The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 18, No. 6, pp.53-61, Dec. 31, 2018. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2018.18.6.53
JIIBC 2018-6-6

# DTLS 보안기술이 적용된 CoAP/6LoWPAN 기반의 스마트 홈네트워크 시스템

# CoAP/6LoWPAN-based Smart Home Network system using DTLS

김연수, 김기태, 이보경*

**Yeon-Su Kim, Ki-Tae Kim, Bo-Kyung, Lee***

**요 약** 최근 사물인터넷 관련 기술이 급속도로 발전하고 있고 이에 따라 모바일 환경 중심의 홈 네트워크 시스템과 관련된 연구들이 활발하게 진행되고 있다. IETF(Internet Engineering Task Force)에서는 저전력, 저용량, 저성능 등 제한된 환경에서 인터넷과 연동하여 사물들을 동작시키기 위한 대표적인 기술로 CoAP/6LoWPAN 프로토콜을 제안하고 있다. 그러나 기존에 구현된 홈 네트워크 시스템은 보안 기능이 거의 구현되어 있지 않은 실정이다. IETF에서는 IoT환경에 적합한 UDP 기반의 보안 프로토콜로 DTLS(Datagram Transport Layer Security)을 권고하고 있다. 본 논문에서는 DTLS 보안기술을 적용하여 모바일 환경에서 CoAP/6LoWPAN 기반의 홈 네트워크 시스템을 동작시키고 제어하는 시스템을 구현하였다. 또한 본 시스템을 활용하여 DTLS 동작 절차에 따른 데이터 전송 시간을 측정하였으며 향후 보안 프로토콜의 성능 개선의 필요성을 제안한다.

**Abstract** Recently, technologies related to the Internet of Things have developed rapidly and research on mobile environment home network systems is actively in progress. The Internet Engineering Task Force (IETF) Working Group proposed the CoAP/6LoWPAN technology as a suitable protocol for internetworking IoT devices with the Internet in a limited environment and adopting it as a standard. However, the existing home network systems hardly include security protocols. IETF recommends DTLS(Datagram Transport Layer Security) on UDP as security protocol suitable for IoT environments. In this paper, smart home network system based on CoAP/6LoWPAN by using DTLS is implemented in mobile environments. The data transfer time is measured according to each procedure of the DTLS protocol and the need to improve DTLS protocol is suggested.

**Key Words :** Internet of Thing, CoAP,/6LoWPAN, Mobile Environment, security, Home Network System, DTLS

## Ⅰ. 서 론

초소형 임베디드 기술이 발전하면서 사물인터넷 분야는 더욱 활발하게 연구되었고, 그 응용 분야인 홈 네트워크는 인간의 삶의 질을 높이는데 크게 기여할 것이라고 예상되며 많은 기대를 받고 있다. 현재 네트워크에 연결된 IoT(Internet of Things) 디바이스는 국내의 높은 ICT(Information Communication Technology) 인프라를 통해 매년 큰 폭으로 늘어나고 있으나, 그만큼 보안에 대한 위협이 커지고 있으며 이와 관련한 보안 대책은 아직

*정회원, 한국산업기술대학교, 컴퓨터공학부(교신저자)
접수일자: 2018년 10월 4일, 수정완료: 2018년 11월 4일
게재확정일자: 2018년 12월 7일

Received: 4 October, 2018 / Revised: 4 November, 2018 / Accepted: 7 December, 2018
*Corresponding Author: bklee@kpu.ac.kr
Dept. of Computer Engineering, Korea Polytechnic University, Korea

부족한 실정이다.[1]

IoT 환경에서 사용되는 무선 통신 칩들은 일반적으로 저성능, 저전력의 제한된 환경에서 작동하면서 기본적인 통신 기능만을 제공하기 때문에, 높은 수준의 보안 기술을 적용하기 어렵다. 최근 IoT환경에 보안을 적용하기 위한 프로토콜로서 IETF(Internet Engineering Task Force)에서는 DTLS(Datagram Transport Layer Security)프로토콜 사용을 권장하고 있다. 하지만 이는 TCP상에서 동작되는 TLS(Transport Layer Security)를 기반으로 하는 보안 프로토콜로서 RSA, ECDHE 등의 사전 비대칭키 교환을 통한 암호화를 제공하기 때문에 보안 기능은 훌륭하나, 적지 않은 크기의 메모리와 연산 과정이 필요하게 되어 많은 데이터 전송 및 소요시간을 필요로 한다.

따라서 본 논문에서는 DTLS 보안기술을 적용하여 모바일 환경에서 CoAP/6LoWPAN 기반의 홈 네트워크 시스템을 동작시키고 제어하는 시스템을 구현하였다. 또한 본 시스템을 활용하여 DTLS 동작 절차에 따른 데이터 전송 시간을 측정하였으며 향후 보안 프로토콜의 성능 개선의 필요성을 제안한다.

II장에서는 주요기술이 DTLS, 6LoWPAN, CoAP에 대한 관련 연구를 설명하고 III장에서는 해당 기술을 적용하여 구현한 스마트 홈네트워크 시스템과 보안 모듈에 대하여 설명한다. IV장에서는 구현된 시스템을 활용하여 보안모듈의 동작 과정에 따른 데이터 전송시간을 분석하여 보안모듈의 개선 필요성을 제시하며 V장에서는 결론을 맺는다.

## II. 관련연구

### 1. DTLS(Datagram Transport Layer Security)[2][3]

DTLS(Datagram Transport Layer Security)는 데이터그램 기반 응용 프로그램이 도청, 변조 또는 메시지를 방지하기 위해 설계된 방식으로 통신할 수 있도록 하여 데이터그램 기반 응용 프로그램에 보안을 제공하는 통신 프로토콜이다. UDP에서 TLS 프로토콜이 동작할 수 없는 이유는 단순하게도 비신뢰적인 전송으로 인한 패킷 손실 때문인데, 이를 해결하기 위한 몇몇 변경점 외에는 TLS와 유사한 프로토콜이라 할 수 있다. DTLS 프로토콜의 대표적인 변경사항은 다음과 같다.

- DOS 공격을 예방하기 위해 임시적인 Cookie를 교환한다.
- 메시지 손실을 처리하기 위해 Handshake Header를 수정 및 재조립한다, IP 단편화를 피하기 위한 DTLS 메시지를 단편화한다.
- 패킷 손실 처리를 위한 재전송 타이머를 구동한다.

또한 DTLS의 Record Layer에는 UDP 상에서의 원활한 세션 유지를 위한 Sequence Number 및 Epoch 필드가 추가되었다. 표 1은 DTLS 레코드 계층의 포맷을 보여준다.

**표 1. DTLS의 Record Layer 포맷**
**Table 1. DTLS Record Layer Format**

| 헤더 정보 | 길이(Bits) |
|---|---|
| Content Type | 8 |
| Version | 8 |
| Epoch | 16 |
| Sequence Number | 48 |
| Length | 16 |
| Fragment | 2^Length |
| 합계 | 96 + 2^Length |

### 2. 6LoWPAN(IPv6 over Low-Power Wireless Personal Area Network)[4]

6LoWPAN은 물리/링크 계층 프로토콜만으로 동작하는 개인 무선통신 프로토콜 상에 IPv6를 접목시켜 인터넷 연결이 가능하도록 하는 기술이자 IETF의 워킹 그룹 이름이다. 원래 6LoWPAN은 802.15.4(Zigbee) 상에서 센서 노드들을 네트워크에 연결시키기 위한 기술이었으나, 블루투스 4.1 버전 이상부터는 사물인터넷을 위한 IPv6 사용 표준이 제정되었고 지그비와의 작동 메커니즘이 유사하여 6LoWPAN을 통해 네트워크에 연결시킬 수 있게 되었다.

### 3. CoAP(Constrained Application Protocol)[5]

CoAP는 저성능, 저 전력 등 제한된 환경에서의 통신을 위해 개발된 UDP 기반 응용 계층 프로토콜이다. HTTP(HyperText Transfer Protocol)와 동일한 RESTful 아키텍처를 기반으로 설계되었기 때문에 HTTP와의 상호 변환이 용이하여 기존 인터넷 환경과의 자연스러운 통합을 도모하였으며, 헤더를 단순화 시켜 패킷을 경량화 하였기 때문에 전송 오버헤드가 줄어들었

다. 해당 프로토콜의 표준 문서는 IETF RFC 7252로 제안되었다.

## Ⅲ. 구현 통합 시스템

### 1. 통합 시스템 구성도[6-9]

본 논문에서 구현한 스마트 홈 네트워크 시스템은 스마트폰을 중심으로 각 방의 CoAP 서버와 통신하여 센서를 조작할 수 있는 구조이다. 서버는 블루투스를 통한 6LoWPAN으로 게이트웨이와 통신하며, 클라이언트는 집안의 WiFi를 통해 게이트웨이와 통신한다.

CoAP 클라이언트와 CoAP 서버에는 DTLS 프로토콜이 동작되고 있으며 CoAP 프로토콜을 통해 메시지를 교환한다. CoAP 서버에는 LED, 온도센서 등이 직접 연결되어 있고 필요에 따라 아두이노와 블루투스 통신을 하여 센서 노드 겸 CoAP 서버 역할을 하게 된다.

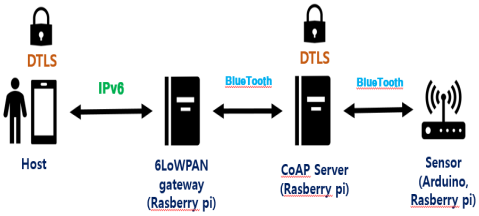


그림 1. 통합 시스템 구성
Fig. 1. Integrated System Configuration

6LoWPAN 게이트웨이와 CoAP 서버는 BLE와 6LoWPAN을 통한 IPv6 통신을 하며 이를 통해 CoAP 서버는 CoAP 및 DTLS 프로토콜을 상위 계층으로 두어 통신할 수 있게 된다.

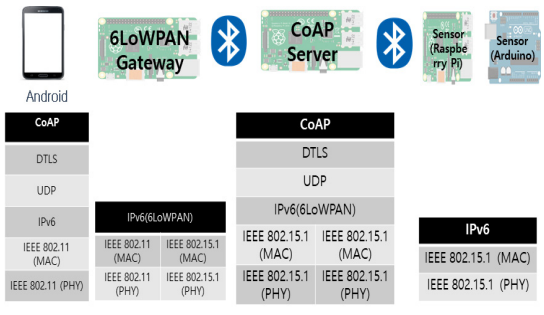


그림 2 프로토콜 스텍
Fig. 2. Protocol Stack

### 2. 통합 시스템 세부 기능

#### 가. CoAP 클라이언트

클라이언트는 스마트폰 어플리케이션을 통해 집 내부 및 외부의 다양한 센서 정보를 획득하고 연결된 사물을 동작시킬 수 있다. 사용된 센서는 온도, 습도, 먼지, 화재 감지 센서이며 먼지 농도가 높을 경우 자동적으로 환풍기를 작동시키거나 화재가 감지될 경우 부저를 울리는 등 몇몇 지능적인 기능이 구현되어 있다. 메시지 전송에는 DTLS 보안이 적용된 CoAP 프로토콜이 사용되며, 구현에는 JAVA 기반의 CoAP 프레임워크인 Californium을 사용하였다. 또한 성능 분석을 위해 HTTP, CoAP, CoAP/DTLS를 모두 지원하도록 하였다.

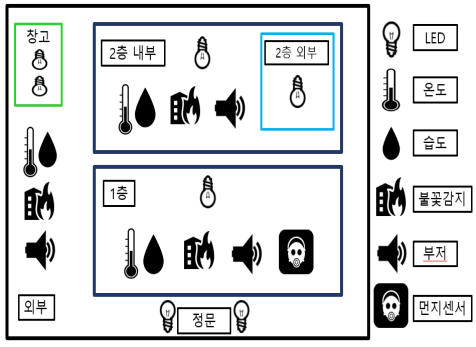


그림 3. 홈 네트워크 시스템 모델 구성
Fig. 3. Home Network System model

CoAP 클라이언트는 WiFi로 통신하기 때문에 블루투스 통신이 이루어지는 서버와 직접 통신할 수 없고 게이트웨이를 거쳐야 한다. 실제로 클라이언트는 서버의 주소만을 가지고 통신하기 때문에 게이트웨이의 역할이 상당히 중요하다.

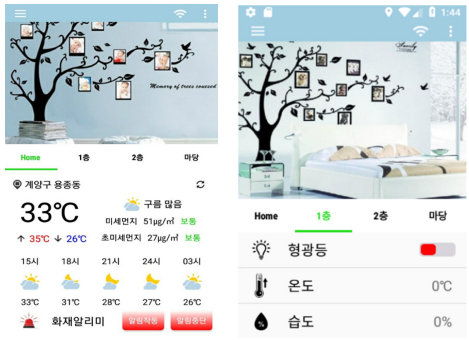


그림 4. 안드로이드 앱 화면
Fig. 4. Android App Screen

### 나. CoAP 서버

CoAP 서버는 클라이언트가 요청한 센서값을 획득하여 응답하는 기능을 수행한다. 요청 URI를 통해 센서의 위치 및 종류를 파악하고, 클라이언트와 합의된 프로토콜을 통해 사물을 동작시킨다. 클라이언트와 마찬가지로 DTLS 보안이 적용된 CoAP 메시지 프로토콜이 사용된다. 서버는 라즈베리파이3에서 동작하며, 서버 겸 센서노드 역할을 한다. 1층, 2층, 외부에 각각 하나씩 총 3대의 라즈베리파이가 사용되었다. 서버는 6LoWPAN/BLE를 통해 게이트웨이와 연결된다. 센서는 라즈베리파이와 직접 연결하여 파이썬 GPIO 라이브러리를 통해 구현한 프로그램으로 센서값을 읽도록 하였고, 얻은 정보는 Californium 프레임워크로 제작한 CoAP/DTLS서버에서 관리한다.

### 다. 6LoWPAN 게이트웨이

게이트웨이는 라즈베리파이 3에서 동작하며, 내장되어 있는 블루투스 4.1과 라즈비안 커널 4.9.y버전의 bluetooth_6lowpan 모듈을 사용하여 서버와 6LoWPAN 연결하였다.

6LoWPAN 게이트웨이는 다음과 같은 역할을 한다.

① 사용자의 WiFi망과 서버의 6LoWPAN망을 연결

② 클라이언트와 서버에 IPv6 주소를 할당

게이트웨이는 서버의 블루투스 인터페이스에 2003::/64의 주소를, 클라이언트에는 2005::/64의 주소를 할당하도록 하였다.

이 게이트웨이가 양쪽 네트워크의 Default Gateway가 되어 라우팅을 수행한다. 이 두 가지 기능을 수행시키기 위해 RADVD(Router Advertise Daemon)라는 프로그램을 사용하였다. 이 프로그램은 백그라운드에서 동작하면서 같은 네트워크에 연결된 디바이스들에 IP를 할당하고 라우팅 테이블을 자동으로 설정 해 주는 기능을 가지고 있다. 게이트웨이는 wlan0 인터페이스에 2003::1/64 주소를, bt0(6LoWPAN over BLE) 인터페이스에는 2005::1/64 주소를 가짐으로써 사용자의 WiFi와 서버의 BLE 6LoWPAN 네트워크 망 사이에서 라우터와 같이 동작하게 된다.

### 라. 보안 모듈

보안 모듈은 클라이언트와 서버의 CoAP 응용 계층 바로 아래에서 동작한다. DTLS 보안 기능을 수행하는 이 모듈은 상위 계층에서 내려오는 CoAP 메시지는 암호화 및 Encapsulation하여 네트워크로 내보내고, 반대로 네트워크에서 들어오는 메시지는 Decapsulation 및 복호화하여 상위계층으로 넘기는 작업을 수행한다. 본 시스템에서 동작되는 암호화 프로토콜(CipherSuite)은 TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8이다. 서버 및 클라이언트의 인증서는 Java의 Keytool 유틸리티를 사용하여 생성하였으며, root(Self Signed) → ca → server/client의 2단계 체인으로 구성된다.

CoAP 클라이언트는 CoAP 요청 메시지를 전송하기 위해, 우선 왕복 3회의 DTLS Handshaking 과정이 진행되며 세부 과정은 그림 3과 같이 동작한다.

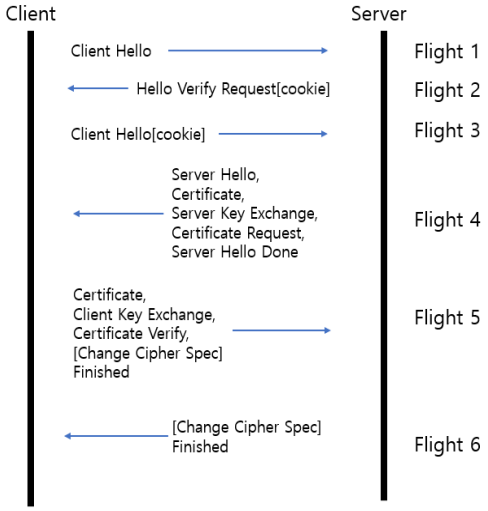


**그림 5. 전체 핸드쉐이크 과정**
**Fig. 5. Message Flights for Full Handshake**

최초 CoAP 클라이언트는 서버에 ClientHello 메시지를 전송함으로써 핸드쉐이크의 시작을 알린다. 여기에는 버전, 세션, Cipher Suite 등에 대한 정보가 포함된다.

CoAP 서버는 Client Hello에 대한 응답으로 Hello Verify Request 메시지를 보낸다. 여기에는 쿠키가 포함되어 있으며, CoAP 클라이언트는 CoAP서버로부터 받은 쿠키를 기존에 사용했던 ClientHello 메시지에 그대로 포함시켜 재전송하여야 한다.

이후 서버는 응답으로 ServerHello, Certificate, ServerKeyExchange, CertificateRequest, ServerHelloDone을 하나의 Flight에 담아 전송한다. 이를 수신받은 클라이언트는 서버의 인증서 및 공개 키의 무결성을 검사하

고 임의의 pre-master secret을 생성하는 과정을 진행한다.

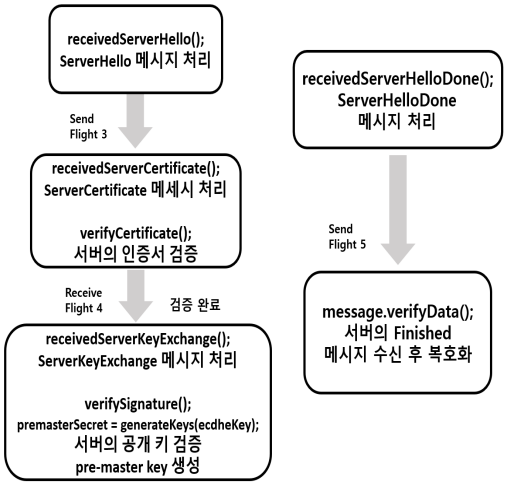


**그림 6. 보안인증 과정**
**Fig. 6. Security certification process**

그림 6의 과정을 거쳐 서버를 신뢰할 수 있음이 확인되면 CoAP 클라이언트는 생성된 pre-master key를 ClientKeyExchange에 포함시켜 서버에 전송한다. 이제 서버와 클라이언트는 pre-master key를 이용하여 동일한 master key를 생성하고, 세션이 유지되는 동안에는 메시지를 master key를 이용한 대칭 키 알고리즘으로 암호화하여 전송한다.

보안 모듈은 JAVA에서 제공하는 전송 계층 통신 인터페이스인 Connector를 dtlsConnector라는 별도의 객체로 구현시킨 상태로 존재한다. CoAP 서버와 CoAP 클라이언트는 CoAP메세지를 CoapEndpoint라는 객체의 인스턴트로써 캡슐화하고 이를 dtlsConnector에 넘기면, 커넥터는 DTLS 헤더와 메시지를 Flight에 담아 전송한다. 또한 클라이언트와 서버는 각각 추상클래스인 Handshaker를 상속받은 ClientHandshaker, ServerHandshaker를 생성하여 핸드쉐이크 과정을 진행한다. CoAP클라이언트에서 동작되는 각 Flight별 보안모듈의 핵심 함수는 다음과 같다.

```
handshaker = new ClientHandshaker()

// 클라이언트의 Handshaker 객체 생성. 이후 아래의 startHandshake 함수를 호출한다.
```

```
handshaker.startHandshake()
        clientHello = new ClientHello(...);
        Flight.add(clienthello)
        Send(Flight)

// handshaker의 startHandshake 함수는 ClientHello 메시지를 생성하여 서버에 전송하며, 핸드쉐이크의 시작을 알리는 방아쇠 역할을 한다. 객체 생성 시 파라미터로 프로토콜 버전, 랜덤값, 클라이언트의 인증서 타입 등의 정보를 넘겨준다.
```

Flight 1은 Handshaker 생성 및 핸드쉐이크 시작하는 과정으로 ClientHello메시지를 생성하여 전송한다.

```
handshaker.receivedHelloVerifyRequest()
        clientHello.setCookie()
        Send(clientHello)

// 서버로부터 HelloVerifyRequest 메시지를 받으면 clientHello 객체의 setCookie 함수를 실행하여 비어있는 clientHello의 cookie 필드를 채운다.
```

Flight 3에서는 ClientHello 메시지에 쿠키를 포함시켜 재전송한다.

```
// 서버의 Flight4를 처리하기 위한 다음의 4가지 함수가 존재한다.

receivedServerHello()
        session.setSessionIdentifier()
        session.setCipherSuite()
        session.setCompressionMethod()
// ServerHello 메시지의 처리. 메시지에 포함된 정보를 통해 세션 식별자, 합의된 Cipher Suite, 압축 방법을 설정한다.

receivedServerCertificate()
        verifyCertificate(serverCertificate)
        serverPublicKey = getPublicKey()
// Certificate 메시지의 처리. 서버의 인증서를 검증하고, 이상 없으면 인증서에 포함된 서버의 공개 키를 저장한다. 인증서 검증을 위해 클라이언트가 보유한 Trust Store가 사용된다.

receivedServerKeyExchange()
        premasterkey = ecdhe.getSecret()
        verifySignature(serverPublicKey)
// KeyExchange 메시지의 처리. pre-master 키를 생성하며, 서버의 키를 검증한다.

receivedServerHelloDone()
        createCertificateMessage(Flight)
        Flight.add(new ECDHClientKeyExchan...)
        Flight.add(new CertificateVerify())
        Flight.add(new ChangeCipherSpecMsg())
        Flight.add(new Finished())
        send(Flight)
// HelloDone 메시지를 받으면 Flight 5를 위한 준비를 시작한다. Certificate, ClientKeyExchange, CertificateVerify, ChangeCipherSpec, Finished를 순서대로 Flight에 담아 한번에 전송한다.
```

Flight 5는 서버가 송신한 Flight 4 메시지를 처리한다. 통신 스펙 설정, 인증서 검증 및 pre-master key를 생성하고 Finished 메시지를 전송하는 클라이언트의 마지막 Flight이다.

CoAP 서버의 동작과정 및 보안모듈 함수는 다음과 같다.

```
handshaker = new ServerHandshaker()
handshaker.startHandshake()
        Flight.add(new HelloRequest())
        Send(Flight)

// 클라이언트와 동일하게 Handshaker 객체의 생성과 startHandshake 함수의 호출을 통해 핸드쉐이크가 시작된다.
```

Flight 2에서는 HelloVerifyRequest 메시지를 송신한다.

```
receivedClientHello()
        Flight.add(new ServerHello())
        Flight.add(new CertificateMessage())
        Flight.add(new ECDHServerKeyExchange())
        Flight.add(new CertificateRequest())
        Flight.add(new ServerHelloDone())
        Send(Flight)

// 두 번째 ClientHello 메시지의 처리 함수이다. 인증서 및 키교환 메시지를 포함하여 전송한다.
```

Flight 4에서는 클라이언트가 보낸 쿠키 검증 완료 시 서버의 인증서, 공개 키, 인증요청서 등의 메시지를 전송한다.

```
receivedClientCertificate()
        verifyCertificate(clientCertificate)
        clientPublicKey = clientCertificate.getKey()..

// 클라이언트가 보낸 Flight 5의 첫 메시지인 Certificate를 처리하는 함수이다. 인증서가 올바른 ca로부터 서명을 받았는지 확인하고 인증서에 포함된 클라이언트의 공개 키를 저장한다.
```

```
premasterSecret = receivecClientKeyExchange()

// 클라이언트가 보낸 pre-master키를 저장한다.
```

```
receivedCertificateVerify()
        verifySignature(clientPublicKey)

// CertificateVerify 메시지를 처리하는 함수이다. 이 함수 내에서는 verifySignature 함수가 실행되어 클라이언트의 공개 키를 검증한다.
```

```
receivedClientFinished()
        message.verifyData()
        Flight.add(new ChangeCipherSpecMsg())
        Flight.add(new Finished())
        send(Flight)

// 마지막으로 pre-master 키를 사용하여 클라이언트의 메시지를 확인하고 이상 없을 시 ChangeCipherSpec 메시지를 생성하여 전달한다.
```

Flight 6은 클라이언트의 인증서를 검증하고 Finished 메시지를 전송하는 서버의 마지막 단계이다.

### 3. CoAP/DTLS GET 메시지 동작 과정

CoAP 클라이언트는 CoAP GET메시지를 이용하여 온도 센서로 값을 받아오며 아래의 과정에 따라 동작한다.

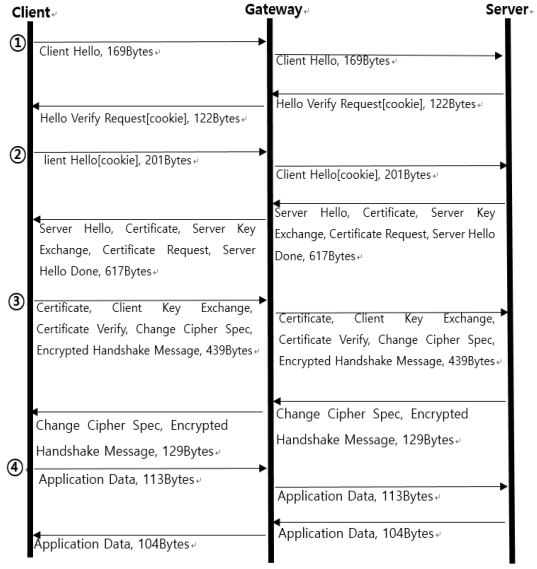


**그림 7. CoAP/DTLS GET 메시지 과정**
**Fig. 7. Process of CoAP/DTLS GET message**

CoAP 클라이언트가 온도 정보를 확인하고자 어플리케이션을 실행하면 우선 DTLS 핸드쉐이킹 과정이 진행된다. 왕복 세 번의 패킷 교환을 통해 핸드쉐이크 과정이 완료되면 적절한 CoAP 요청 메시지를 제작, 서버와 합의한 암호화 알고리즘을 사용하여 전송하면 서버는 이를 확인하여 응답 메시지를 보낸다.

## Ⅳ. 실험 및 결과

본 논문에서는 개발한 시스템을 활용하여 DTLS 동작 절차에 따른 데이터 전송 시간을 측정하기 위하여 사용자가 1층 서버의 온도센서 값을 핸드쉐이크 후 GET 메소드를 통해 받아오는 과정의 소요 시간을 분석하였다. 비교를 위해 CoAP와 DTLS 프로토콜을 함께 사용한 경우, CoAP 프로토콜만 사용한 경우, HTTP 프로토콜을 사용한 경우로 나누었다. 우선 CoAP클라이언트가 센서를 통해 온도 값을 요청한 뒤 확인이 완료될 때 까지 소요되는 시간을 측정하였으며 세부 결과는 그림 8에서 보여준다.

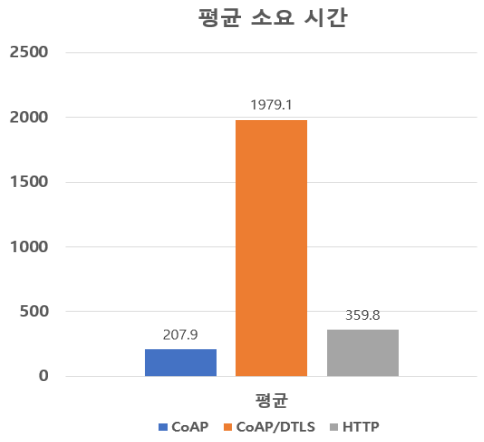


그림 8. CoAP, HTTP, DTLS의 전송 시간(ms)
Fig. 8. Transmission Times of CoAP and HTTP and DTLS (ms)

CoAP는 HTTP에 비해 58% 정도 속도가 빨랐다. UDP기반이기 때문에 TCP 상에서 동작하는 HTTP와는 달리 쓰리-웨이 핸드쉐이킹이 존재하지 않으며, 응용 계층 메시지(Payload) 자체도 경량화 된 결과이다. 그러나 DTLS를 적용하였을 경우 평균 1979.1ms로 사용자 입장에서 거의 2초에 가까운 대기시간이 필요했는데, 이는 CoAP 만을 적용하였을 경우와 9.5배 정도 차이나는 결과이다. 또한 HTTP와 비교했을 때도 5.5배 느렸다.

시간 지연의 원인 분석을 위해 시간 측정에 사용되었던 DTLS 패킷을 분석하였다. 클라이언트와 게이트웨이 사이, 게이트웨이와 서버 사이에서 전송에 소요된 시간과 클라이언트, 서버에서 패킷을 받고 응답하기까지의 처리 시간을 구간별로 나누어 합산하였다.

표 2. 구간 별 소요시간(ms)
Table 2. Interval operation time(ms)

| 구간 / 시간 | 클라이언트와 게이트웨이 | 게이트웨이와 서버 | 서버와 센서모듈 | 전체 시간 |
|---|---|---|---|---|
| 평균 | 467.9 | 1167.4 | 343.8 | 1979.1 |
| 비율 | 23.6% | 58.1% | 17.1% | 100% |

분석 결과, 게이트웨이와 서버 사이의 전송에 소요되는 평균 시간이 1167.4ms로 전체 평균의 합의 58.1%를 차지하여 가장 높았다.

다음으로 한 차례의 CoAP 응답이 완료될 때 까지 각 구간에서의 각 flight별 전송 시간을 측정하였으며 100회 정도 측정하여 평균을 구하였다.

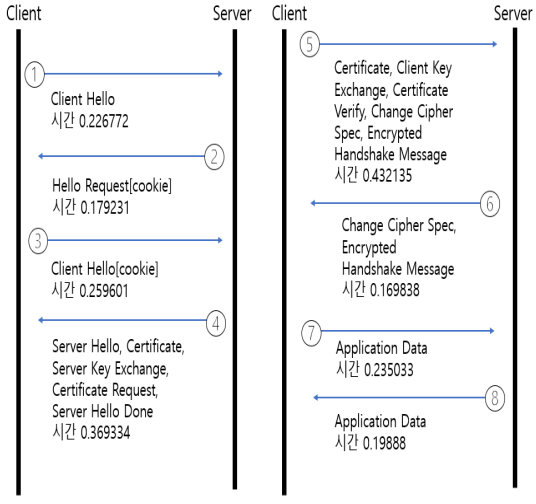


그림 9. 구간 별 소요시간(s)
Fig. 9. Interval operation time(s)

위의 결과를 바탕으로 분석 가능한 점은 CoAP 서버와 게이트웨이 사이의 6LoWPAN 통신 속도가 패킷의 크기가 늘어남에 따라 큰 폭으로 느려진다는 점이다. ClientHello와 HelloRequest 메시지의 경우 200byte 이하의 크기에 0.2초의 시간이 소요되는 반면, ServerHello나 ClientCertificate 메시지의 경우 600byte 정도의 크기에 0.4초가량, 즉 2배의 시간이 소요된다. 이는 6LoWPAN/BLE 구간의 전송효율이 낮다는 것을 보여준다. 패킷 분석 결과 CipherSuite는 RSA, ECDHE, AES128과 같은 암호화 과정 방식이 그대로 사용되었다. 이로 인해 Handshaking 과정에서 많은 부하가 발생하였

을 것으로 보여 지며, BLE 4.1의 MTU가 불과 23옥텟 이라는 점을 고려하면 심한 단편화가 발생하였을 것으로 판단된다.

## V. 결 론

본 논문에서는 DTLS 보안기술을 적용하여 모바일 환경에서 CoAP/6LoWPAN 기반의 홈 네트워크 시스템을 동작시키고 제어하는 시스템을 구현하였다. 또한 구현된 시스템을 이용하여 DTLS 프로토콜의 동작 과정에 따른 데이터 전송 시간을 측정하였다. 그 결과, 센서의 측정값을 받아오는 과정에 보안 프로토콜을 적용하였더니 평균 약 2초 정도의 시간이 소요되었다. 보안 프로토콜의 각 절차 별 구간별로 소요 시간을 분석하였다. 각 구간별로는 CoAP서버와 게이트웨이 사이에 소요되는 시간이 전체 시간 대비 58.1%로 가장 많은 시간을 차지하였다.

또한 보안모듈을 적용하였을 때 소요되는 시간과 그렇지 않은 경우를 비교하기 위하여, CoAP에 보안 프로토콜을 적용하지 않았을 때, CoAP에 DTLS 보안프로토콜을 적용하였을 때, HTTP를 이용할 때 3가지의 경우로 나누어서 센서 정보를 가져오는데 소요되는 시간을 측정하여 비교 분석하였다. DTLS 보안 모듈을 적용하였을 때 소요되는 시간은 보안 모듈을 적용하지 않았을 때 대비 무려 약 9.5배 이상의 소요시간을 나타냈다.

DTLS 프로토콜은 TCP환경의 보안 프로토콜인 TLS를 기반으로 하는 UDP 보안 프로토콜이다. 그래서 서로 유사한 점이 많고 그 중에는 작지 않은 크기의 인증서 및 키 알고리즘도 포함된다. DTLS 프로토콜은 UDP기반의 IoT 환경에서 사용 가능하기는 하지만 본 논문의 결과에서 제시된 것처럼 많은 소요시간을 보인다. 특히 핸드쉐이킹 과정에서 많은 시간을 소요하는 것으로 나타나며 데이터를 전송할 때 암호화 데이터를 응용 데이터에 같이 전송하는 방안 등 암호화 과정을 줄일 수 있는 다양한 방안이 향후 연구과제로 제시된다.

## References

[1] SooHyun Ahn, Kwangjo Kim “A Method of lightweight DTLS protocol for IoT”, Graduate school of information security, KAIST.

[2] HanByul Yeon, A Study on End-to-End Security from the Viewpoint of IoT Network Architecture, http://www.riss.kr/link?id=T14477869

[3] IETF-RFC6347 Datagram Transport Layer Security Version 1.2, https://tools.ietf.org/html/rfc6347

[4] IETF-RFC7252 The Constrained Application Protocol (CoAP), https://tools.ietf.org/html/rfc7252

[5] IETF-RFC6347 IPv6 over BLUETOOTH(R) Low Energy, https://trac.tools.ietf.org/html/rfc7668

[6] Sun-Jin Oh, Design of the Smart Application based on IoT
DOI: https://doi.org/10.7236/JIIBC.2017.17.5.151

[7] Lei Hang, Wenquan Jin, Do-Hyeun Kim, A Design and Implementation for Registration Service of IoT Embedded Node using CoAP Protocol-based Resource Directory in Mobile Internet Environments
DOI: http://dx.doi.org/10.7236/JIIBC.2016.16.1.147

[8] Minzheong Song, A Study on Business Types of IoT-based Smarthome: Based on the Theory of Platform Typology
DOI: http://dx.doi.org/10.7236/JIIBC.2016.16.2.27

[9] Hyoung-Ro Lee, Chi-Ho Lin, Design and Implementation of Arduino-based Efficient Home Security Monitoring System
DOI: http://dx.doi.org/10.7236/JIIBC.2016.16.2.49

### 저자 소개

**김 연 수(준회원)**

• 2012년 3월 ~ 현재 : 한국산업기술대학교 컴퓨터공학부 학부생

<주 관심분야> : IoT, 보안, 네트워크

**김 기 태(준회원)**

- 2012년 3월 ~ 현재 : 한국산업기술대학교 컴퓨터공학부 학부생

<주 관심분야> : IoT, 보안, 네트워크

**이 보 경(정회원)**

- 1987년 2월 : 고려대학교 수학과 (이학사)
- 1995년 12월 : University of Birmingham Department of Computer Science
- 2000년 8월 : 고려대학교 컴퓨터학과 (이학박사)
- 2003년 3월 ~ 2005년 2월 : Telecom Paris PostDoc
- 1987년 1월 ~ 1998년 9월 : 데이콤 연구소
- 2001년 3월 ~ 현재 : 한국산업기술대학교 컴퓨터공학부 교수

<주 관심분야> : 사물인터넷, 모바일, 홈 네트워크 시스템, LoRa, 보안

※ 본 논문은 한국연구재단 중견연구자지원사업(NRF-2017R1A2B1005577)의 연구결과로 수행되었음 (교신저자 이보경)