논문 15-40-08-16 The Journal of Korean Institute of Communications and Information Sciences '15-08 Vol.40 No.08
http://dx.doi.org/10.7840/kics.2015.40.8.1588

사물인터넷에서 경량화 장치 간 DTLS 세션 설정 시 에너지 소비량 분석

권 혁 진◆, 강 남 희°

Analysis on Energy Consumption Required for Building DTLS Session Between Lightweight Devices in Internet of Things

Hyeokjin Kwon◆, Namhi Kang°

요 약

사물인터넷에서는 센서와 같은 자원이 제한된 장치들이 인터넷을 경유하여 통신하고 정보를 공유할 수 있다. 이러한 경량화 장치가 응용계층에서 데이터를 전송할 수 있도록 IETF에서는 전송계층 UDP를 이용하는 CoAP을 표준으로 제정하였으며, 보안을 위해 DTLS를 사용할 것을 권고하고 있다. 그러나 DTLS는 데이터 손실, 단편화, 리오더링 그리고 리플레이 공격 문제를 해결하기 위해 부가적인 보상 기술이 추가되었다. 이로 인해 DTLS는 TLS보다 성능이 저하된다. 경량화 장치는 배터리로 구성된 경우, 배터리 효율의 극대화를 위해 저전력으로도 동작될 수 있는 보안 설계 및 구현 역시 반드시 고려되어야 한다. 따라서 본 논문에서는 에너지 소비량 관점에서 DTLS의 성능에 대해 논의하고자 한다. 성능 분석을 위해 Cooja 시뮬레이터를 이용하여 센서 장치와 IEEE 802.15.4 기반의 네트워크 실험 환경을 구축하였다. 실험 환경을 통해 DTLS 통신을 하고자 하는 서버와 클라이언트의 에너지 소비량을 각각 측정하였다. 또한 DTLS의 핸드쉐이크 Flight 별 에너지 소모량, 처리 시간 및 수신 시간, 전송 데이터 크기를 측정하여 코드 크기, 암호 프리미티브 그리고 단편화 관점에서 분석된 결과를 함께 기술하였다.

Key Words : DTLS, Handshake, Energy consumption, Internet of Things, Cooja simulator

ABSTRACT

In the Internet of Things (IoT), resource-constrained devices such as sensors are capable of communicating and exchanging data over the Internet. The IETF standard group has specified an application protocol CoAP, which uses UDP as a transport protocol, allows such a lightweight device to transmit data. Also, the IETF recommended the DTLS binding for securing CoAP. However, additional features should be added to the DTLS protocol to resolve several problems such as packet loss, reordering, fragmentation and replay attack. Consequently, performance of DTLS is worse than TLS. It is highly required for lightweight devices powered by small battery to design and implement a security protocol in an energy efficient manner. This paper thus discusses about DTLS performance in the perspective of energy consumption. To analyze the performance, we implemented IEEE 802.15.4 based test network consisting of constrained sensor devices in the Cooja simulator. We measured energy consumptions required for each of DTLS client and server in the test network. This paper

※ 본 연구는 2015년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No.2014R1A1A2056961); 또한 본 연구는 미래창조과학부 및 정보통신기술진흥센터의 대학ICT연구센터육성 지원사업의 연구결과로 수행되었음. (IITP-2015-H8501-15-1008).

◆ First Author : Department of Digital Media, Duksung Women's University, hyeok@duksung.ac.kr, 학생회원
° Corresponding Author : Department of Digital Media, Duksung Women's University, kang@duksung.ac.kr, 정회원
논문번호 : KICS2015-05-159, Received May 29, 2015; Revised July 9, 2015; Accepted July 9, 2015

compares the energy consumption and amount of transmitted data of each flight of DTLS handshake, and the processing and receiving time. We present the analyzed results with regard to code size, cipher primitive and fragmentation as well.

Ⅰ. 서 론

최근 전 세계적으로 사물인터넷(IoT: Internet of Things) 기술이 주목 받고 있다. 사물인터넷에서 각 사물은 네트워크에 연결되어 사람과 사물 또는 사물과 사물 간에 상호 소통하고 상황인식 기반의 지능형 서비스를 제공하게 된다. 이를 위해 다양한 형태의 사물은 필요 시 서버와 클라이언트의 기능을 병행하여 수행할 수 있어야 한다. 따라서 일반적인 서버-클라이언트 모델이 내포하고 있는 보안 위협과 이에 대응할 수 있는 보안 기술들은 사물인터넷 환경에서도 여전히 고려할 사항이 된다.

국제 표준기구인 IETF에서는 자원이 제한된 환경에서 클라이언트/서버 모델을 고려한 응용 계층 프로토콜로 CoAP(Constrained Application Protocol)을 표준화하였다[1]. CoAP은 HTTP와 유사하지만, 전송의 효율을 고려하여 TCP 대신 UDP를 전송 프로토콜로 사용한다. 보안 관점에서는 HTTP가 클라이언트와 서버 간 안전한 데이터 전송을 위해 TLS(Transport Layer Security)를 사용하듯, CoAP에서는 TLS의 UDP 기반 버전인 DTLS(Datagram TLS)를 사용하도록 권고하고 있다[2]. 안전한 전자상거래를 위해서 넷스케이프 사에 의해 1994년 최초로 개발된 SSL을 토대로 IETF는 TCP 전송 계층에서 범용적으로 사용이 가능한 TLS를 1999년 인터넷 표준으로 제정하였다. 21세기가 되면서 인터넷의 급격한 성장과 함께 UDP 전송계층 프로토콜을 이용한 실시간 데이터 처리가 증가함에 따라 이를 보호하기 위한 DTLS가 2006년 IETF에 의해 추가적으로 표준으로 제정되었다.

자원이 제한적인 전송환경 및 소형 장치의 능력을 고려할 경우 적용할 수 있는 보안 기술은 다양한 측면에서 경량화 되어야 한다. 현재 제안되고 있는 많은 연구들은 소형 장치의 성능(배터리, CPU 및 메모리) 자원을 우선 고려하고 있다[15,16]. 그러나 연산처리 능력과 메모리가 보장되더라도 전원장치가 배터리로 구성된 경우, 배터리 효율의 극대화를 위해 저전력으로도 동작될 수 있는 보안 설계 및 구현 역시 반드시 고려되어야 한다[14].

본 논문에서는 국제 표준 규약인 CoAP에서 권고하고 있는 DTLS 보안 프로토콜을 에너지 사용량을 기반으로 분석한다. 특히, DTLS에서 안전한 소켓을 개설하기 위해 반드시 필요한 핸드쉐이크(handshake) 과정에서 에너지 사용 문제점에 대해 논의하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문의 분석을 위해 필요한 관련 기술을 설명한다. 3장에서는 DTLS 프로토콜의 에너지 소비량을 분석하기 위해 수행한 실험 환경과 결과를 기술한다. 특히, 코드 크기, 암호 프리미티브 그리고 단편화 관점에서 분석된 결과를 기술한다. 이어서 4장에서 결론을 맺는다.

Ⅱ. 관련연구

2.1 DTLS (Datagram TLS)

DTLS는 보안 관점에서 TLS가 가진 기능 요구사항과 동일하다. 암호화된 통신으로 비밀성이 보장되며 메시지마다 무결성 체크를 통해 신뢰성이 보장된다. 그러나 TCP기능에 의존하여 구현된 TLS에서 TCP를 UDP로 대체할 경우, 전송 계층에서 UDP가 가진 데이터 손실, 단편화(Fragmentation), 리오더링 그리고 리플레이와 같은 문제가 그대로 계승된다. 이로 인해 DTLS는 TLS에서 부가적인 보상 기술이 추가되었다. 이것은 DTLS가 세션을 개설하는 과정에서 TLS보다 더욱 많은 데이터 전송과 처리를 해야 하는 것을 의미한다. 즉, DTLS는 TCP보다 경량화된 UDP 프로토콜을 사용함에도 불구하고 TLS보다 성능이 저하된다.

표 1은 세션을 개설하는 과정에서 TLS와 DTLS간의 전송데이터 차이를 나타낸다[3]. 표 1에 표기된 Packets 컬럼은 전송된 패킷의 총 개수를 의미하며, Bytes는 패킷들의 총 크기를 의미한다. 일반적으로 DTLS 핸드쉐이크 과정에서 서버는 3회의 패킷 전송을 한다.그러나 표 1에서는 인증서 크기로 인해 단편화가 발생되어 서버의 측정값은 4로 표기되었다. 결과적으로 DTLS의 전송량이 15%이상 많은 것을 볼 수 있다.

DTLS는 주체간 안전한 소켓을 연결하기 위해 2개의 구분된 계층으로 구성된다[2]. 1) 주체 간 상호인증 및 키 알고리즘을 교환하기 위한 핸드쉐이크 계층과 2) 핸드쉐이크 전송 데이터를 포함하여 암호화된 데이터를 전송할 때 사용하는 레코드 계층이 주요 구성 요소이다. 그림 1은 클라이언트와 서버가 핸드쉐이크 계

표 1. DTLS와 TLS 간의 패킷 수와 전송량 비교 (PMTU: 1500, 인증서 크기: 1671 bytes)
Table 1. Comparison DTLS with TLS in view of the number of packet and the amount of transmitted data (PMTU: 1500, Certificate size: 1671 bytes)

	DTLS		TLS	
	Packets	Bytes	Packets	Bytes
Client	3	446	2	228
Server	4	2313	3	2105
Total	7	2759	5	2333

층 시작부터 암호화된 어플리케이션 데이터를 주고받기까지의 과정이다.

핸드쉐이크 과정은 클라이언트에 의한 Flight 1로 시작하여 서버가 응답하는 Flight 6까지로 구성된다. 클라이언트에서 Flight 6을 수신하고 검증에 문제가 없다면 이 후부터 암호화된 어플리케이션 데이터 전송이 시작된다[2]. Flight 2와 Flight 3은 UDP 특성으로 인해 발생할 수 있는 서비스거부공격을 방지하기 위해 인터넷 키 교환 및 관리 표준 프로토콜 Photuris, IKE에서 Cookie기술을 차용하였다. 이는 DTLS에만 존재하는 Flight이며 TLS에서는 존재하지 않는다.

핸드쉐이크 과정의 주된 목적은 어플리케이션 데이터를 암복호화할 때 사용할 무작위 세션키를 공유하기 위한 것이다. 이를 위해서 핸드쉐이크 과정에서 크게 2가지 분류의 알고리즘 교환이 발생한다. 세션키 생성방법과 사용법에 대한 알고리즘 목록을 교환하여 세션키를 생성하는 ChangeCipherSpec과 그 정보를

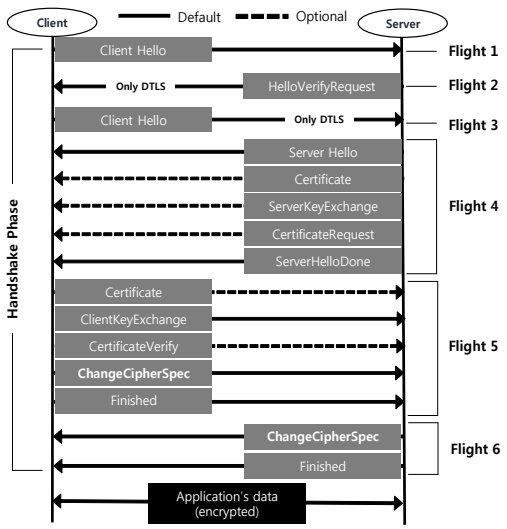

그림 1. DTLS 핸드쉐이크 과정
Fig. 1. DTLS Handshake procedure

안전하게 주고받기 위한 키 교환 알고리즘을 교환하는 KeyExchange 과정이다. 그림 1에서 일부 Flight에 실선으로 표시된 것은 키 교환 알고리즘이 비대칭키가 사용되는 경우에만 실행된다. 키 교환 알고리즘으로 대칭키를 사용할 경우에는 실행되지 않으며 성능 향상에 큰 이점이 있다.

키 교환 알고리즘과 CipherSpec의 알고리즘은 CipherSuite라는 이름으로 IANA에 의해 관리된다[4]. 현재 약 300여개 이상의 알고리즘이 존재하는데 표준에서는 이 중에서 TLS_RSA_WITH_AES_128_CBC_SHA를 기본적으로 반드시 구현하도록 표준에 명시되어 있다[5].

핸드쉐이크 과정이 정상적으로 수행되면 이 과정에서 생성된 세션키를 이용해서만 어플리케이션 데이터를 전송할 수 있게 된다.

2.2 IEEE 802.15.4와 6LoWPAN

OSI 7계층 모델에서 링크 계층에 해당하는 IEEE 802.15.4 무선 표준은 저전력 개인용 네트워크(low-power personal area networks)를 위해 개발되었다[6]. Zigbee 라는 이름으로 불리는 이 표준기술은 WiFi로 잘 알려진 802.11 대비 1%의 전력 소비량을 가지며 250 kbit/s의 대역폭을 가진다. 802.15.4 스펙에서는 저전력 무선 환경에서 발생 가능한 데이터 손실률을 고려하여 MTU (Maximum Transmission Unit) 127바이트의 특징을 가진다. 이는 802.15.4에서 127바이트보다 큰 데이터 전송 시 메시지 단편화를 발생 시킨다. 이 특성으로 인해 헤더 길이가 40바이트인 IPv6 주소 인터넷 체계를 그대로 사용하면 어플리케이션 계층의 데이터가 포함되는 페이로드크기가 매우 작아지는 문제가 발생한다. 예를 들어, UDP 프로토콜을 사용할 경우에 각 계층마다 추가되는 헤더정보로 인해 한 번에 전송할 수 있는 데이터의 최대 크기는 33바이트가 된다. 802.15.4의 이 문제를 해결하기 위해 IETF 표준화 워킹그룹에서는 6LoWPAN (IPv6 over Low-Power Wireless Personal Area Networks) 프로토콜을 표준으로 제정하였다[7].

6LoWPAN은 802.15.4와 IPv6 전송 계층 중간에 존재하는 어댑테이션 계층으로 802.15.4가 IP주소 기반의 인터넷에 효율적으로 접속될 수 있도록 주소할당, 헤더 압축과 같은 기술을 제시한다. 헤더 압축은 네트워크 구성 환경에 따라 압축률이 달라질 수 있다. 이로 인해 802.15.4에서의 6LoWPAN 환경에서 패킷 페이로드 크기는 가변적으로 변하게 된다. 페이로드보다 큰 데이터가 전송되어야 할 때에는 단편화가 발생

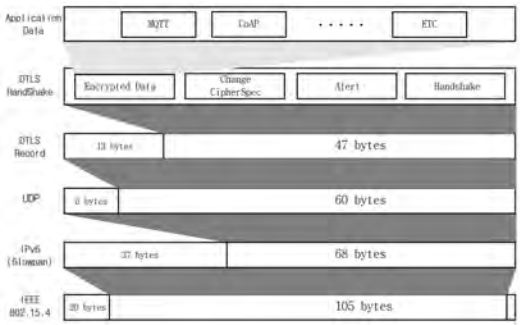

그림 2. 계층별 헤더 및 페이로드 크기
Fig. 2. Length of headers and payloads in OSI Layer

하며 경량화 장치에서 단편화로 인한 송수신 지연(Latency)과 재조립(Reassembly) 과정에서 발생하는 처리량은 배터리 성능에 큰 문제를 야기한다. 그림 2는 각 계층별 헤더와 페이로드의 크기를 도식화한 것이다.

2.3 Contiki OS

Contiki OS는 경량화된 IoT 센서 장치를 위한 오픈소스 운영체제이다[8]. ST마이크로일렉트로닉스, 아트멜, 텍사스인스트루먼트와 같은 제조업체가 생산하는 저전력 MCU가 내장된 임베디드 장치에 탑재될 수 있다. C언어로 개발된 Contiki는 표준 C 라이브러리를 이용한 어플리케이션 개발이 가능하다. 특정 하드웨어에 종속적이지 않은 오픈 소스 라이브러리를 재사용할 수 있는 장점이 있다. 또한 운영체제 크기가 약 40킬로바이트 정도임에도 불구하고 LLN(Low power and Lossy Networks)을 위한 6LoWPAN, RPL, IPv6, TCP, UDP, CoAP, MQTT 같이 IoT를 위한 다양한 프로토콜이 지원된다. 특히 에너지 소비량 측정을 위한 함수 제공과 고수준의 시뮬레이션 기능을 내장한 Cooja 네트워크 시뮬레이터가 포함되어 산업계뿐만 아니라 학계에서도 널리 사용 되고 있다.

성능이 제한된 센서 장치에 Contiki를 펌웨어로 적용하기 위해서는 1) Contiki에서 제공되는 SDK와 표준 C 라이브러리를 활용하여 코드를 작성하고 2) 장치의 MCU에 맞는 컴파일러로 빌드하여 ELF 형식의 바이너리를 얻은 후, 3) MCU 개발도구에 포함된 부트스트랩 로더 유틸리티를 이용하여 앞서 빌드된 바이너리를 롬에 업로드 과정을 거쳐야한다.

Contiki에 내장된 Cooja는 텍사스인스트루먼트사의 MSP430 시리즈 마이크로프로세서를 에뮬레이트 할 수 있는 MSPSim을 기반으로 만들어진 GUI 네트워크 시뮬레이터이다. Cooja를 이용하면 가상의 802.15.4 무선 네트워크 환경을 구성하고 미리 정의된 가상의 센서 장치에 빌드된 바이너리를 사용하여 원하는 코드를 시뮬레이션 할 수 있다. 이 때 무선 환경에서의 데이터 로스율, 수신 가능 거리 같은 설정뿐만 아니라 다양한 종류의 스펙을 가진 실존하는 센서 장치를 선택할 수 있다. Cooja는 어플리케이션 개발과정에서 표준 출력 함수를 통해 디버깅 메시지를 확인하는 기능을 제공한다.

Cooja를 실행하는 호스트 컴퓨터에 TUN 네트워크 가상 장치와 SLIP(Serial Line Internet Protocol) 프로토콜을 결합하면 호스트 컴퓨터를 통해 시뮬레이터 내의 무선 네트워크를 인터넷에 연결시킬 수 있다. 즉, IPv4 또는 IPv6 주소 체계를 통해 시뮬레이터 내 센서 장치가 외부 인터넷의 서버에 접속하거나 반대로 외부에서 시뮬레이터 내 장치에 접속하는 것 또한 가능하다. 이 때 LLN망과 인터넷 사이 경계에서 게이트웨이 역할을 하는 장치가 반드시 필요하며 Contiki의 border-router 예제를 이용해 구현할 수 있다.

그림 3은 Cooja 시뮬레이터에서 DTLS 서버와 클라이언트를 테스트 하는 화면이다.

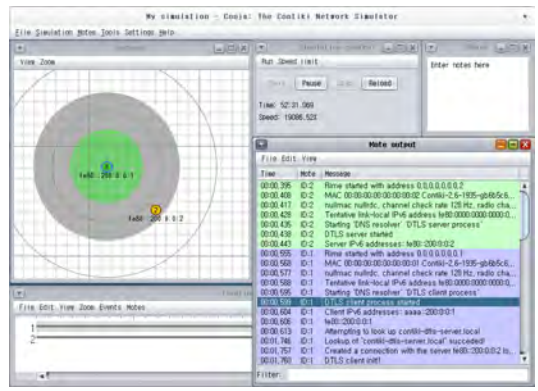

그림 3. Cooja 시뮬레이터 화면 캡처
Fig. 3. Screenshot of Cooja Simulator

2.4 에너지 소비량

Contiki에 내장된 energest_type_time() 함수는 센서 장치가 소비하는 에너지를 측정할 때 사용되는 함수이다[9]. energest_type_time() 함수는 입력받은 인자 값을 기준으로 장치가 부팅된 순간부터 함수 자신이 호출된 시점까지의 클럭 틱 시간을 구하여 반환한다.

함수 인자로 사용 가능한 여러 가지 타입 중 CPU, LPM(Low Powered Mode), TX(네트워크 전송), RX (네트워크 수신) 4가지 인자를 사용하여 구한 클럭 틱 값 4개를 모두 합산하면 센서 장치가 특정 기능을 실

```
static uint32_t last_cpu, last_lpm, last_listen, last_transmit;
static uint32_t last_idle_transmit, last_idle_listen;

// to start of flight
last_cpu = energest_type_time(ENERGEST_TYPE_CPU);
last_lpm = energest_type_time(ENERGEST_TYPE_LPM);
last_transmit = energest_type_time(ENERGEST_TYPE_TRANSMIT);
last_listen   = energest_type_time(ENERGEST_TYPE_LISTEN);

// to end of flight
PRINTF("%lu,%lu,%lu,%lu",
       energest_type_time(ENERGEST_TYPE_CPU) - last_cpu,
       energest_type_time(ENERGEST_TYPE_LPM) - last_lpm,
       energest_type_time(ENERGEST_TYPE_TRANSMIT) - last_transmit,
       energest_type_time(ENERGEST_TYPE_LISTEN)   - last_listen
);
```

그림 4. 에너지 소비량 측정 함수
Fig. 4. Measuring function for energy consumption

행하는 동안 소요된 클럭 틱 시간 정보를 구할 수 있다.

클럭 틱은 프로세스가 장치를 선점하여 사용하는 시간을 측정하기 위해 소프트웨어적으로 정의된 시간 단위이다. 예로 일반적으로 경량화 장치 실험에서 주로 사용되는 Zolteria Z1 센서 장치는 Contiki에 의해서 1초가 32,768클럭 틱으로 정의된다. 클럭 틱 정보를 이용해 에너지 소비량을 구하기 위해서는 센서 장치에 사용된 MCU를 생산하는 제조업체의 데이터시트를 반드시 확인해야 한다.

일반적으로 데이터시트에는 CPU 연산, 데이터 송수신, 대기상태에 대한 1초당 에너지 소비량이 mA(밀리암페어) 단위로 표기가 되어 있다. 이 정보를 토대로 1 클럭 틱 당 소비되는 mA를 구하고 입력 전압을 곱하면 그림 5와 같이 mW(밀리와트) 단위를 가지는 에너지 소비량이 계산된다.

energest_type_time() 함수는 에너지 소비량 측정에 있어 하드웨어 기반의 에너지 소비량 측정 기기를 이용한 값과 비교했을 때 약 94%의 정확도를 가진다[10].

- **Contiki OS 설정 확인 내역**
  - 1초당 클럭 틱 = 32,768 ticks
  - Contiki 운영체제에서 CPU 작동 클럭 = 8 MHz
- **데이터 시트 확인 내역**
  - 작동 가능한 입력 전압 = 3v
  - 1MHz 당 에너지 소비량 = 0.5 mA
  - 무선데이터 수신 모드일 때 에너지 소비량 = 18.8 mA
  - 무선데이터 송신 모드일 때 에너지 소비량 = 17.4 mA
- **에너지 타입 별 1 클럭 틱 당 에너지 소비량**
  - **CPU** = 0.5 * 8 / 32768 * 3 = **0.00037** mW
  - **RX** = 18.8 / 32768 * 3 = **0.00172** mW
  - **TX** = 17.4 / 32768 * 3 = **0.00159** mW

그림 5. 에너지 타입 별 클럭 틱 당 에너지 소비량
Fig. 5. Energy consumption per ticks for each type of energy

Ⅲ. DTLS 소비 에너지 분석

3.1 실험 환경

본 논문은 DTLS 통신을 하고자 하는 서버와 클라이언트가 인터넷을 경유하는 분리된 저전력 LLN망에 각각 속해 있을 때, 데이터 전송 과정에서 발생하는 DTLS Flight 간 특성을 에너지 소비량 관점에서 분석하기 위해 다음과 같은 실험 환경을 구성하였다.

Cooja 시뮬레이터를 통해 구성된 무선 네트워크와 센서 장치는 데이터 손실률을 0%로 설정하였으며, 클라이언트는 서버측 LLN망에 도달하기 위해서 인터넷 구간을 경유한다. 시뮬레이터 내 LLN망이 인터넷을 경유하도록 설정하기 위해 LLN망에서 게이트웨이 역할을 수행하는 보더 라우터에 tunslip6 명령어를 사용하여 호스트 컴퓨터의 네트워크 인터페이스와 통신이 가능한 IPv6주소를 부여하였다. 또한 호스트 컴퓨터의 네트워크 인터페이스간 IPv6 포워딩이 가능하도록 sysctl 명령어를 적용하였다. 서버와 클라이언트가 교환하는 메시지는 평문 상태에서 "Hello\n" 6 바이트이다. 실험을 위한 측정의 시작은 클라이언트가 서버에 최초로 UDP접속을 하는 순간부터 시작된다. 그리고 클라이언트가 서버에게 전송한 어플리케이션 데이터가 메아리 되어 돌아올 때를 측정 종료 구간으로 정하

표 2. 시뮬레이션 환경 및 적용 값
Table 2. Simulation environment and parameter

Environment	Parameter
Simulator	Cooja(MSPSim)
Sensor Device	Zolteria Z1(MCU:msp430)
Compiler	msp430-gcc(in Ubuntu)
OS	Contiki 3.0
DTLS Library	TinyDTLS 0.8.2
CipherSuite	TLS_PSK_WITH_AES_128_CCM_8
Network Protocol	RPL/6LowPAN/IPv6/UDP
Wireless Media	IEEE 802.15.4
Method for Energy Mesurement	energest function of Contiki OS
Sending Data	6 bytes (String "Hello\n")

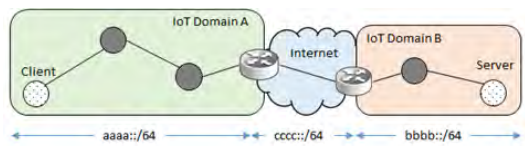

그림 6. 실험 네트워크 구조
Fig. 6. Test Network model

였다. 실험 환경의 네트워크 구조는 그림 6과 같다.

TinyDTLS는 DTLS를 경량화하여 C언어로 구현한 오픈소스 라이브러리이다[11]. 리눅스와 Contiki에서 이 라이브러리의 API를 이용해 어플리케이션 개발이 가능하다. 본 논문의 실험에서는 TinyDTLS 소스코드에서 각 Flight의 시작과 끝 지점 그리고 어플리케이션 데이터를 전송하기 위한 시작과 끝 지점에 각각 energest_type_time 함수 코드를 삽입하였다. 이 때 성능 측정에 영향을 최소화하기 위하여 기본적으로 디버깅 메시지를 출력되도록 선언된 매크로 코드를 제거하였다. DTLS 키 교환 알고리즘은 Pre-shared Key 방식의 대칭키를 사용한다.

Contiki는 적용되는 경량화 장치의 특성에 따라 기본 제공되는 설정 값을 변경할 필요가 있다. 실험에 사용하는 Z1장치가 가진 램 8킬로바이트, 롬 92킬로바이트 크기 제한이 있다. 이 장치에 Tiny- DTLS가 적용된 펌웨어를 업로드하기 위해서는 Contiki에서 지원되는 기능 일부를 비활성화해야 한다. 본 논문의 실험에서는 불필요한 TCP 기능과 일부 6LoWPAN 드라이버 상에서 구현된 수신 큐의 버퍼크기를 일부 조정하였다. 전송되는 데이터 크기측정은 시뮬레이터의 네트워크 덤프 기능과 인터넷 구간에서의 측정은 패킷 덤프 프로그램 Wireshark을 사용하였다.

3.2 실험 결과

실험에 사용된 Zolteria Z1을 기준으로 클럭 틱 당 에너지 소비량은 그림 5에 나타낸 방식을 이용하여 구할 수 있다. 실험 결과는 Flight간 비교가 가능하도록 Flight 별 에너지 소모량, 처리 시간 및 수신 시간,

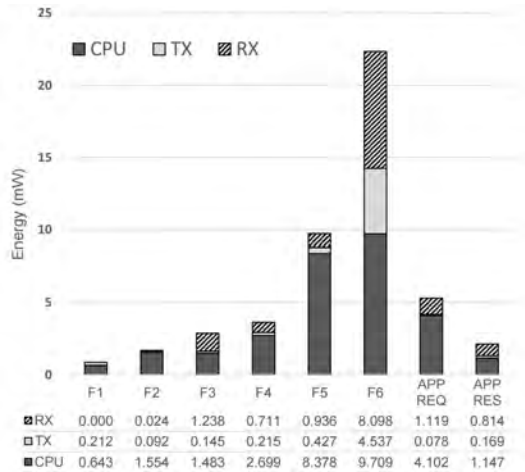

그림 7. Flight 별 에너지 소비량 (F1: Flight 1)
Fig. 7. Energy consumption per flight

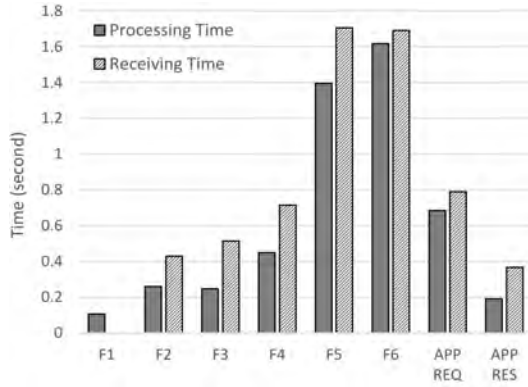

그림 8. Flight 별 처리 시간 및 수신 시간
Fig. 8. Processing and receiving time per flight

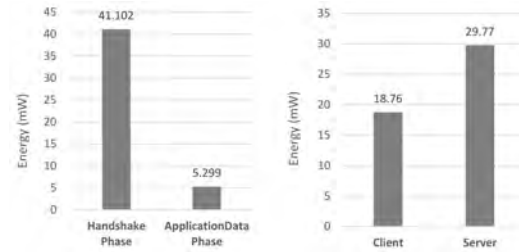

그림 9. DTLS 내 계층별 에너지 소비량 비교 (좌)
Fig 9. Energy consumption of each phase

그림 10. 클라이언트와 서버의 에너지 소비량 비교 (우)
Fig. 10. Energy consumption of client and server

전송데이터 크기 3가지를 그래프로 나타내었다. 또한 핸드쉐이크 단계와 어플리케이션 데이터 단계의 에너지 소비량 비교 및 클라이언트와 서버의 에너지 소비량을 각각 비교하였다.

에너지 소비량 측정에서 LPM은 반드시 고려되어야 하는 요소 중 하나이다. 그러나 정확한 시간을 구하기 위한 목적으로 하드웨어 타이머 호출이 사용되면 비활성화된다[12]. TinyDTLS 내부 구현에서 이 호출로 인해 LPM 기능은 비활성화되어 저전력 모드가 사용되지 않았으므로 측정 결과에서 제외하였다.

클라이언트가 서버에 DTLS를 이용해 접속하여 암호화된 메시지로 데이터를 요청하고 다시 동일 데이터를 응답받은 메시지를 복호화하기까지 11.162 초가 소요되었다. 이 때 소비된 에너지 양은 밀리와트 단위를 기준으로 클라이언트 18.76mW, 서버 29.77mW이다(그림 10). 클라이언트와 서버가 교환된 정보를 토대로 ChangeCipherSpec 과정을 통해 키를 생성하고 이를 각 주체 간 검증하는 과정이 포함되어 있기 때문에 Flight 5, 6에서 핸드쉐이크 전체 과정의 에너지 소비량

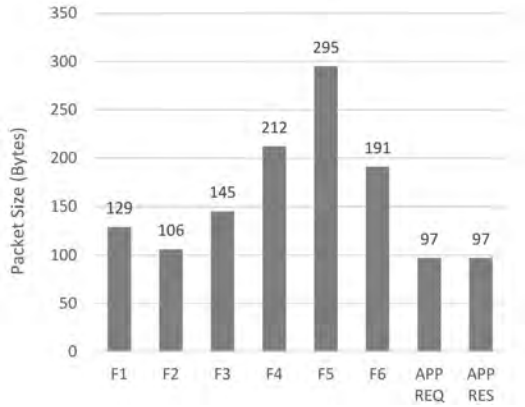

그림 11. Flight 별 전송 데이터 크기
Fig. 11. Data size measured per Flight

에 가장 큰 영향을 미치는 것으로 나타났다(그림 7).

특히 클라이언트와 서버의 에너지 소비량을 비교했을 때 서버의 수치가 더 큰 이유는 에너지 소비량이 가장 큰 Flight 6의 영향을 받았기 때문이다. 그림 1에서 Flight 5와 Flight 6는 차이가 없지만 내부적으로 Flight 6는 Flight 5에서 수신된 데이터를 검증하는 과정이 포함되어 있기 때문에 에너지 소비량이 더 크다.

그림 7, 8, 11의 그래프 상에서 x축의 APPREQ와 APPRES는 어플리케이션 데이터가 암호화된 레코드 계층에 속한다. APPREQ는 클라이언트에서 최초로 송신하는 어플리케이션 데이터를 의미하며 APPRES는 클라이언트로부터 수신한 암호화 데이터를 복호화한 후, 동일한 데이터를 암호화하여 클라이언트에게 다시 응답하는 어플리케이션 데이터를 의미한다. 그림 11에서 각 주체가 생성하는 어플리케이션 데이터 전송 크기는 97바이트로 동일하다. 하지만 클라이언트에서 어플리케이션 데이터를 처리하기 위해서는 반드시 Flight 6에 대한 검증이 선행되어야하기 때문에 에너지 소비량은 2배 이상 차이가 난다. 이 후 부터는 요청과 응답의 평문크기가 같은 경우, 에너지 소비량 차이가 발생하지 않는다.

동일한 LLN망에서의 장치 간 통신과 달리 실험 환경과 같이 인터넷 외부로 패킷이 전송되는 경우에는 6LoWPAN에 의한 IPv6 헤더 압축률이 현저하게 떨어진다. 본 논문의 실험 환경에서 단편화가 발생하지 않는 최대 페이로드 크기는 59바이트인 것으로 확인되었다. 59바이트 이상의 데이터를 전송하는 경우, 센서 장치와 보더 라우터 간에는 단편화가 발생하며 이것은 더욱 많은 배터리 소모와 네트워크 지연을 유발하게 된다. 그림 11의 각 Flight 전송 데이터 크기에서

표 3. Flight 별 6LoWPAN 단편화 발생 여부
Table 3. 6LoWPAN fragmentation occurred in each flight

Flight No	Message Type	Payload Size	Frag-mentation
1	ClientHello	67 bytes	O
2	HelloVerifyRequest	44 bytes	X
3	ClientHello	83 bytes	O
4	ServerHello	63 bytes	O
	ServerHelloDone	25 bytes	X
5	ClientKeyExchange	42 bytes	X
	ChangeCipherSpec	14 bytes	X
	Finished	53 bytes	X
6	ChangeCipherSpec	14 bytes	X
	Finished	53 bytes	X
APP REQ	Application Data	35 bytes	X
APP RES	Application Data	35 bytes	X

이더넷 프레임헤더 14바이트, IPv6 패킷헤더 40바이트, UDP헤더 8바이트를 합한 62바이트를 제외하면 각 Flight의 페이로드 크기를 구할 수 있다. 표 3은 계산된 페이로드 크기와 단편화 발생 여부를 정리한 것이다.

3.3 실험 분석

DTLS에서 처리되는 모든 데이터는 그림 12로 요약할 수 있다. 또한 에너지 소비량을 결정하는 요소를 예측 가능한 요소와 불가능한 요소 2가지로 분류할 수 있다. 첫째, 예측 가능한 요소는 센서 장치 하드웨어 성능과 핸드쉐이크 과정에서 선택된 CipherSuite 암호화 알고리즘의 성능이다. 둘째, 예측 불가능한 요소는 무선 네트워크와 인터넷 구간에서 발생되는 전송지연율과 단편화 여부이다.

이 요소들을 바탕으로 에너지 소비량 관점에서 다음과 같이 정리할 수 있다.

- 코드 크기

Contiki 펌웨어가 약 40킬로바이트 정도의 크기임

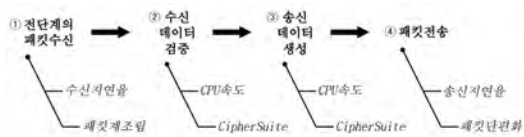

그림 12. Flight 성능에 영향을 미치는 요소
Fig. 12. Factors affecting on performance of each flight

에도 DTLS 구현체인 TinyDTLS를 배포된 버전 그대로 사용하기 위해서는 일부 불필요한 기능의 비활성화가 필요하다. 본 실험의 경우, Z1 장치에 DTLS 관련 코드만 포함하여 실행 상에 문제는 없었으나, 실제 서비스를 위해 서비스 구현과 부가 기능 구현이 포함되어야 할 경우에는 램과 롬 크기 제약으로 인해 문제가 발생할 수 있다. 따라서 Contiki와 TinyDTLS의 소스코드에서 적절한 최적화가 반드시 필요하다. IETF에서 성능이 제약된 장치에 대해 Class 0~2 까지 3가지로 분류한 기준에 의하면, 본 실험에 사용된 장치는 제약이 적은 Class 2에 속한다[13]. 그러나 많은 DTLS 구현체 중에서 가장 경량화된 것으로 알려진 TinyDTLS가 최적화를 하더라도 Class 1의 장치에서 사용이 불가능하다는 것은 코드 수준에서의 경량화가 더욱 필요한 것을 의미한다. 그러므로 경량화 장치로 구성된 사물인터넷 환경에서 DTLS기반 암호화 통신을 위해서는 현재 공개된 구현체에 대한 보완이 필요하다.

• CipherSuite

그림 5와 같이 하드웨어 칩셋에 의한 에너지 소비량은 센서 장치의 관점에서 고정 상수로 표현할 수 있다. 그러나 키 교환 알고리즘과 키 생성 및 어플리케이션 데이터 암복호화 알고리즘을 의미하는 CipherSuite는 선택되는 알고리즘에 따라 성능 변화의 폭이 매우 크다. 특히 키 교환 알고리즘에서 Pre-shared Key 방식과 Public Key 방식의 연산 속도는 약 10배 정도 차이가 나는 것으로 알려져 있다. 비대칭키 방식의 알고리즘은 자원이 제한된 네트워크와 장치에서 현실적으로 사용이 불가능하다. 따라서 경량화 장치의 성능을 고려한 CipherSuite의 선택과 구현이 요구된다.

• 단편화

일반적으로 무선 환경에서 전송지연과 단편화는 NlogN 이상으로 성능이 악화될 수 있다. DTLS에서 세션 개설을 위한 핸드쉐이크 계층에서 단편화가 발생할 수록 데이터 처리 성능은 악화된다. 특히 키 교환 알고리즘으로 비대칭키 암호화 알고리즘이 선택된 경우, 연산 속도 저하뿐만 아니라 그림 1에서 실선으로 표시된 부분이 처리되어야 하지만, 이 때 교환되는 데이터 크기는 단편화가 발생하는 페이로드 크기 59 바이트를 훨씬 상회한다. 또한 802.15.4 무선 네트워크에서 멀티홉으로 라우팅 경로가 설정된 경우 단편화에 의한 에너지 소모량은 극단적으로 악화될 것을 충분히 예상할 수 있다. 따라서 멀티홉 환경을 지양할 수 있는 네트워크 설계가 고려되어야 한다.

Ⅳ. 결 론

본 논문에서는 배터리, CPU와 메모리, 무선환경 등 모든 자원이 제한된 환경에서 DTLS를 사용할 때 발생하는 에너지 소모량, 처리 시간 및 수신 시간 그리고 전송데이터 크기를 측정하였다. 그 결과, 제한된 자원을 가진 센서 장치에서 DTLS 프로토콜을 사용할 경우 핸드쉐이크 계층의 에너지 소비량은 레코드 계층 대비 약 8배 이상을 소모하는 것을 확인하였다. 또한 802.15.4 기반 네트워크 환경에서 핸드쉐이크 수행 시, 소요되는 시간을 각 Flight 별로 제공하였다. 특히, Fight 5와 6의 에너지 소모량은 적용하는 보안 프리미티브에 영향을 받게 된다. 본 논문에서는 소비 에너지가 가장 적은 PSK 기반 방식의 결과를 제공하고 있다. 공개키 기반 방식의 DTLS 보안 모드를 사용할 경우 에너지 소비는 더욱 크게 증가할 것이다. 이를 포함한 본 논문의 실험 데이터들은 경량 장치로 구성된 사물인터넷 환경에서 DTLS 프로토콜 이용 시, 네트워크 환경을 구성하는데 참조될 수 있을 것으로 기대된다.

References

[1] Z. Shelby, K. Hartke, and C. Bormann, *The constrained application protocol (CoAP)*, IETF, RFC 7252, 2014.

[2] E. Rescorla and N. Modadugu, *Datagram transport layer security version 1.2*, IETF, RFC 6347, 2012.

[3] M. Nagendra and E. Rescorla, “The design and implementation of datagram TLS,” *in Proc. NDSS*, San Diego, USA, Feb. 2004.

[4] M. Ortega, P. Eronen, and S. Holtmanns, Transport layer security (TLS) parameters (2015), Retrieved May., 27, 2015, from http://www.iana.org

[5] T. Dierks and E. Rescorla, *The transport layer security (TLS) protocol version 1.2*, IETF, RFC 5246, 2008

[6] IEEE 802.15.4 Standard, IEEE 802.15 Working Group for WPAN(2015), Retrieved May., 27, 2015, from http:// www.ieee802.org/15/

[7] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, *Transmission of IPv6 packets over IEEE 802.15. 4 networks*, IETF, RFC 4944, 2007.
[8] http://www.contiki-os.org
[9] G. Pietro and S. Duquennoy, Hands on contiki OS and Cooja simulator: Exercises (Part II)," Retrieved Jul. 7, 2015, from http://team.inria.fr
[10] A. Dunkels, J. Eriksson, N. Finne, and N. Tsiftes, *Powertrace: Network-level power profiling for low-power wireless networks*, SICS Technical Report, T2011:05, ISSN 1100-3154, 2011.
[11] http://tinydtls.sourceforge.net
[12] H. Razi and T. Qamar, *Asymmetric-key cryptography for contiki*, Dept. of Computer Sci. and Eng., 2010.
[13] C. Bormann, M. Ersue, and A. Keranen, *Terminology for constrained-node networks*, IETF, RFC 7228, 2014
[14] G. Park, H. Han, and J. Lee, "Design and implementation of lightweight encryption algorithm on OpenSSL," *J. KICS*, vol. 39, no. 12, pp. 822-830, 2014.
[15] J. Park and N. Kang, "Entity authentication scheme for secure WEB of things applications," *J. KICS*, vol. 38, no. 5, pp. 394-400, 2013.
[16] J. Park, S. Shin, and N. Kang, "Mutual authentication and key agreement scheme between lightweight devices in internet of things," *J. KICS*, vol. 38, no. 9, pp. 707-714, 2013.

권 혁 진 (Hyeokjin Kwon)

2010년 1월~2011년 7월 : (주) 안랩 CERT팀 대리
2011년 7월~2014년 8월 : (주) 다음카카오 메일개발팀
2014년 9월~현재 : 덕성여자대학교 디지털미디어학과 석사과정
관심분야 : 인터넷 보안, 프로토콜, 빅데이터

강 남 희 (Namhi Kang)

2001년 2월 : 숭실대학교 정보통신대학원 공학석사
2004년 12월 : University of Siegen 컴퓨터공학과 공학박사
2009년 3월~현재 : 덕성여자대학교 디지털미디어학과 부교수
관심분야 : 유무선 인터넷통신, 통신보안프로토콜, 시스템 보안, 사물인터넷 보안