, CoRE Working Group, December 2014.

6.1.5 IETF CoAP 기반 센서 접속 프로토콜 기술 동향, 고석갑, 박일균, 손승철, 이병탁, 한국전자통신연구원, 2013.

6.1.6 IoT in Five days, Antonio Linan Colina, Alvaro Vives, Marco Zennaro, Antoine Bagula, Ermanno Pietrosemoli, ICTP, March 2015.

6.1.7 Lightweight DTLS implementation in CoAP-based Internet of Things, Vishwas Lakkundi, Keval Singh, ADCOM, September 2014.

6.1.8 CC2538 Powerful Wireless Microcontroller System-On-Chip for 2.4-GHz IEEE 802.15.4, 6LoWPAN, and ZigBee Applications, Texas Instruments, April 2015.

6.1.9 How to port openwsn - prof. jong won Lee, HGU, December 2015.


### 6.2 도서


6.2.1 Ad Hoc Networks - Natalie Mitton, Melike Erol Kantarci, Antoine Gallais, Symeon Papavassiliou - Springer - September 1-2, 2015.

6.2.2 Cryptography and network security : principles and practice - Stallings, William - Upper Saddle River, N.J. : Prentice Hall – 2006.

**6.3 웹사이트 및 기타**

6.3.1 Raspberry Pi OS https://www.raspberrypi.org/downloads/

6.3.2 Scandium https://github.com/eclipse/californium.scandium

6.3.3 Californium https://github.com/eclipse/californium

6.3.4 Libcoap https://sourceforge.net/projects/libcoap/

6.3.5 6lbr source code https://github.com/cetic/6lbr

6.3.6 6lbr 설치 방법 및 모드 설정과 사용 방법 https://github.com/cetic/6lbr/wiki/

6.3.7 6lbr 설치 방법 http://processors.wiki.ti.com/index.php/Cc26xx_sw_examples

6.3.8 tinydtls source code in 6lbr https://github.com/cetic/tinydtls

6.3.9 tinydtls source code https://sourceforge.net/projects/tinydtls/

6.3.10 openmote build and putty configuration http://www.openmote.com/blog/getting-started-with-contiki-and-openmote.html

6.3.11 CoAP과 TinyDTLS build http://sunmaysky.blogspot.kr/2015/11/how-to-build-dtls-example-for.html

Appendix :

# Part A 공동작성

## A.1 How to Use It

### A.1.1 User

본 팀은 사용자를 1.3에서 명시했던 Key Project Stakeholders를 대상으로 한다.

1. 사물인터넷 물품을 판매하는 업체
2. 사물인터넷 보안 솔루션 개발 업체
3. 사물인터넷 보안 연구를 목표로 하는 학부생 혹은 대학원생

### A.1.2 How to use

본 프로젝트에서 사용하는 Hardware는 OpenMote CC2538과 raspberry Pi 2 B+ 이다. 각 hardware에 따라 사용법을 설명하면 아래와 같다.

1. **OpenMote CC2538**
   - ① OpenMote에 source code를 퓨징하기 위해서는 JLink Debbuger를 OpenMote, OpenBase에 연결한 후, source code가 저장된 컴퓨터와 연결해야 한다.
   - ② JLink GDB Server
     JLink에 관련된 설치 파일을 다운로드 받은 후 다음과 같은 명령어를 통해 JDB server를 켠다.
     ```
     /opt/Segger/JLink/JLinkGDBServer -device CC2538SF53
     ```
   - ③ 퓨징할 코드를 Cross compile하여 실행파일로 만든다.
   - ④ 실행파일을 OpenMote에 올리기 위하여 아래와 같은 명령어를 통해 퓨징한다.
     ```
     arm-none-eabi-gdb
     ```
     ```
     target remote localhost:2331
     monitor interface jtag
     monitor speed 5000
     monitor endian little
     monitor flash download = 1
     monitor flash breakpoints = 1
     monitor reset
     load hello-world.elf
     ```
   - ⑤ 본 프로젝트에서는 slip-radio와 6lbr-demo를 퓨징하였다. Slip-radio의 경우 아래 driver를 설정해 주었다.
     MAC: nullmac_driver
     RDC: contikimac_driver

## 2. Raspberry Pi

### 2-1. OpenMote와의 통신

① 6lbr로 구동 시키기 위하여 OpenMote에 slip-radio 코드를 퓨징하여 XBee dongle로 라즈베리파이에 연결한다.

② 다음과 같은 명령어를 사용하여 6lbr을 활성화 시킨다.

```
sudo service 6lbr start
```

③ Routing table에 센서 노드와 통신할 수 있도록 경로를 설정한다.

```
sudo route -A inet6 add aaaa::://64 gw bbbb::100
```

④ 6lbr web page에 접속한 후, detect된 센서 노드의 IP주소를 사용해 센서 노드로 resource를 요청한다.

```
./client -m get coaps://[IPv6_address]:5684/.well-known/core
```

### 2-2. Raspberry Pi loopback 통신

① 다음 명령어를 사용하여 server를 구동한다. 이 때 server가 가지고 있는 resource는 ultrasonic sensor로 감지한 distance value이다.

```
./sudo server
```

② Client가 server에게 distance resource를 아래와 같은 명령어를 사용하여 요청한다.

```
./client -m get coaps://[::1]:20220/distance
```

③ Server가 측정한 값을 client가 확인하기 위해 JAVA API를 구동 시킨다.

```
java Test.java
```


④ Server와 client간 암호화된 통신을 확인하려면 Wireshark를 사용할 수 있다.

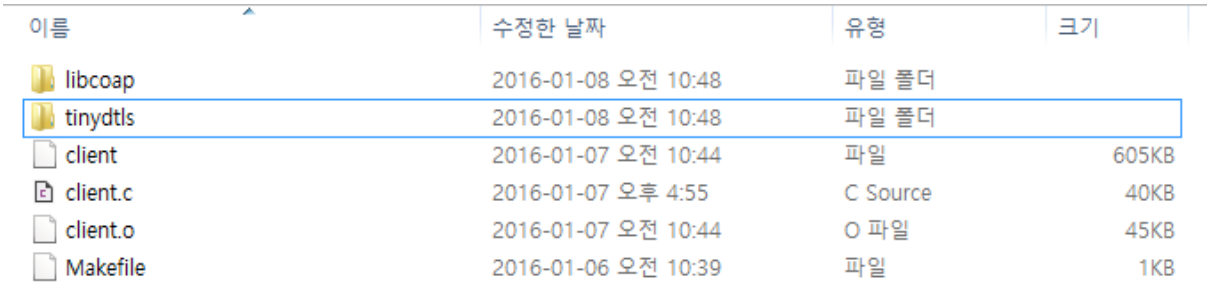
```
sudo wireshark
```

⑤ 만약 client가 받은 distance value가 20cm 미만이라면 경고창을 보여준다.

**A.2 상세설계 및 소스코드 블록도 설명**

**소프트웨어의 경우 소스코드 파일 구조를 포함한 목록과 설명, 소프트웨어 공학적 소스코드 설명 문서 등.**

A.2.1 기본적인 디렉토리 구조



| 이름 | 수정한 날짜 | 유형 | 크기 |
|---|---|---|---|
| libcoap | 2016-01-08 오전 10:48 | 파일 폴더 | |
| tinydtls | 2016-01-08 오전 10:48 | 파일 폴더 | |
| client | 2016-01-07 오전 10:44 | 파일 | 605KB |
| client.c | 2016-01-07 오후 4:55 | C Source | 40KB |
| client.o | 2016-01-07 오전 10:44 | O 파일 | 45KB |
| Makefile | 2016-01-06 오전 10:39 | 파일 | 1KB |

기존의 libcoap과 tinydtls의 make 방법은 유지

Client.c에 include 시켜 client.c에 대한 make 파일만 새로 생성.

A.2.2 dtls-client.c 파일 소스코드 블록도

| 변수 선언 및 초기화 부분 |
|---|
| 옵션 처리 부분 ( i : psk_id설정, k: psk_key 설정, p : serve의 port 번호 설정 v : log_level 설정) |
| 목적지에 대한 ip설정 및 port번호 설정, socket 생성 (UDP , IPv4, v6 모두 지원) |
| 만든 socket과 dtls을 이용한 통신을 할 상대와의 세션 정보들 (cookie, peers, sendqueue, socket #, retransmit_timer, dtls_handler)을 생성하여 dtls_context_t 구조체에서 관리 |
| dst에 저장된 상대에게 client_hello를 먼저 보내고 상대로부터 받은 response의 프로토콜 타입, 메시지 종류에 따라 다르게 메시지를 만들어 전송함. (무한 루프) client:close를 입력할 경우 프로세스 종료, client:renegotiate을 입력할 경우, 다시 handshake를 진행하여 새로운 암호화 parameter들을 생성. |

A.2.3 dtls-server.c 파일 소스코드 블록도

| 변수 선언 및 초기화 부분 |
| --- |
| 옵션 처리 부분 ( A: listen address설정 , p : serve의 port 번호 설정 v : log_level 설정) |
| Listen port번호 설정, socket 생성 (UDP , IPv4, v6 모두 지원) |
| 만든 socket과 dtls을 이용한 통신을 할 상대와의 세션 정보들 (cookie, peers, sendqueue, socket #, retransmit_timer, dtls_handler)을 생성하여 dtls_context_t 구조체에서 관리 |
| dtls_handle_read함수를 통하여 listen. Handshake / Change Cipher Spec / Alert / Application Data 가 올 경우 그에 대하여 read후 response(필요한 경우만) ->(무한루프) |

A.2.4 libcoap client.c 파일 소스코드 블록도

| 변수 선언 및 초기화 부분 |
| --- |
| 옵션 처리 부분 (f : 파일로부터 data 수신, g : group 설정, p: 자신에 대한 port 설정, m: method 설정, N : 메시지 타입을 NON으로 설정, s: 해당 자원에 대해 observe 설정. A: accept option 설정, t: content type 설정 P : 프록시 주소 설정, T: 토큰 설정) |
| 목적지에 대한 ip설정 및 port번호 설정, socket 생성 (UDP , IPv4, v6 모두 지원) |
| 만든 socket과 통신하는 remote에 대해서 retransmission timer, sendqueue, recvqueue, sendqueue_base time, message id 등 메시지 전송에 있어서 필요한 정보들을 관리하는 coap_context_t를 생성. |
| CoAP 메시지를 coap_new_request 함수를 통해 생성한 다음, 각 메시지 타입에 따라 다른 방법으로 위에서 만든 메시지를 보내고, 받을 때에도 각 메시지 타입에 따라 다른 방법으로 처리한다. |

A.2.5 libcoap server.c 파일 소스코드 블록도

| 변수 선언 및 초기화 부분 |
| --- |
| 옵션 처리 부분 (p: 자신에 대한 port 설정, A: interface address 설정, t: content type 설정 P : 프록시 주소 설정, v: log level 설정) |
| listen port번호 설정, socket 생성 (UDP , IPv4, v6 모두 지원) |
| 해당 socket과 retransmission timer, sendqueue, recvqueue, sendqueue_base time, message id 등 메시지 전송에 있어서 필요한 정보들을 관리하는 coap_context_t를 생성. |
| CoAP메시지가 오기 까지 listen이후 CoAP message 포멧에 맞추어 parsing 그 이후 해당 메시지에 따른 response를 생성, 전달. |

A.2.6 tinydtls 파일 main의 주요 함수 분석

① resolve_address(const char *server, struct sockaddr *dst) 함수

사용자가 입력한 IP 주소와 SOCK_DGRAM, AF_UNSPEC hint를 참고하여 getaddrinfo함수로부터 주소 정보를 얻어와 dst에 저장함.

② dtls_new_context (void* app_data) 함수

App_data에는 생성한 socke의 fd가 전달되는데, 이 fd와 관련하여 이 소켓과 세션을 생성한 peer에 대한 context (메시지를 주고 받는데 필요한 변수들)구조체를 생성하고 변수(시간, sendqueue,peers,cookie) 값들을 초기화 하는 함수.

③ dtls_connect (dtls_context_t *ctx, const session_t *dst) 함수

사용자가 입력한 ip주소가 현재 dtls_context (세션(?))의 peer에 있는지 확인하고 없으면, peer의 목록에 넣어준다. 그리고 해당 peer에 client hello를 전송함으로써 연결을 한다.

④ dtls_handle_read (struct dtls_context_t *ctx) 함수

파일 디스크립터로 부터 받은 메시지에 대해 프로토콜 종류와 메시지 타입에 따라서 적절한 행동을 취하게 하고 (handshake의 경우에는 다음 메시지에 대한 전송까지 한다.), 자신의 state와 role을 수정한다.

⑤ try_send 함수와 dtls_write 함수

사용자로부터 입력 받은 app data를 record protocol을 이용하여 record를 만들어 전송한다. 또한, 현재 연결된 peer가 없는 경우, handshake 과정을 진행한다.

A.2.7 libcoap 파일 main의 주요 함수 분석

① pdu = coap_new_request(ctx, method, optlist)

Coap request message를 만들어 pdu에 반환해 주는 함수. pdu에 받아진 메시지를 이후 coap_send_confirmed 혹은 coap_send 함수를 통해 보내짐. 그러므로 이 함수 안에 sendto 함수 부분을 찾아 pdu의 내용을 암호화하여 보내면 될 것으로 생각됨.

② coap_send_confirmed(ctx,&dst, pdu), coap_send(ctx, &dst, pdu)

메시지 타입이 CON인 경우, coap_send_confirmed 함수를 통해 전송하고 그 외의 경우는 coap_send 함수를 통해서 전송된다.

③ coap_delete_pdu(pdu)

메시지 타입이 CON이 아니거나 tid가 올바른 값이 아니면 해당 pdu는 전송하자마자 삭제한다.

④ coap_read(ctx)

Remote로부터 recvfrom 함수를 이용하여 메시지(pdu)를 받아 coap_context_t ctx의 recvqueue에 해당 메시지를 저장한다.

⑤ coap_dispatch(ctx)

recvqueue에 받아진 메시지를 메시지 타입에 맞게 처리하여 재전송 할지 해당 프로세스를 종료할지 결정한다.

## A.3 기술적 추가설명

프로젝트를 발전시키기 위해 아래와 같은 참조할 수 있는 도서 및 사이트를 첨부합니다. 본 프로젝트를 개선하고 싶을 개발자들은 아래 자료를 통해서 배경지식을 얻을 수 있고, 나아가 실제적인 구현을 하는 데에 도움이 됩니다.

도서 참고

1) IoT in Five days, Antonio Linan Colina, Alvaro Vives, Marco Zennaro, Antoine Bagula, Ermanno Pietrosemoli, ICTP, March 2015.

사이트 참고

1) tinydtls overview
https://projects.eclipse.org/projects/iot.tinydtls
2) 6lbr wiki
https://zenodo.org/record/48524#.V1HM6Clf3IU
3) Fork of Contiki 3.0: TinyDTLS 0.8.2 integrated to Contiki (CoAPS)
https://zenodo.org/record/48524#.V1HM6Clf3IU

## A.4 본 프로젝트에서 생성된 지식재산권

생략

## Part B 개인 작성

### B.1 Relationship of Individual Project Assignment to Overall Project :

- 김연희 (1 ~ 2 페이지): (1) 전체 설계를 나눈 기준은 OpenMote와 Raspberry Pi로 크게 2가지이다.

  OpenMote의 경우, 사물인터넷 통신에 적절하게 설계된 hardware라고 판단되어 이 hardware에 사물인터넷 표준 프로토콜인 CoAP과 보안 프로토콜인 DTLS를 퓨징하기로 결정하였다. 처음에 설계할 때에는 1개의 OpenMote는 센서가 부착된 OpenBattery에 연결해서 server node로 사용하기로 하였고, 다른 OpenMote는 라즈베리파이에 연결하여 6LoWPAN Border Router 역할을 하도록 하였다. OpenMote에 코드를 퓨징하기 위하여 노트북에 CentOS 설치를 하였고, 이 리눅스 컴퓨터를 통하여 소스코드를 퓨징하였다. 그 방법은 우선 6lbr git repository에서 소스코드를 받은 후에, JLink Debbuger를 통하여 할 수 있었다. JLink Debugger의 경우, 그 관련된 설치 software를 모두 설치하고나서 정해진 명령어를 통하여 OpenBase에 연결된 OpenMote에 소스코드를 퓨징할 수 있다. 또한 한번 소스코드를 퓨징하면 OpenMote를 flash해야 다시 퓨징할 수 있기에, SmartRF06 DK를 사용하여 OpenMote 퓨징을 해야 한다. 본 팀은 2개의 OpenMote에 각각 6lbr-demo와 slip-radio 소스코드를 퓨징하였다.

  Raspberry Pi의 경우, 6lbr로 동작함과 동시에 client 역할을 하도록 Californium client program을 설치하였다. 6lbr로 동작하게 하기 위하여 6lbr 설치 프로그램을 다운받았고, sudo service 6lbr start라는 명령어를 사용하여 6lbr로 동작하게 설정할 수 있다. 또한 IPv6 설정도 해주고 네트워크 경로도 설정해주면 OpenMote server node를 인식하여 6lbr web page에서 확인할 수 있다. 확인된 server로 request를 하기 위해서는 Californium/scandium 소스코드를 다운 받은 후에 client 소스코드를 켠다. 그리고 CoAP의 ./well-known/core 경로로 요청을 하면 server가 어떤 resource를 가지고 있는지 응답을 받을 수 있다. 또한 raspberry pi 상에서 client 와 server를 본 팀이 제작한 libcoap+tinydtls 코드를 컴파일 하여서 loopback 통신도 완료하였다. 이 때에는 raspberry pi에 ultrasonic sensor를 연결하여서 resource로 사용하였다.

(2) 위와 같은 설계를 완료하기 위하여 아래와 같은 작업을 하였다.

**김연희**

| 작업 주제 | 설명 |
|---|---|
| CentOS 설치 및 개발환경 구축 | 노트북을 포맷하고 OpenMote 퓨징에 사용할 리눅스 환경의 CentOS를 다운받았다. 그리고 프로젝트 진행에 적절하게 OS 환경 설정을 하였다. |
| OpenMote, OpenBattery, OpenBase 세팅 | OpenMote에 코드 퓨징 및 flashing, 또한 putty창에서 debugging 과정을 완료하였다. 또한 OpenMote에 문제가 발생할 경우, OpenMote를 제작한 Pere Tuset, 6lbr를 제작한 Laurent Deru와 mail을 주고 받으며 문제를 해결하였다. 특히 6lbr 통신이 되지 않을 때, 채널 설정, MAC과 RDC 계층 설정, RDC check rate 등을 통신이 가능하도록 설정하였다. |
| JLink Debbuger, JLink GDB Server 설치 및 구동 | JLink Debugger를 OpenMote 퓨징에 사용하기 위하여 설치 프로그램을 다운 받고, 환경 설정을 완료하였다. 그리고 gdb server를 구동 시켜서 OpenMote 퓨징을 하였다. |
| 라즈베리파이에 6lbr 설치 | XBee dongle에 연결된 라즈베리파이가 6lbr로 동작하게 하기 위하여 6lbr 환경 설정을 해 주었고, IPv6 설정, network 경로 설정 등을 해 주었다. |
| 라즈베리파이 환경 설정 | 라즈베리파이의 해상도, 언어, 디렉토리 관리 등 환경설정을 해 주었다. |
| SmartRF06 Evaluation Board를 사용한 Flash programmer 2 사용 | Flash programmer 2 를 사용해서 OpenMote를 flash 하였다. |
| CoAP resource 제작 | CoAP server가 가지고 있는 distance resource를 등록하였다. |
| CoAP method 제작 | CoAP request를 받은 경우에 메시지를 handle 할 수 있는 GET, PUT, POST, DELETE method를 구현하였다. |
| JAVA GUI 제작 | CoAP client에서 받은 응답에 따라 JAVA GUI 창을 적절히 보여주도록 JAVA 코드를 작성하였다. |
| 6lbr+tinydtls 디버깅 | 6lbr mailing list에 등록하여 tinydtls 퓨징 문제가 있을 때, 제작자에게 의뢰하고 버그를 해결하였다. 해당 버그는 hash 함수 이후의 server 코드를 수정하여 해결하였다. |

- 김진우 (1 ~ 2 페이지): (1) 캡스톤 초기 전체 설계는 라즈베리파이와 openmote 두 부분으로 나뉜다. 각 파트에서 기본적인 개발 환경을 구성하고 나면 각 device 위에 올려질 coap+tinydtls를 위한 코드 수정 작업을 진행한다. 라즈베리파이에는 6lbr process가 동작하여 클라이언트의 역할을 수행하고 서버로부터 받은 정보를 저장한다. Openmote는 서버의 역할을 수행하고 클라이언트로부터 받은 요청에 해당하는 resource value를 서버에 다시 전송한다. 라즈베리파이에서 6lbr이 네트워크 1계층부터 4계층까지 지원해주기 때문에 libcoap과 Tinydtls0.8.2를 결합한 코드를 빌드한 후 서버로 실행하면된다. Openmote 또한 6lbr에서 지원하는 네트워크 풀 스택과 함께 er-coap 18과 tinydtls를 결합하여 최종 빌드한 후 opemote에 퓨징한다. 개발 환경 설정과 함께 가장 중요한 것은 coap과 Tinydtls의 결합이다. 따라서 공학프로젝트 기획에서는 coap과 Tinydtls의 RFC 표준 문서를 보며 팀원들과 함께 각 프로토콜이 어떻게 동작하는지 공부하였고
  openmote와 라즈베리파이가 통신하기 위한 기본 단계를 성공시켰다. 캡스톤 디자인에서는 라즈베리파이의 라우터 구동과 coap+tinydtls의 코드 수정을 진행하였다. 코드 수정에는 특별히 tinydtls의 handshake 과정이 중요하다. Handshake 과정은 네트워크에서 노드들간의 암호화 작업에 필요한 재료들을 미리 교환하는 단계이다. 따라서 이 과정이 성공적으로 실행되어야 메시지 즉 데이터가 암호화 혹은 복호화 될 수 있다.
  ① 라즈베리파이(6LBR) 환경설정
  Openmote위에 Coap과 TinyDTLS를 결합한 코드를 올리기 위해 필요한 기본적인 개발 환경을 구성하였다. 사물인터넷을 구성하기 위해 위해 TCP를 기반으로 하는 일반 PC의 IPv4가 아닌 새로운 주소체계의 IPv6 네트워크 설정이 필요하였고 우리 팀에서 사용하게 될 라즈베리파이가 ethernet과 wireless network 사이에서 라우터 역할을 해야 하기 때문에 6LBR process를 사용하기로 결정하였다. Github에서 제공하는 6LBR은 기본적으로 다양한 플랫폼을 지원하고 있지만 우리 팀이 사용하는 openmote와 함께 동작시키기 위해서는 환경설정을 미리 구성하여야 했다. 6LBR이 라즈베리 파이에 동작시키는 데에는 3가지 모드가 있다. Bridge 모드, Router 모드, Transparent Bridge 모드. 이 중 Router mode가 Ethernet side에 가상의 포트(tap0)를 생성하고 802.15.4 포트를 통해서 wireless sensor network를 위한 subnet을 구성한다. 라즈베리파이 위에 있는 6lbr을 Router 모드로 동작시키기 위해 '6lbr.conf'에는 관련된 paramete들을 set하면 된다. 이 과정이 마치고 6lbr process를

동작시키면 웹의 [bbbb:100]에 접속하여 6lbr 웹페이지 정상적으로 활성화되는 것을 확인할 수 있다. 6lbr process가 정상적으로 동작함을 확인한 후에는 라즈베리파이에 slip-radio가 퓨징된 openmote를 Xbee explore dongle을 통해 라즈베리파이에 연결한다. 또한 클라이언트로써 라즈베리파이가 무선네트환경에 있는 node에 request를 하기 위해서는 라우팅 테이블 명령을 설정하여야 한다. route -A inet6 add aaaa::/64 gw bbbb::100 는 무선 네트워크 환경에 있는 node의 IP prefix aaaa::에 메시지를 보내기 위해서는 bbbb:100을 경유하여 전송해야 한다는 의미이다.

② Openmote setting, building and fusing : openmote에 크로스컴파일된 소스코드를 퓨징하기 위해서는 관련된 개발 툴을 먼저 설치하여야 한다. 개발 툴 환경에 적합한 리눅스 OS를 먼저 pc에 설치한 후 gcc 컴파일러와 JLink를 위한 GDB Server를 설치한다. 이 둘은 openmote에 올라갈 소스 코드를 최종적으로 크로스컴파일을 하여 모트 위에 동작시키도록 하는 tool이다. Openmote에 올라갈 소스코드로 우리 팀은 contiki OS 기반의 6lbr-demo를 사용하였다. 6lbr-demo는 사물 인터넷을 위한 네트워크 풀 스택을 제공하고 있으면 coap 프로토콜이 구현되어 있다. 6lbr-demo에 있는 coap과 tinyd신를 결합한 코드를 빌드할 때, "sudo make TARGET=openmote WITH_TINYDTLS=1 WITH_COAPSERVER=1 WITH_DTLS_COAP=1" 를 리눅스 terminal에서 명령어로 준다. 즉 이 명령어는 소스코드의 옵션으로 TinyDTLS와 COAP이 같이 동작하도록 하는 명령어이다. Build를 통해 생성된 파일 '6lbr-demo.openmote'가 최종 실행파일이다. 이 파일을 openmote 위에 컴파일 실행시키기 위해서는 다음과 같은 작업이 필요하다. 다른 terminal 창을 띄운 후 /opt/Segger/JLink/ 디렉토리로 들어간 후, $JLinkGDBServer --device CC2538SF53 명령을 주어 JLink Server를 활성화시키고 연결을 마친다. 소스코드 실행 파일이 생성된 디렉토리로 들어가서 $arm-none-eabi-gdb를 통하여 GDB 창을 띄운 후, localhost 번호와 speed 등을 설정해주어야 한다. 이것은 GDB client가 GDB server에 configure하는 과정이다. 마지막에 'load' 명령을 주어 '.openmote' 실행 파일을 openmote에 퓨징하면 된다. 우리 프로젝트에서 사용할 소스코드는 Server 역할을 할 openmote를 위해 'slip-radio'를 사용하였고 Client 역할을 할 openmtoe를 위해 '6lbr-demo'를 사용하였다.

(2)

| | |
|---|---|
| 라즈베리파이 | 라즈베리파이에 6lbr 환경 설정 |

| | |
|---|---|
| openmote | Centos 설치하여 개발환경 구축 |
| | OpenMote, OpenBattery, OpenBase 세팅 |
| | JLink Debbuger, JLink GDB Server 설치 및 구동 |
| | SmartRF06 Evaluation Board를 사용한<br>Flash programmer 2 사용 |

- 한영광 (1 ~ 2 페이지): (1) 먼저 크게 전체 설계를 openmote 파트와 raspberry pi 파트로 나누었다. 그리고 raspberry pi에는 다른 openmote에 slip-radio 프로그램을 fusing 하여 xbee 동글과 연결하여 raspberry pi와 결합한다. 그리고 raspberry pi 내부에서 wsn과의 라우팅이 작동할 수 있도록 6lbr 프로그램을 설치해 준다.

  Raspberry pi 파트는 구상한 사물 인터넷 모델에서 client를 맡고 있다. Raspberry pi는 앞에서 만든 libcoap + tinydtls 라이브러리의 client 프로그램을 실행 하여 openmote에게 센서값(자원)을 여러 method로 요청하여 원하는 종류의 센서 값을 얻는다. 동시에 wireshark 프로그램을 통해 openmote와 교환되는 패킷에 대한 정보를 확인할 수 있다. 그리고 raspberry pi가 인터넷에 연결되어 있는데 이를 wsn과 연결 시켜주는 역할을 raspberry pi가 할 수 있다. 이 역할을 가능하게 하기 위해 6lbr이라는 프로그램을 raspberry pi 안에 설치하여 실행시켰다. 그러면 6lbr은 raspberry pi를 wsn의 border router 역할을 할 수 있도록 만들어 wsn과 유선 네트워크를 raspberry pi가 이을 수 있도록 한다. 또한 6lbr이 제공하는 웹 인터페이스를 통해 raspberry pi는 사용자에게 wsn과의 네트워크 상태를 확인할 수 있다. 현재 raspberry pi와 연결된 node들의 ipv6 주소, node들 간의 tree 구조, 패킷 전송 횟수 등을 사용자에게 보여준다.

  Slip-radio 파트는 raspberry pi가 openmote와의 무선 통신을 위해 필요하다. Openmote에 fusing하여 xbee dongle을 통해 raspberry pi와 연결되며 raspberry pi에서 제공하지 않는 rdc layer에 대한 기능을 제공한다. 또한 csma에 대한 configuration을 할 수 있도록 한다.

  마지막으로 openmote 파트는 구상한 사물인터넷 모델에서 server를 맡고 있다. Openmote는 6lbr에서 제공하는 libcoap server와 tinydtls가 결합된 libcoap + tinydtls server 임

베디드 코드를 퓨징한다. 이 코드에서는 app layer 아래에는 contiki os가 구현되어 있고 application layer의 프로그램에는 er-coap과 tinydtls 라이브러리를 사용하였다. Openmote는 open battery와 연결시켜 open battery에 부착되어 있는 센서들로부터 센서 값들을 얻어와 client에 coap 메시지에 담아 tiny-dtls를 통해 암호화하여 전송한다.

(2)

| | |
|---|---|
| 한영광 | Raspberry pi 파트 및 library 병합 담당<br>libcoap + tinydtls 병합<br>Wireshark를 통한 libcoap + tinydtls 병합 코드 자체 평가<br>라즈베리파이에 6lbr 환경구축<br>라즈베리파이 기본 환경설정 |

- 위성일 (1 ~ 2 페이지): (1) 먼저 OpenMote와 라즈베리파이가 통신을 하기 위하여 크게 client와 server에 대한 설계를 각각 따로 하였다.

  OpenMote의 설계는 본 프로젝트에서 진행한 CoAP + TinyDTLS의 통신이 embedded적으로 구현되어 사물인터넷(IoT)의 표준 보안 플랫폼을 제시하고자 하는데 큰 의의가 있다. 특별히 사물인터넷의 대표적인 오픈 소스에 적합하게 설계되었고, IEEE 802.15.4e, 6LoWPAN, CoAP 등의 표준 stack이 적용 가능한 hardware를 server로 삼아서 보안적 요소까지 가미하여 device적 기능을 하게 끔 하는 것을 본 프로젝트에서의 주안점으로 삼았다. OpenMote에 대하여는 code fusing, cross compile – 코드를 OpenMote에 올리는 것, flash – OpenMote안에 올려져 있는 코드를 지워버리는 것, code finding – 적합한 오픈소스 라이브러리를 찾는 것, code modification – 소스코드를 수정하는 것 등의 설계적 요소가 있다.

  라즈베리파이는 전체 설계 중에서 client의 역할을 담당한다. 값 싼 컴퓨터임에도 불구하고 리눅스os를 기반으로 많은 역할을 담당 할 수 있기에 본 프로젝트에 적합한 요소 중 하나이다. server를 대상으로 요청을 할 수 있는 메커니즘이 적용된다. 추가적으로 packet을 캡쳐하고, 분석하는 것, libcoap+tinydtls client code에 대한 수정 및 병합, 라즈베리파이 환경 설정, 라우터 환경을 위한 slip-radio, 6lbr 환경조성, scandium을 통한 테스트 등의 설계적 요소들이 있다.

  큰 설계로써 라즈베리파이와 OpenMote가 있다면, 이 가운데 server로는OpenMote이외에 test및 simulation program

을 위한 라즈베리파이상에서의 자체 제작 서버와 이에 따른 resource가 있다. 이는 본 프로젝트에서 제작한 libcoap + tinydtls client 와 루프백 통신 가능하다. Simulation server에 대하여서는 libcoap + tinydtls server code에 대한 수정 및 병합, resource인 distance의 request handler 제작, distance의 value를 위한 ultrasonic sensor 와의 연결 등이 있다.

또한 통신에 대하여 CoAP에 보안 적 요소를 가미하기 전에 CoAP자체에 대한 통신이 가능한지 판단하기 위한 설계, CoAP + TinyDTLS의 통신이 옳바르게 동작하는지 판단하기 위한 설계적 요소도 존재한다.

(2)

필자의 작업내용을 표로 정리하도록 한다.

| **이름 : 위성일** | |
|---|---|
| libcoap + tinydtls 병합 | libcoap과 tinydtls 코드를 결합하기 위하여 두 코드에 대하여 분석하고, module간 dependency를 확인하였다. 이를 위하여 code insight 툴을 이용하였다. server와 client코드를 작성하였으며, 코드의 API를 수정, 결합으로써 CoAP메시지에 보안적 요소를 가미하였다. |
| OpenMote, OpenBattery, OpenBase 세팅 | Openmote에 코드를 fusing하고 putty창을 이용 하여 debuging하는 등의 작업을 수행하였다. 특별히 slip-radio와 OpenMote상의 통신을 위하여 수정해야 할 부분을 인지하고 그에 따라 channel, rdc, mac, framer, netstack conf rdc channel check rate 등을 수정, 조정하였다. |
| Wireshark를 통한 libcoap + tinydtls 병합 코드 자체 평가 | 자체 제작 코드 (server, client)를 실행한 후 wireshark를 통하여 패킷을 캡쳐하였다. handshake과정이 정상적으로 이루어 지는지, 그 이후 CoAP message의 encrypt와 decrypt가 잘 이루어 지는지 확인하기 위하여 wireshark를 이용하였다. |
| Californium/Scandium 보편성 테스트 | Californium의 subproject인 scandium과의 통신을 위하여 californium 모듈을 설치하였다. Scandium과 자체제작 client의 통신 중에 handshake 과정 상 문제가 있음을 발견하였고, 이에 따라 client를 수정하였으며, 후에 정상적 통신이 가능하게 하여 보편성 테스트를 완료하였다. |
| 라즈베리파이에 6lbr 설치 | 라즈베리파이와 slip-radio를 XBee Dongle Interface를 통해 결합 한 이후에 라즈베리파이에 6lbr환경을 설치하여 라우터 역할을 하게 하고, 이더넷 망과 wsn 망의 연동이 이루어 지도록 하였다. |
| 라즈베리파이 환경설정 | 라즈베리파이의 OS인 wheezy를 설치하였으며, 포멧 이후 |

| | |
|---|---|
| | 각종 환경 설정을 하였다. 특별히 원격조정을 GUI적으로 하기 위하여 xrdp를 이용하였으며, 터미널에 대한 원격조정을 위해 putty를 이용하였다. |
| JLink Debbuger, JLink GDB Server 구동 | OpenMote에 코드를 fusing하기 위하여 JLink GDB Server를 구동 시켰다. |
| Ultrasonic sensor 연결 | Simulation 프로그램을 위하여 라즈베리파이 server에 resource value로 sensor와의 거리값을 부여하였다. ultrasonic sensor을 GPIO를 통하여 연결시켰으며, wiring Pi를 통하여 제어하였다. 특별히 자체제작 server 코드의 resource 와 연동시켜 client가 그 값(거리)을 받게 끔 하였다. |
| CoAP resource 제작 | 자체 제작 server코드에서 distance라는 resource를 만들고 GET메서드에 대한 핸들러를 추가하였다. |
| JAVA GUI 제작 | client상에서 user와의 interface를 위하여 GUI를 작성하였다. 특별히 hi.java코드를 작성하여 서버로부터 요청 한 값을 읽을 수 있도록 하였으며, 그 값이 20cm 이하가 될 경우 Test.java에서 화면이 경보화면으로 전환되도록 코딩하였다. |
| 6lbr+tinydtls 디버깅 및 코드 수정 | JLink Debbuger를 통한 디버깅을 할 수 는 없었지만, putty창에서 serial에 원격 접속하여 로그를 찍는 방법은 이용할 수 있었다. 이를 토대로 6lbr의 CoAP+TinyDTLS 코드의 문제점을 파악하고, peer의 handshake parameter의 initialization 코드를 수정하였다. |

**B.2 Economical, Industrial and Social Effects: 본 설계와 관련된 환경적, 사회적, 경제적, 문화적, 세계적 영향력을 이해하고 분석(개인별로 작성)**
**: 이 프로젝트의 결과물이 (미미하더라도) 영향을 미칠 만한 (연관된) 산업계 내에서의 기술적 효과, 경제적 효과 (시장형성), 사회적 효과(보건, 안전, 환경, 문화)는 무엇인가? 이 프로젝트 결과의 기독교 세계관적 함의는 무엇인가? (본 설계결과물이 성공적으로 사회에서 유통되고 사용된다고 가정하고 작성)**

- 김연희:
  본 CoAP + TinyDTLS 캡스톤 프로젝트 설계를 통하여 사물인터넷 상용화 연구에 보탬이 될 것이라 생각한다. 사물인터넷은 이제 전 세계적으로 사용될 예정이기에 보안이 필수적이다. 보안이 적용되지 않으면 크게는 생명에 까지 위협을 줄 수 있다. 따라서 경제적으로 사물인터넷 시장을 형성하는 데에 도움을 줬다고 생각한다. 또한 보안성을 제공함으로써 발생할 수 있는 위험 요소들을 제거하였다고 생각한다. 이 프로젝트가 사회에서 성공적으로 유통된다면 사물인터넷에 대한 해킹 문제를 해결할 수 있기에 기독교적으로도 의미를 가질 수 있다.

- 김진우:
  (기술적 효과)
  본 설계물이 가지고 있는 장점은 현 보안이 취약한 사물인터넷 네트워크에 보안을 적용하였다는 점이다. 본 프로젝트에서 사용한 Coap은 IETF에서 사물인터넷을 위하여 표준으로 정한 프로토콜이 앞으로 산업 시장에서 많이 사용될 것이라 기대한다. 또한 TinyDTLS는 기존의 TCP를 위한 SSL, UDP를 위한 DTLS와 마찬가지로 사물 인터넷을 위한 보안 프로토콜로 많이 사용될 것이라 예상한다. CoAP+Tinydtls 결합은 각 센서들이 탐지한 데이터들이 암호화된 것을 보여 주고 사물 인터넷을 판매하는 산업시장에 실용적인 보안 솔루션을 제공한다.

(사회적 효과)

현재 사물 인터넷 네트워크는 필요성이 증대되는 만큼 보안에 대한 이슈 또한 점점 증가되고 있다. 특히 사물 인터넷을 사용하게 될 분야는 단순히 가정 안의 온, 습도 정보 뿐만 아니라 개인의 건강과 같은 사생활에 관련된 데이터 분야로 점점 확대될 것이고 이를 활용한 빅데이터 분석은 한 개인의 생활에 상당한 영향을 미칠 것이다. 사물 인터넷을 통해 전송되는 데이터가 네트워크 상에서 무방비로 노출된다면 악의를 가진 제 3자에 의해 무단으로 사용되고 개인정보의 노출 위협 및 악용의 여지가 있다. 따라서 제 3자가 네트워크 상에 있는 데이터에 쉽게 접근하지 못하고 나아가 데이터를 분석하지 못하도록 본 프로젝트는 암호화된 데이터 통신을 솔루션으로 제공하고 그 방법으로 사물 인터넷에 특화된 Coap 과 경량화된 보안 프로토인 Tinydtls를 사용하여 현재 사물 인터넷의 기술적인 부분의 통합과 함께 보안을 적용하였다. 이를 사용하게 될 산업시장이나 일반 사용자는 보다 안전한 통신을 사용하여 개인정보의 노출과 위협을 줄일 수 있고 기업 입장에서 해커에 의한 위협 risk를 줄일 수 있다.

(기독교 세계관)

사물 인터넷이 다가오면서 한 개인이 인터넷에 노출되는 시간이 점점 더 많아질 것이다. 무방비 상태로 인터넷에 노출되는 개인은 자신의 정보가 어디로, 어떻게 흘러가서, 어떻게 이용되는지 모른다. 이들의 정보를 1차적으로 보호해 주는 역할이 중요하다. 사물 인터넷에 보안 프로토콜 적용은 개인의 사생활 정보가 1차적으로 유출되는 것을 방지해준다. 물론 직접적인 해커의 공격 발생시 유출 자체를 막을 수는 없지만, 유출된 데이터는 암호화 처리되어 있기 때문에 해커나 혹은 제 3자의 의해 분석, 이용, 가공되는 것을 막아준다. 이러한 보안 환경이 가장 필요한 대상은 컴퓨터에 대한 지식이 부족하고 전문적인 지식에 대한 접근성이 부족한 일반 사용자들이다. 기독교 세계관적인 의미에서 공학자는 단순히 일반인들에게 편의를 위한 공학적 상품만을 주기보다는 컴퓨터라는 전문성에서 주변부로 밀려나는 일반인들을 위해 공학적 지식 자체가 아니더라도 일반인들을 이해할 수 있을 정도의 공학적 지식을 얘기해주어야 한다. 가령 이 프로젝트는 학부생 수준이면서 동시에 사물인터넷 보안의 중요성을 담고 있기 때문에 일반 사용자들에게 네트워크에 대한 기초 지식과 함께 최소한의 기초 이론과 동작 흐름을 얘기해 줄 수 있고 나아가 보안에 대한 중요성을 환기시켜 줄 수 있다. 이는 기독교적인 의미에서 자신의 직업과 자신의 직업이 사회에 미칠 영향에 대한 책임의식을 이행할 수 있다.

- 한영광:

  먼저, 본 프로젝트에서 한 설계가 산업계 내에서 미칠 기술적 효과로는 세계 표준을 따라서 사물인터넷 플랫폼을 구축하여 여러 테스트를 해볼 개발자 혹은 기업체에 하드웨어적인 부분에서 어떠한 구조로 설계할지 또한 그 구조에 적합한 쉽게 구할 수 있는 하드웨어로는 어떤 것이 있는지에 대한 가이드 라인을 제시해 주었다고 볼 수 있다. 또한, 하드웨어가 플랫폼에 적합하게 쓰이기 위해 필요한 기본적인 환경 설정에 대한 가이드 라인을 제시해 주었다. (예를 들어, 6lbr, slip-radio, contiki os의 사용 등) 그리고 현재, 많은 개발자들이 iot 환경을 구축하기 위해서 여러 soc들에 대한 적합한 os 코드와 application layer에 대한 소스코드를 개발 중에 있는데 본 프로젝트에서는 국제 표준을 가장 잘 지키고 보편적으로 사용할 수 있는 c – implementation library인 libcoap 과 tiny-dtls의 결합을 시도 하였다. 그리고 이러한 시도는 후에 coap 통신을 통한 iot 환경 구축에 대한 오픈 소스 제작에 있어서 하나의 기술적 가이드라인을 제공했다고 볼 수 있다.

  또한, 경제적 면에서는 만약 사물인터넷에 관한 인프라가 많은 나라에 널리 적용되어 현재 인터넷 망이 구축된 것처럼 사회에서 떼어낼 수 없는 부분이 되었을 때, 각 나라간의 인프라 연결 또한 필연적이 될 것이며 이런 상황이 되었을 때, 각 나라간에 대한 연결을 위해선 국제 표준을 따라야 할 상황이 올 것이다. 이런 상황에 대비하여 국제 사물 인터넷의 표준 프로토콜인 CoAP에 대한 보안 솔루션을 선점하여 기술적인 권리를 갖고 있는다면 후에 사물인터넷에서의 보안 솔루션이 각광을 받게 되었을 때 유리한 자리를 차지 할 수 있을 것이다. 많은 보안 솔루션이 제시되어 하나의 시장

을 이룰 수 있지만 결국엔 표준을 따르고 가장 효율적인 동작을 제공하는 보안 솔루션 하나만 살아남을 것으로 생각이 된다. 이에 본 프로젝트에서 제시한 tiny-dtls와 결합되어있고 표준을 따르는 coap + tinydtls 라이브러리는 유리한 자리를 차지 할 수 있을 것이라고 생각된다.

본 프로젝트는 기본적으로 사물 인터넷의 보편화가 된 사회를 염두하여 진행한 것이다. 사물 인터넷이 사회에 보편화 되면 수많은 정보들이 인터넷을 통해 전달 될 것이며 이에는 인간의 생명에 영향을 줄 수 있을 정도로 중요한 정보들도 전송 / 교환 될 것이다. 이에 본 프로젝트에서 진행한 정보의 암호화는 악의를 가진 사람들로부터 중요한 정보들을 지켜낼 수 있을 것이다.

본 프로젝트에서 진행한 결과물은 기독교 세계관에 비추어 보아 예수님께서 많은 사람들에게 도움을 주고 특별히 약자들을 위해 많은 사랑을 배푸셨듯이 많은 사람들에게 좀더 안전한 삶을 제공하며 개인 정보 유출을 막아주고 사회적 약자들을 위해서는 무료로 플랫폼을 제공하여 도움을 제공할 수 있을 것이다.

- 위성일:

CoAP프로토콜은 IETF(Internet Engineering Task Force)기관에서 선정한 사물인터넷 통신에 있어 표준화된 프로토콜이다. 인터넷 네트워크를 넘어 컴퓨터공학의 이슈에 있어 보안적 요소는 새로운 패러다임이 나올 때 마다 필수 불가결적으로 중요한 요소이다. 비록 RFC 7252에 의하면 CoAPS(constraint application protocol security) 에 대하여 정의가 되어 있으나 UDP기반의 DTLS를 바탕으로 하며 이는 제한된 리소스에 있어 많은 부담이 된다. 본 프로젝트에서 제시하는 방향은 CoAP프로토콜 에 대한 보안적 이슈를 바탕으로 한정된 node 환경에서 효율적으로 application data 를 암호화 하는데 있다. CoAP이라는 표준 프로토콜에 그 특성과 맞는 보안 프로토콜인 tinydtls를 결합함으로써 미래에 전망이 있는 사물인터넷 산업계에서의 보안적 요소의 방향과 플랫폼을 제시 했다는데 큰 의미가 있다.

경제적으로 최근 한국사회에 대두가 되고 있는 TK GIGA Iot홈 매니저, LG유플러스 Iot 사물인터넷 등이 등장하면서 사물인터넷 보안 솔루션 시장도 점차 형성되고 있는 실정이다. 물론, TinyDTLS에도 몇가지 취약점이 있지만, 본 프로젝트의 성과물이 보안 솔루션 시장에서 사용될 기술적 요소 중 하나가 될 수 있을 것이라 예측되며, 이에 따라 경제적으로도 많은 영향을 미칠 수 있을 것이라 생각한다.

사물인터넷에 있어 하나의 보안에 대한 페러다임이므로 사회적으로도 개개인의 '안전' 에 초점이 맞추어 져 있다. 개인 정보의 노출을 막을 수 있으며, 외부의 공격으로부터 사람을 보호함으로써 기능의 편의성을 걱정 없이 이용하게 끔 한다. 비록 이 보안적 페러다임이 외부의 공격으로부터 뚫릴지라도 그에 대한 새롭고 강력한 보호대책이 성립되므로 어떠한 방향이든 사람의 '안전'에 큰 영향력을 미칠 것이다.

기독교 세계관적으로 사람은 온전한 하나님의 자녀로써 하나님의 온전한 것들을 이루기 위하여 이 세상가운데서 노력해야 한다. 즉 하나님의 자녀 됨으로써 '존귀함', '존엄성'을 이루기 위하여 노력해야 한다. 보안의 취약함으로 인하여 개인 정보의 노출이 일어난다면, 그 정보가 세상가운데 악용될 수 있으며 하나님의 영광을 가리는 일에

이용 될 수 도 있다. 사람을 통해 자신의 온전함을 보여주시는 하나님의 뜻에 합당하게 보안적 요소는 적용될 수 있을 것이다.

**B.3** ***Engineering Ethics (개인별로 작성) :***

- 김연희: (1 페이지)

(1) 이해 당사자로는 사물인터넷 물품을 판매하는 업체, 사물 인터넷 보안 솔루션 개발 업체, 사물인터넷 기능을 더 연구하고자 하는 대학원생 혹은 학부생, 사물인터넷을 이용하는 사용자 등이 있다.

(2) 본 팀이 적용한 보안 솔루션인 TinyDTLS의 취약점을 발견하여 공격한다면 오용할 수도 있을 것 같다. PSK 문제나 혹은 사물인터넷 메시지의 Fragment에 대해 공격한다면 이를 해결하기 위한 또 다른 솔루션이 있어야 한다.

(3) 이를 예방하기 위해서는 대안 솔루션을 개발해야 할 것이다. 하나의 방법은 PSK 모드를 사용하지 않는 것이다. 이를 위해서는 DTLS handshake 과정을 대신 해주는 다른 장치가 중간에 있으면 된다. 혹은 Fragment 공격에 대비하기 위해서 다른 보안 프로토콜을 사용할 수도 있다.

- 김진우: (1 페이지):

(1) 본 프로젝트 팀은 사물 인터넷 보안의 구현 모델을 제시하였다. 향후 사물 인터넷을 통하여 산업 시장에 진출하는 IT 기업들에게 적절한 보안 솔루션을 제시해 줄 수 있다. 특히 유틸리티(전기, 가스, 수도) 분야에 가장 큰 혜택을 줄 수 있을 것이다. 유틸리티(전기.가스.수도)는 가정과 기업, 국가에서 사용하게 될 에너지 자원을 효율적으로 배분, 관리를 위한 시스템을 계획 중이다. 예를 들어, 원격으로 검침하고 배관 분석을 통해 누수, 누전 방지 등의 방법을 지속적으로 발전시키고 있다. 이 분야에 사물 인터넷이 적용된다면 단순히 송전 시스템을 보호하고 제어하는 지금의 역할에서, 발전소의 송전시스템이 기업 및 일반사용자들과 지속적 커뮤니케이션 상태를 유지하여 가전제품들이 스스로 적은 전력을 사용할 수 있도록 최적화할 수 있다. 즉 에너지의 효율성을 극대화 시킬 것이다. 사물인터넷 보안은 바로 이러한 에너지 시스템이 국민의 안전과 생활에 밀접하기 때문에 해커에 의해 야기될 수 있는 광범위한 사회적 문제를 보안을 적용함으로써 미리 예방할 수있다.

(2) Coap+TinyDTLS는 사물 인터넷을 위한 보안 솔루션을 제공하고 있지만 TinyDTLS가 안고 있는 취약점이 있다. 실제 사용자들에게는 이용상의 문제가 발생하지 않을 것이라 예상하지만 기업 입장에서는 고려해야 할 부분은 TinyDTLS가 단편화(Fragmentation) 과정에서 문제가 생길 수 있다. 즉 기업입장에서는 사물인터넷이라 하더라도 노드가 처리해야야할 데이터의 size 등을 충분히 고려해야하고 사전에 테스트 과정이 필요하다. 또한 TinyDTLS는 DTLS를 기반으로 만들어진 프로토콜이기 때문에 DTLS가 안고 있는 취약점들이 있다. 가령 DTLS는 데이터 자체

를 암호화 하여 통신하기 때문에 네트워크 중간에서 이 패킷을 가져가도 실제 내용을 확인할 수 없다. 하지만 비정상적인 DTLS 핸드쉐이크나 비정상적인 DTLS 프래그먼트 같은 공격에 클라이언트나 서비스 거부 공격이나 임의의 코드가 실행될 가능성이 있다.

(3) 현재 DTLS가 가지고 있는 취약점은 계속해서 업그레이드 중이다. 사물인터넷 특성상 모든 노드들이 인터넷으로 연결되어 있기 때문에 인터넷을 통한 접속이 누구에게나 가능하다. 따라서 학부생 수준에서 생각하기에는 노드들을 연결하는 네트워크 망을 폐쇄망으로 구축하여 외부로부터의 접근을 제어하는 게 도움이 될 것이라 생각한다. 즉 서버를 위한 네트워크 망이 별도로 존재하여 다른 곳에서의 접속이 불가능하게 하고 데이터를 보관하고 요청하는 클라이언트의 보안 기술을 개발한다면 보안 취약점을 완화시킬 수 있다고 생각한다. 또한 네트워크 공격의 한 형태로 두 노드간에 중간자가 개입하여 비정상적인 메시지를 전송하는 경우가 많은데 이러한 메시지를 구분하도록 하는 인증절차를 좀 더 강화할 뿐만 아니라 이에 따른 시간과 여러가지 리소스 소모량을 개선하여 실제 사용자에게는 사용에 따른 불편이 없도록 해야한다.

- 한영광: (1 페이지): (1) 사물인터넷 물품을 판매하는 업체
사물인터넷 보안 솔루션 개발 업체
사물인터넷 보안 연구를 목표로 하는 학부생 혹은 대학원생
국가에 사물인터넷 플랫폼 구축을 하려는 정부기관

(2) 제품을 제공 하였을 때, 기존 통신사가 제작한 사물인터넷 프로토콜과 호환의 문제가 생길 수 있을 것이다. 현재 통신사들이 자체적으로 만든 많은 사물인터넷 제품들이 자사에서 개발한 프로토콜을 이용하여 제작 하였을 경우, 국제 표준 프로토콜을 사용한 본 프로젝트에서의 제품과는 호환이 되지 않을 수 있으며, 그렇다면 기존 통신사와의 솔루션 제공에 있어서 마찰이 있을 것으로 예상이 된다.

(3) 앞에서 언급했듯이 본 프로젝트에서 제안한 제품이 보급되었을 경우, 기존에 사물인터넷 물품을 제공하고 있는 통신사와의 납품 마찰이 예상 된다. 이러한 상황을 완화하기 위해서 기존 통신사에서 제품에 맞게 제작한 프로토콜을 대신해 국제 표준 프로토콜을 사용하더라도 제품에 미치는 영향은 없는지에 대해 충분한 대화를 통해 미래를 바라보아 프로토콜의 변경 및 현 제품의 통합 솔루션을 사용하도록 설득할 수 있다. 그리고 개발자로써 해당 통신사에서 제작한 프로토콜을 사용해야 하는 경우, coap + tinydtls 통합 솔루션 제공보다 해당 업체에서 사용하는 프로토콜에 보안 프로토콜을 적용하는 방법을 생각해

볼 수 있을 것이다.

- 위성일: (1 페이지): (1) 본 프로젝트의 제품과 관련된 이해 당사자는 이러하다. 사물인터넷 물품을 판매하는 업체, 사물인터넷 보안 솔루션 개발 업체, 사물인터넷 보안 연구를 목표로 하는 학부생 혹은 대학원생, IETF, 네트워크 보안 업체, 사물인터넷의 기능을 이용하는 사용자 등

  (2) 무엇보다도 TinyDTLS는 PSK(Pre Shared Key)를 client와 server가 미리 가져야 하는데 그 key가 제품 배급 과정중 에 client간 중복돼서 배급이 될 경우 보안적으로 훗날 문제가 될 수 있다. 또한 통신상의 암호화 key를 알아내기 위하여 host자체에 공격을 가하여 key를 습득할 경우 이 배급된 사물인터넷 기능을 이용하는 사람의 정보가 위험해 질 수 있다.

  (3) 먼저 ssl에서 인증서를 보관하기 위하여 하나의 큰 기관이 존재하는 것 처럼, key를 관리하는 기관을 세워두어 key가 중복이 생기지 아니하고 각 client에게 unique하게 분배되도록 해야 한다. 또한 secure 터널뿐만 아니라 server와 client 즉, 각 host 단에서의 보안에 대한 대책도 강구해야 한다. 추가적으로, tinydtls자체적으로 fragmentation으로 인한 보안의 취약점 등을 해결해야 한다.

**B.4 전문인으로서 자기주도적 평생학습 계획**

- 김연희: (1 페이지)

  컴퓨터공학 공부를 하면서 1년에 걸친 커다란 프로젝트에 참여한 것은 이번이 처음이었다. 프로젝트 주제 선정에 있어서 내가 관심있는 '네트워크'를 선택하였다. 네트워크는 공학도라면 필수적으로 알아야 하는 과목이다. 왜냐하면 네트워크 없이는 정말 아무것도 할 수 없는 시대이기 때문이다. 이제는 사물조차도 네트워크에 연결된다고 하니 이 분야에 대해 더 연구하고, 또 이후에 취직하고 나서도 적용해 보고 싶어서 이 주제를 선택하였다. 캡스톤 프로젝트는 학기 중의 프로젝트와는 다르게 하나의 생소한 주제에만 집중하여 연구하기 때문에, 문제가 발생하면 교과서나 인터넷 조차에서도 답을 쉽게 구할 수 없다. 특히나 사물인터넷의 경우에는 최신 기술이기에 직접 제작한 제작자와 연락을 주고받으며 피드백을 받는 경우도 많다. 캡스톤 프로젝트를 통하여 얻은 점은 문제 해결 능력인 것 같다. 어떠한 어려운 문제가 있더라도, 그 문제를 해결할 방법이 있다는 점을 배웠다. 다만 프로젝트가 얼마나 걸릴지 예상할 수 있으면 더 좋을 것 같다. 그리고 이상적으로는 모든 팀원이 열심히 참여하면 좋겠지만, 그렇지 않은 경우라도 그것에 동요하지 않고 내 할 일과 참여하지 않은 학생의 할 일까지 열심히 해야한다는 점을 배웠다. 이후에 취직을 하고 난 후에도 과제의 난이도적으로 어려운 문제와, 모든 팀원들 혹은 회사원들이 똑같이 열심히 참여하지 않는다는 문제에 대해서 분명 부딪혔을 것 같다. 이번에는 잘 대처하지 못하였고, 따라서 캡스톤 프로젝트의 원래 목표를 달성하지 못했지만, 다음에는 어떻게 해결해야 하는지 배울 수 있었다. 즉, 내게 주어진 일을 지금처럼 책임감 있고 열심히 하되, 내 시간을 다른 사람들보다 많이 써서 했다고 할지라도 그것을 정량적으로 재서 내 자신을 힘들게 만들지 않아야 한다는 점을 배웠다. 네트워크 분야는 정말 재미있는 것 같기에, 이 재미있어 하는 마음을 유지하면서 지금의 열정을 잃지 않고 한다면 내 전문성도 자연스럽게 늘어날 것이라 확신한다. 나는 다음학기에 취업 준비를 할 것인데, 네트워크 쪽으로 최대한 준비할 것이다. 취업이 된 후에는 내가 좋아하는 네트워크 분야를 직접 실생활에 적용해볼 수 있다니 벌써 기대된다.

- 김진우: (1 페이지)

  재학 기간동안 프로그래밍 실력이 부족하여 방학을 이용하여 opentutorial.org와 같은 기관을 이용하여 자바와 C에 대한 기본적인 이론을 배우고 실습하였습니다. 특히 대학교 수업에서 다룬 언어는 주로 C 기반이었기 때문에 자바를 경험하는 절대 시간이 부족하여 위의 사이트를 이용하여 복습을 하였습니다. 또한 평소 보안에 대한 관심이 있어서 '해커스쿨'이라는 사이트를 통해 기초적인 개념을 공부하였습니다. 앞으로 이러한 관심을 가지고 보안에 대한 공부를 지속해 나갈 예정입니다. 현재 정보보호 대학원에 진학할 계획을 가지고 있고 대학원 진학 전 최근에 생긴 정보보호자격증 시험을 위한 공부를 진행하며 기초 지식을 공부함과 함께 한국정보기술연구원(KITRI)에서 진행하는 정보보호 교육 커리큘럼에서 교육을 받으려고 합니다. 정보보호 분야에서 하고 싶은 분야는 보안 컨설턴트입니다. 보안 컨설턴트는 회사 내부의 취약점을 발견해 그에 맞는 보안대책을 설계하고, 각종 보안위협으로부터 기업을 보호하는 역할입니다. 요구되는 전공지식은 시스템

OS, 데이터베이스, 정보 시스템 동작 운영원리 등 정보시스템에 대한 이해입니다. 하지만 수업에서 배울 수 없는 부분인 실제 운영체제나 솔루션의 아키텍처 이해가 필요하기 때문에 교육을 받을 수 있는 관련된 국가 기관을 먼저 찾아 볼 생각입니다. 보안 컨설턴트가 되기 위해서는 커뮤니케이션 스킬을 위한 발표, 프레젠테이션, 보고서작성 능력, 정보보안관련 지식, 기본적인 회화가 가능한 영어실력이 있어야 합니다. 평소 국제개발경영학회에서 보고서와 발표에 대한 경험을 쌓아왔고 방학을 이용하여 기본적인 영어와 회화실력을 공부하였습니다. 향후 대학원에 진학하여 보안에 관련된 리버스 엔지니어링과 시스템 해킹, 네트워크 해킹을 공부할 것입니다. 이러한 전공지식을 바탕으로 개인적인 블로그를 운영하여 직접 개발한 보안 알고리즘이나 이론을 게시하고 다른 필요한 개발자들과 공유하고 싶습니다

- 한영광: (1 페이지)

  먼저, 본 프로젝트를 진행하면서 프로젝트에 필요했던 핵심 지식인 네트워크와 보안 및 임베디드에 관련한 수업을 학교에서 한가지도 수강하지 못한 채, 공프기를 시작하게 되었습니다. 방학 때는 이에 대비하여 coap 프로토콜에 관한 공부를 지속적으로 하였으며 2학기 수업에서는 네트워크 수업과 컴퓨터 보안 수업을 수강 신청하여 캡스톤에 필요한 지식들에 대해 배웠습니다. 네트워크 수업에서는 네트워크에서 중요한 각 layer에 대한 여러 프로토콜들과 기본적인 기능들을 배웠고 컴퓨터 보안 수업에서는 tiny-dtls에서 필요한 기본 개념들이 적용된 ssl 프로토콜에 대해서 알게 되었습니다. 그리고 사물인터넷 플랫폼 구축 목표를 하였기 때문에 임베디드 코드들을 실제 하드웨어에 fusing 하는 법을 알아야 했습니다. 이에 대해서 비록 임베디드 수업은 듣지 못했지만 각 하드웨어에 필요한 datasheet를 읽어보고 fusing 하기 위해 어떠한 환경설정 및 도구들이 하드웨어 별로 필요한지 공부해 보았습니다.

  이제 저는 올해면 졸업 예정을 앞두고 있는데, 캡스톤 주제를 고를 때도 평소에 관심이 있었던 보안 주제를 선정하여 지금까지 캡스톤 진행 및 공부를 하였고, 또한 컴퓨터 보안 수업도 4학년 과목이지만 3학년때 미리 들었습니다. 그리고 대학교 이후에 진로로 대학원 진학을 생각 하고 있는데, 학부 때 배운 이러한 지식들을 통해 성공적인 진학을 위한 발판으로 만들려고 합니다. 그리고 보안 대학원에서 쌓은 실력을 바탕으로 국가정보원 및 기업의 컴퓨터 보안팀으로 들어가 외부로의 침투에 대한 방어를 위해 힘쓸 것입니다. 또한, 컴퓨터 보안이야 말로 끊임없이 새로운 공격법이 만들어지고 연구되기 때문에 현재 알려진 보안 방법 이외에 다른 잠재적인 위협에 대해 끊임없이 공부하고 가능성에 대해 생각해보며, 보안에 관심이 많은 사람들이 있는 그룹에 들어가 최신 정보 습득을 위해 노력하려고 합니다.

- 위성일: (1 페이지)

  필자는 문과 출신으로 한동대학교에 입학하였다. 컴퓨터에 많은 관심이 있었으나 1학년 과정가운데 전공적으로 접근 하기 위하여 수학, 과학 등의 지식습득에 초점을 맞추어 공부를 하였다. 수업 외에 도 선형대, 미분방정식에 대하여 공부하였고, 전공과 연관시키기 위하여 주변 전산전자를 희망하는 새내기들과 달리 미리 이산수학 강의를 들

었다. 전공을 시작한 이후에는 학업에 항상 쾌감을 느끼며 심화적인 차원까지 들어가고자 하였다. C 프로그래밍 수업에서 변수할당에 대해 배웠으면 그 날 집에 가서 복습적 차원에서 그에 대한 메모리 구조는 어떻게 되고, CPU가 어떤식으로 이 명령어를 처리하는지 더 나아가 포인터에 대하여 미리 예습하는 등의 노력을 하였다. Java를 모두 수강한 이후에는 과학고에서의 논문작업을 도우며 시뮬레이션 프로그램을 만드는 과정가운데 multi-thread를 다뤘으며, OpenGL에 대하여 공부하는 시간들을 가졌다. 그 외에도 OS, 네트워크, DS, 알고리즘, DB등 을 배우는 과정 가운데 배우는 내용에 대하여 심화적으로 접근하고자 하였다. 또한, 보안에 대한 공부를 위하여 보안전문동아리인 'GHOST'동아리에 들어가 리눅스, 유닉스, 네트워크, 리버싱, 네트워크 해킹 등을 일반 다른 학부생들보다 일찍 공부하며 학문의 깊이를 더하였다. 다만 자신의 전문성을 높이기 위하여 배운 내용에 대하여 구글링을 통해 심도 있는 공부를 하였지, 여러 변화와 최근 컴퓨터 공학 동향에 맞는 공부를 깊게 하지 못하였다. 인공지능, 빅 데이터, 데이터 마이닝 등에 대하여 알아가고자 하는 마음이 있었으나, 자신 학업에 치우쳐 따로 심도 있는 공부를 하지 못하여 아쉬움을 남긴다.

네트워크나 보안쪽으로 대학원을 생각하고 있는 필자가 더욱 전문성을 높이기 위하여는 무엇보다도 수업을 통해 습득하는 지식에만 초점을 맞추지 말고 독자적으로 하는 공부와, 탐구에도 자신의 발전을 위해 많은 기여를 할 필요가 있다고 생각한다. 특별히 보안에 대해서는 모든 컴퓨터 공학의 분야를 총괄하여 전문적 지식과, 배경이 필요하다고 생각한다. 한 교수님 밑에서 일하는 연구원으로써 원활한 소통을 위해 공학적 대화방식을 습득하려 노력하고, 더욱 효율적인 구글링 방법에 대해 알아가면서, 영어에 대한 기본 지식도 늘려야 한다고 생각이 된다. 무엇보다도 이 지식과 개념을 알아가고자 하는 열정과 궁금증을 계속 유지하고자 노력해야 한다.

**B.5 프로젝트 수행 개인별 소감**

- **김연희**

프로젝트가 우선 끝났다는 것이 너무 행복하다. 이 프로젝트를 위하여 다 합쳐서 두 달은 밤을 샜을 것이다. 비록 프로젝트의 원래 목표는 달성하지 못하였지만, output을 보여주기 위해 성일이와 제작한 ultrasonic sensor와 raspberry pi CoAP+TinyDTLS 라도 완성할 수 있어서 다행이라 생각한다. 사물인터넷의 표준 프로토콜인 CoAP에 대해 연구할 수 있어서 좋았고, 하나의 프로토콜을 공부해서 메시지 포맷과 여러 옵션들을 공부하면서 protocol 자체도 공부할 수 있어서 정말 유익했다. 그리고 캡스톤 축제에서 사용할 포스터와 UCC 내용을 성일, 진우와 나 세명이 함께 밤을 새면서 이때까지의 내용을 모두 정리해 보았는데, 결코 쉬운 내용들이 아니었고 마치 전문가가 된 것처럼 뿌듯했다.

이번 프로젝트가 내용적으로 난이도가 높아서 어려운 점도 있었지만 의미 있었던 시간이라 생각한다. 왜냐하면 팀원들의 역할이 얼마나 중요한지, 그 팀원들을 생각하는 내 마음 관리는 어떻게 해야하는지 배웠기 때문이다. 나의 경우에는 객관적으로 보았을 때 맡은 바 역할을 다 하지 않는 사람을 정말 마음적으로 미워하는 편이고 그것을 또 숨기지 못하는 편인데, 이 부분에 대한 훈련을 많이 받은 것 같다. 그럴 시간에 그 사람 몫을 다 우리 팀 좋은 것이라 생각하고 더 몰입했어야 하는 것 같다. 세상에는 여러 종류의 사람이 있으므로 그것을 하나하나 내가 옳다고 여기는, 또는 내가 한 만큼 다른 팀원들도 해야 한다는 그런 기준을 낮출 필요가 있는 것 같다. 왜냐하면 그렇게 하는 것 조차 내 에너지 소비이고 마음을 어렵게 만들기 때문이다. 이번 프로젝트를 어제는 되었던 코드가 오늘은 안되고, 또 버전 문제로 안되고, 하드웨어의 작은 결함으로도 안되는 정말 황당한 경우를 많이 겪으면서, 다시 하면 된다는 용기를 갖게 된 것이 감사하다. 프로젝트를 진행하면서 힘들었던 만큼 하나님을 더 깊게 만난 것 같다. 이제 취업 준비를 열심히 할 것이다.

이종원 교수님 정말 감사하고 사랑합니다!

- **김진우**

처음 공학프로젝트기획을 시작하면서 굉장히 많은 고민이 있었다. 전공 3년차이지만 기본적인 프로그래밍 언어를 활용하는 거 외에는 컴퓨터에 대한 전반적인 지식과 응용이 부족하였기 때문이다. 공학프로젝트기획에서 주제를 정할 때 고려했던 기준은 전산으로써 독학하기 힘든 분야를 선택하는 것이었다. 예를 들어 application이나 웹은 다른 과목에서도 충분히 배우고 독학이 가능할 것이라 생각하였고 따라서 팀웍으로 반드시 공부해야 하는 주제를 선택하고 싶었다. 'CoAP을 이용한 가상 구현'은 내가 고려하는 기준에 적합하였고 실제로 공학프로젝트기획 수업을 들으면서 이론적인 공부에 많은 시간을 투자하였다. RFC 표준 문서를 보면서 CoAP 과 TinyDTLS 프로토콜의 동작 flow를 충분히 이해하여야 두 프로토콜 간의 코드 결합이 가능하였다. 또한 이종원 지도교수님이 참고하라고 주신 논문이 굉장히 큰 도움이 되었다. 어려웠던 점은 네트워크에 대한 사전 지식이 없는 상태로 프로토콜을 공부해야 하기 때문에 이해하기가 쉽지 않았다. 캡스톤

디자인을 겨울 방학 때부터 시작하면서 원래 계획했던 CoAP 프로토콜의 임베디드 구현을 진행하였다. 하지만 인터넷을 통한 자료는 기본적인 방법을 알려주기는 하였지만 임베디드 주변 환경을 구성하는 부분은 우리 팀 스스로가 찾아야 되는 부분이었다. 따라서 프로토콜의 실제적인 결합 전에 개발 환경을 구축하는데 겨울 방학을 모두 사용하였기 때문에 시간이 많이 부족한 상황이었다. 개발을 진행하면서 가장 어려웠던 점은 TinyDTLS 코드 분석이다. TinyDLTS는 학교에서 강의로 배워왔던 C 기반의 소스코드가 아니라 임베디드를 위한 Contiki-OS 기반의 소스코드이기 때문에 기본적인 flow부터 사용하는 헤더파일과 라이브러리 등이 모두 달랐다. 사전에 Contiki-OS에 대한 지식과 경험부터 시작해야 분석이 가능한 코드였기 때문에 TinyDTLS와 CoAP 프로토콜 사이의 인터페이스만을 수정하는게 쉽지가 않았다. 전체적으로 아쉬운 부분이 많았지만 네트워크를 수업만으로 듣지 않고 직접 인터넷을 찾으면서 내가 원하는 하드웨어에 적용시키고 개발환경을 구성하고 실제 코드를 분석하는 경험은 굉장히 특별하였다. 특별히 이번 프로젝트에서 코딩 과정보다 소스 코드를 많이 분석함으로써 다른 코드에 대한 이해가 빨라졌다. 또한 네트워크 환경 설정에 필요한 기본적인 구성(라우팅 테이블, 스위치, prefix 등)을 생각해 볼 수 있었고 개발 단계에서 디버깅의 중요성을 알게 되었다. 네트워크에서 문제 발생시, 소프트웨어적인 부분에서 문제를 찾기 위해서는 wireshark tool과 소스코드가 임베디드적으로 표현될 수 있는 방법(예: LED 제어를 통한 코드 분석)이 필요하다. Wireshark 를 통해서 해당 패킷이 어느 계층까지 통과하고 어떤 정보를 담고 있는지 알게 되면, 해당하는 부분의 소스코드를 분석할 수 있다. 본 프로젝트를 진행하면서 리눅스 OS를 많이 사용하였다. 이런 경험은 리눅스에 쉽게 적응하게 되었고 실제 개발에 필요한 단축키나 빌드 및 컴파일러 과정을 이해하였으며 앞으로 임베디드 프로젝트 진행시 크로스 컴파일에 대한 개념을 가질 수 있게 되었다.

- **한영광**

  학교에서 배운 지식들이 이제 기업 혹은 연구소에 들어 갔을 때 어떠한 방식으로 쓰일 수 있을지 대략적인 감을 알게 된 것 같고, 어떠한 제품을 개발 하는 데 있어서 개발자 혹은 기술자들의 각자 맡은 역할이 모두 중요하다는 것을 알게 되었다. 그리고 팀 프로젝트에 있어서 구성원과의 의사소통에 대한 중요성도 깨닫게 되었다.

- **위성일**

  교수님의 지도를 이렇게 가까이서 받으며 큰 프로젝트를 수행 할 수 있어서 정말 유익했습니다. 특별히, CoAP, SSL, DTLS등에 대하여 RFC를 보며 심도 있는 공부를 할 수 있던 좋은 시간들이었습니다. 장기간 프로젝트를 진행하며 여러 좌절과, 힘든 일들이 있었지만, 팀원들과 교수님을 알아가고, 주어진 문제를 하나하나 해결해 나갈 때마다 그를 통해 큰 쾌감을 느낄 수 있었습니다. 특별히 팀원들 중 막내로써 어떻게 다른 팀원들을 대하고, 자신의 지식과 생각, 할 수 있는 능력을 어필 할 수 있을 지에 대한 고민이 많았는데, 결국 프로젝트를 진행함에 있어 큰 보탬이 된 것 같아 큰 기쁨이 있었습니다.

(비록, 완벽한 구현에는 실패하였지만.) 팀원들과 협동하여 스터디를 진행하고, 프로젝트를 진행하면서 분업, 서로의 부족함에 대해 채워주는 것의 중요성에 대해 느꼈습니다.

프로젝트의 과정가운데 자신의 지식의 한계와, 나의 부족함이 얼마나 큰지에 대해서 크게 느낄 수 있었습니다. 특별히 임베디드 프로그래밍을 듣지 않은 상태에서 라즈베리파이에 센서를 연결시키고, OpenMote를 이용하는 것들에 대하여 전체적 flow를 프로젝트 초기에는 이해하기 힘든 면이 있었습니다. 또한, 교수님과의 대화를 통해 자신이 얼마나 더 공부하고 이해하기 위해 노력해야 하는지에 대하여 알 수 있었습니다.

저는 졸업 후 대학원 진학을 목표로 공부하고 있습니다. 본 프로젝트 주제가 가고자 하는 대학원 분야와 크게 관련이 있어 후에 많이 도움이 될 것이라 예상합니다. 프로젝트의 과정을 통해 대학원에 가면 얼마나 더 어려운 내용을 배우게 될지 미리 예측하고 기대할 수 있었습니다.

이 프로젝트를 진행하게 해 주신 이종원 교수님, 팀원들에게 감사를 표하며 글을 마무리 짓습니다!