Journal of the Korea Industrial Information Systems Research Vol. 23 No. 3, Jun. 2018 :1-11
http://dx.doi.org./10.9723/jksiis.2018.23.3.001 ISSN:1229-3741

수신 대기시간 보정을 통한 IEEE 802.15.4 MAC 프로토콜의 비컨 동기화 신뢰성 개선

(Improving the Reliability of Beacon Synchronization of IEEE 802.15.4 MAC Protocol using a Reception Time Compensation Scheme)

김 희 철[1)*]
(Hiecheol Kim)

요 약 이 논문은 OS가 제공하는 소프트웨어 타이머를 사용한 네트워킹 스택의 구현 상황에서 고도의 정확성이 요구되는 시분할 동기적 통신의 신뢰성 문제를 다룬다. IEEE 802.15.4를 목표 프로토콜로 선정해 비컨 동기화의 신뢰성 향상 이슈를 상세히 탐구한다. 먼저, IEEE 802.15.4 비컨 모드의 구현에 소프트웨어 타이머를 사용할 때, 소프트웨어 타이머에 사용되는 하위 하드웨어 타이머 인터럽트의 처리 지연 및 유실과 소프트웨어 타이머 관리 오버헤드 등이 실제로 비컨 동기화의 저해 요인으로 작용한다는 점을 실험을 통해 확인한다. 이러한 상황을 개선하기 위한 제안한 비컨 대기시간 보정기법은 비컨 동기화의 신뢰성을 향상시킨다는 점을 입증한다.

핵심주제어 : IEEE 802.15.4, 비컨 모드, 비컨 동기화, 소프트웨어 타이머

Abstract This paper explores the reliability issue especially associated with the time-division synchronous communication when a networking stack is implemented using software timers provided by embedded operating systems. Especially, we explore the reliability of beacon synchronization of IEEE 802.15.4. Our experiments based on its practical implementation clearly show that processing delays or losses of hardware timer interrupts used for software timers lead to occasional failures in beacon synchronization. To avoid such failures, we suggest a reception time compensation scheme that turns on the receiver earlier than expected.

Key Words : IEEE 802.15.4, Beacon Mode, Beacon Synchronization, Software Timer

1. 서 론

IEEE 802.15.4는 저전력 무선 통신을 위한 근거리 무선 개인 네트워크(LR-WPAN, Low-Rate Wireless Personal Area Network)를 위한 MAC(Medium Access Control Layer)과 물리계층(Physical Layer)의 국제 표준이다[1]. 이 프로토콜은 ZigBee, WirelessHART, MiWi 표준의 기저 계층으로 사용되고 있으며, 위치 기반 서비

* Corresponding Author : hckim@daegu.ac.kr
+ 이 논문은 2014학년도 대구대학교 학술연구비 지원에 의한 논문임
Manuscript received April 27, 2018 / revised June 8, 2018/ accepted June 15, 2018
1) 대구대학교 정보통신공학부, 제1저자

스, 원격 검침 등의 많은 응용분야에서 이용되고 있다[2,3]. IEEE 802.15.4에서는 디바이스의 종류를 RFD(Reduced Function Device)와 FFD(Full Function Device) 두 가지로 정의한다. FFD는 MAC 계층의 모든 기능을 제공할 수 있으며 네트워크 코디네이터(Network Coordinator) 역할뿐만 아니라 단말 노드의 역할도 수행할 수 있다. FFD가 네트워크 코디네이터 역할을 수행할 때는 네트워크의 단말에 대한 동기화, 단말의 네트워크 참여(Association)와 탈퇴(Disassociation) 서비스를 담당한다. 한편, RFD는 오직 단말 노드의 역할만을 수행하며, 응용요구사항에 따라 센서 및 액추에이터 등 다양한 장치를 장착할 수 있다.

IEEE 802.15.4 표준은 다양한 저전력 무선 네트워크와 IoT(Internet of Thinks) 네트워크의 기저 계층으로서의 잠재력으로 인해 이미 많은 연구가 진행되었다[3-6]. 이러한 연구는 네트워크의 성능 분석, 네트워크 구성속도 향상과 주소할당 방식 개선, 네트워크 수명 연장 등에 거친 다양한 주제에 대한 많은 유용한 결과가 산출했다. 그렇지만 기존 연구는 대부분 이론적 분석이나 시뮬레이션 방식을 채택하고 있어 실환경을 충분히 반영하고 있지 못하다는 한계가 있다. 결국, 이런 연구 결과를 실환경의 실질적인 성능 결과로 삼기에는 한계가 있다.

본 연구에서는 대부분의 기존 연구와는 달리 실제 구현을 통한 실험 기반 연구를 수행한다. 특히, IEEE 802.15.4의 비컨 모드를 기반으로 구축한 네트워크(Beacon-Enabled Network)에서 가장 복잡하고 어려운 비컨 동기화(Beacon Synchronization) 문제를 다룬다. 비컨 모드에서는 네트워크에 참여한 일반 노드는 네트워크 코디네이터의 비컨 메시지에 의거해서 활성구간 및 비활성구간과 슬롯을 식별하고 이를 기반으로 송수신 동작을 수행한다. 일반적으로 비컨 모드는 논비컨(Non-Beacon) 모드와 비교해서 에너지 절감, 충돌 최소화, 전송 신뢰성을 향상시킬 수 있는 장점을 가진다. 하지만, 이러한 성능 및 신뢰성 향상의 장점을 실효화하기 위해서는 코디네이터와 다른 일반 노드 간에 반드시 비컨의 정확한 송수신이 이뤄져야 하는데, 이를 비컨 동기화 문제라고 한다[7-9].

본 연구는 크게 두 부분으로 구성된다. 먼저, 비컨 모드에서 비컨 동기화의 신뢰성을 저하하는 요인들을 도출한 후, 이러한 요인들이 복합적으로 작용해 실제로 비컨 동기화의 신뢰성이 저하하는 현상을 실험을 통해서 확인한다. 다음으로 이런 신뢰성 저하 현상을 극복하려는 방법으로 수신 대기시간 보정기법을 제시한 후, 실제 구현을 기반으로 한 실험을 통해 이 기법이 비컨 동기화의 신뢰성 향상에 기여한다는 점을 검증한다.

본 논문은 다음과 같이 구성된다. 2절에서는 본 연구 주제와 관련한 배경지식으로 IEEE 802.15.4 표준의 비컨 모드를 간략히 소개한다. 3절에서는 비컨 동기화 문제의 주요 요인을 분석한다. 4절에서는 비컨 동기화 문제 발생 현상을 실험을 통해 파악한다. 5절에서는 제안하는 비컨 대기시간 보정기법의 적용 결과를 설명한다. 마지막으로 6절에서는 본 연구의 결과에 대한 고찰로 결론을 맺는다.

2. IEEE 802.15.4의 비컨 모드

2.1 시스템 구성

IEEE 802.15.4는 프레임 전송 동작과 관련하여 비컨 모드와 논비컨 모드를 둘 다 지원한다. 비컨 모드는 전체 네트워크가 시간 동기화 환경에서 동작할 수 있도록 하며, 시간 동기화에는 비컨 메시지를 사용한다[1]. 이런 비컨 동기화 환경에서는 시분할을 통해 활성화 구간과 비활성화 구간을 나눈 후에, 비활성화 구간에는 네트워크에 참여한 모든 노드가 송수신기 전력을 차단해 전력소모를 줄일 수 있다. 한편, 논비컨 모드의 네트워크에서는 네트워크의 동기화 기능이 존재하지 않아서 모든 노드는 항상 송수신기를 켜 놓은 상태를 유지해야 하므로 전력의 소모가 클 수밖에 없다. IEEE 802.15.4를 채택한 저전력 무선 네트워크는 일반적으로 자원 제약적 환경에서 가동되므로 네트워크 운영에 있어 논비컨 모드보다는 비컨 모드가 선호될 수밖에 없다.

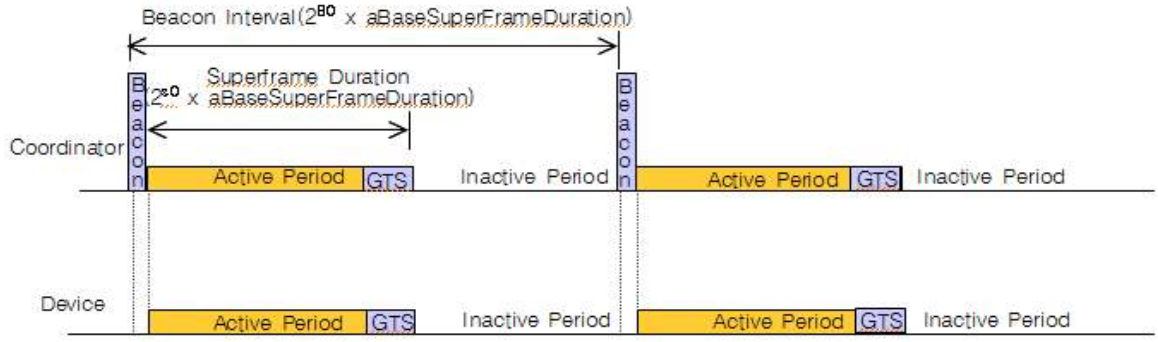


Fig. 1 Superframe Structure & Beacon Synchronization between the Coordinator and a Device

비컨 모드에서는 비컨과 비컨 사이규격은 슈퍼프레임으로 명시된다. Fig. 1에서 볼 수 있듯이 슈퍼프레임은 일종의 듀티 사이클(Duty Cycle)이며, 활성화(Active) 구간과 비활성화(In-Active) 구간으로 구성된다. 세부적으로 살펴보면, 활성화 구간은 전체 16개의 슬롯으로 나뉘는데, 세부적으로 일부는 충돌기반접근구간(CAP, Collision Access Period)으로 나머지는 충돌자유구간(CFP, Collision Free Period)으로 분리해 운용할 수 있다. 충돌기반접근구간에서는 각 노드가 CSMA-CA(Carrier Sense Medium Access - Collision Avoidance) 기반의 경쟁방식으로 매체에 접근한다. 충돌자유구간에서는 코디네이터가 보낸 비컨 메시지 내에 매체 접근이 가능한 노드가 명시되어 있으며, 매체 접근에 있어 노드 간의 경쟁이 배제된다. 슈퍼프레임 내의 활성화 구간의 크기는 고정된 것이 아니라 BO(Beacon Order)와 SD(Superframe Duration) 값을 조정해서 여러 다른 크기로 설정할 수 있다. 이 두 파라미터에 대한 의미와 설정값의 범위 등은 이후 성능평가에서 설명한다.

3. 비컨 동기화 문제

3.1 비컨 동기화 정의

IEEE 802.15.4에서 지원하는 네트워크의 구성형태는 성형 토폴로지(Star Topology)와 피어 투 피어 토폴로지(Peer-to-Peer Topology)가 있다. 성형 토폴로지에서는 하나의 FFD가 네트워크 코디네이터의 역할을 맡는다. 나머지 RFD와 FFD은 오직 이 네트워크 코디네이터와 단일 홉 통신을 수행한다. 반면, 피어 투 피어 토폴로지에서의 FFD는 통신 반경 내의 다른 FFD와 통신하고 통신반경 외의 FFD들과 메시지를 중계하는 멀티 홉 통신을 수행한다. 하나의 네트워크 코디네이터와 다른 노드 간의 시간을 맞추는 것을 동기화라고 한다. 코디네이터는 Fig. 1에 나타난 비컨 메시지를 통해 비컨 주기와 슈퍼프레임의 구성을 알려서 네트워크를 구성하는 노드들이 동기를 맞추도록 지원한다.

Fig. 1은 디바이스가 코디네이터로부터 비컨을 정상적으로 수신한 비컨 동기화한 상황을 보여준다. 만약 디바이스가 비컨을 수신하지 못하면 직접 동기 메시지를 요청한 후, 비컨을 수신할 때까지 수신기를 활성화한다. 만약 네트워크 코디네이터가 전송하는 비컨을 지정된 시간 내에 정상적으로 수신하지 못하는 상황이 지속해서 반복되면, 해당 노드는 동기화에 실패하게 된다.

3.2 본 연구에서의 시스템 모델과 비컨 동기화 문제

비컨 기반 네트워크에서 비컨 동기화, 즉 코디네이터와 일반 노드 간의 비컨의 정상적인 송수신을 저해하는 요인은 클록 드리프트(Clock Drift)와 같은 미시적 현상부터 타이머 인터럽트 지연이나 소실 등과 같은 거시적 현상에 이르기까지 매우 다양하다.

클록 드리프트는 클록을 발생시키는 소자가 온

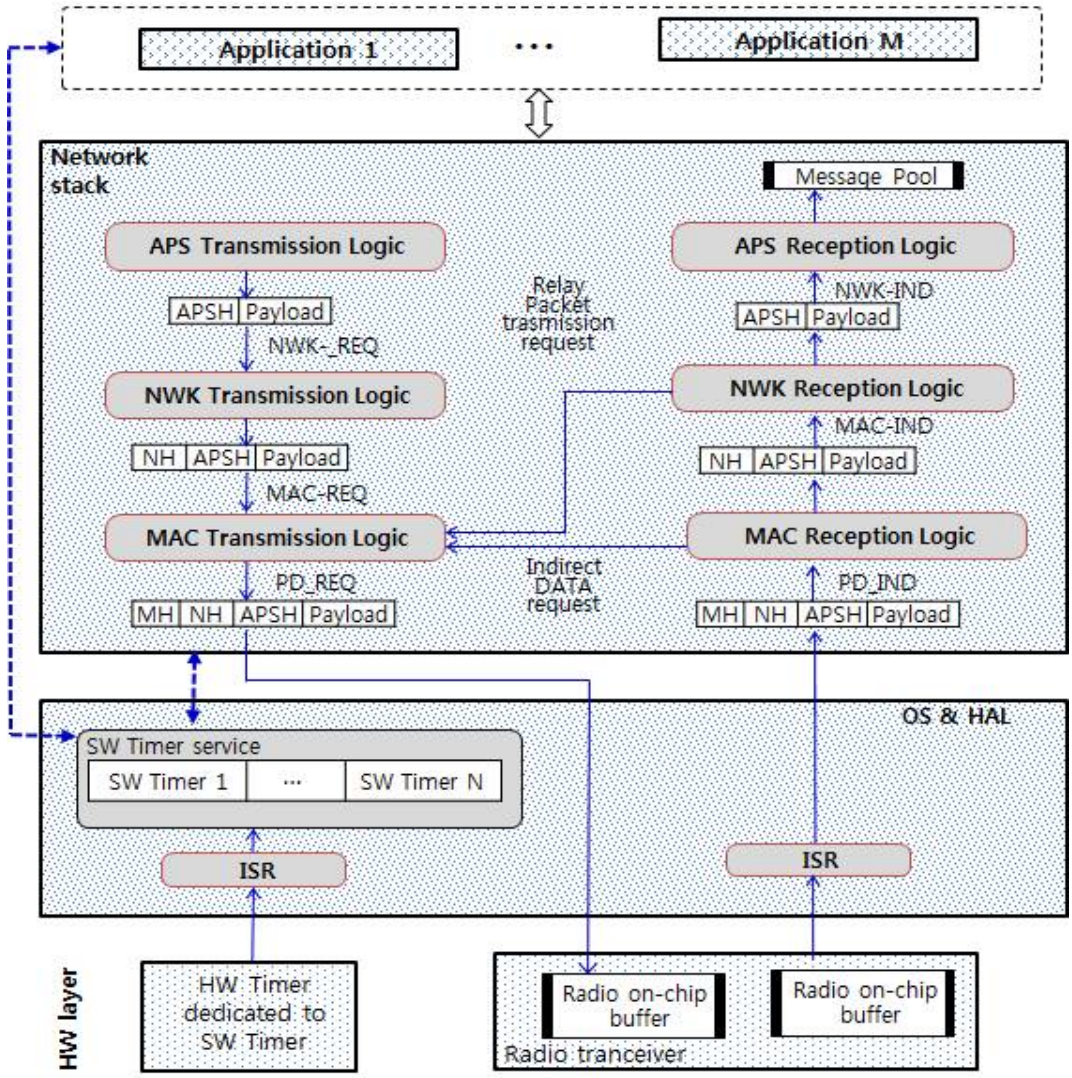


Fig. 2 Software organization on a node in a beacon-enabled network

도나 압력 등의 변화에 따라 클록 주기가 미세하게 변화하는 현상을 말한다. 클록 드리프트는 기준 클록보다 빠르거나 느릴 수 있는데, 그 변화량이 매우 미세할지라도 지속해서 누적되는 특성으로 일정 임계치가 넘어서면 클록에 의해 구동되는 타이머의 주기에 영향을 미친다. 이러한 영향은 비컨의 발생 및 수신 주기를 왜곡시켜 궁극적으로 비컨의 송수신이 불안정해지는 상황을 초래할 수 있다. 클록 드리프트 현상을 극복하기 위한 다양한 연구는 최근까지 수행되고 있어 다양한 타임 동기화 알고리즘이 개발되고 있다[10-12].

본 연구는 위와 같은 클록 드리프트에 초점을 둔 타임 동기화 알고리즘과는 달리 실제 네트워크 스택의 구현 과정에서 발생하는 비컨 동기화 저해 문제를 다룬다. Fig. 2는 본 연구에서 다루는 네트워크 스택 소프트웨어의 구현 모델을 보여준다. 이 소프트웨어 모델은 TinyOS나 Mantis, 다양한 ZigBee 플랫폼 등에서 채택하고 있는 일반적인 모델이며, 멀티스레드 OS와 한 개 이상의 스레드로 구성된 네트워크 스택, 그리고 하위의 HAL(Hardware Abstraction Layer)로 구성된다[13,14]. OS 내에는 하드웨어 타이머의 부족 상황을 해결하기 위한 소프트웨어 타이머

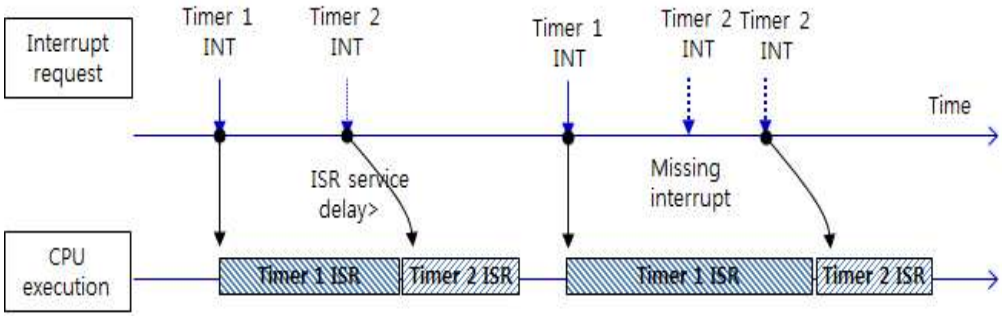


Fig. 3 Interrupt Delay and Missing Phenomena

서비스가 제공되며, 네트워크 스택의 구성요소나 애플리케이션 스레드는 이 서비스를 통해 필요할 때마다 SW 타이머를 생성해서 사용한다.

일반적으로 임베디드 운영체제 환경에서 네트워크 스택을 구현하는 경우에는 스레드 라이브러리나 소프트웨어 타이머 서비스 등 다양한 운영체제 서비스를 사용할 수 있어 그 구현이 쉽다는 장점이 있다. 특히 네트워크 스택을 위한 자체 전용 하드웨어 타이머를 갖지 않고 소프트웨어 타이머 서비스를 사용할 수 있어 다양한 타이머의 요구가 유연하게 만족될 수 있다. 하지만, 소프트웨어 타이머의 부정확성이 문제점으로 제기되는데 그 문제점은 크게 두 가지로 압축된다. 하나는 운영체제의 스레드 간의 문맥전환이나 소프트웨어 타이머의 등록, 시간 갱신, 콜백(Call-Back) 호출 등과 같은 소프트웨어 타이머 관리에 수반되는 오버헤드에 따른 소프트웨어 타이머의 부정확성이다. 다른 하나는 소프트웨어 타이머에 사용하는 하드웨어 인터럽트 지연 및 유실 현상에 의한 소프트웨어 타이머의 부정확성이다.

위의 두 가지 문제점 중에서 후자인 인터럽트의 지연 및 유신 현상의 발생 원인은 두 가지로 하나는 인터럽트 우선순위이며, 다른 하나는 인터럽트 금지이다.

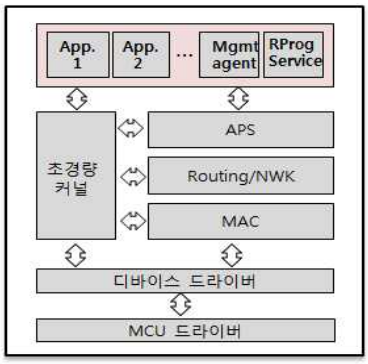
인터럽트 우선순위 : 임베디드 시스템에 사용되는 대부분의 MCU(Micro-Controller Unit)는 인터럽트 우선순위를 지원한다. 만약 타이머 인터럽트보다 우선순위가 높은 인터럽트의 서비스가 진행되는 동안에 타이머 인터럽트가 발생하면, 타이머 인터럽트의 처리가 지연된다. 센서나 IoT 디바이스에 사용하는 ATmega128, MSP430와 같은 대부분의 경량 MCU는 인터럽트가 동시에 발생할 경우 Interrupt Pending을 지원하지 않는다. 한 인터럽트가 발생했지만 처리가 지연되고 있는 상황에 동일한 인터럽트가 재발생하면 대기하고 있는 인터럽트의 유실이 발생할 수 있다. Fig. 3은 Timer 1의 우선순위가 Timer 2보다 높을 때, Timer 2 인터럽트의 처리지연과 유실 발생 상황을 나타내고 있다.

인터럽트 금지: 병행 태스크 간의 문맥전환(Context Switching)이나 임계영역(Critical Section) 등의 실행에 있어 무결성을 보장하기 위해서 인터럽트를 금지해야 하는 상황이 종종 있다. 이러한 상황에서도 타이머 인터럽트의 유실이나 지연 현상이 발생할 수 있다.

소프트웨어 타이머 서비스 오버헤드, 인터럽트 우선순위, 인터럽트 금지는 네트워크 스택의 구현에서 피할 수없는 요소이다. 이런 요소들로 인해 궁극적으로 상위 소프트웨어 타이머의 주기가 애초보다 커져 부정확해지는 상황이 촉발될 수 있다. 소프트웨어 타이머의 주기 증가에 따른 비컨 주기의 변동은 클록 드리프트와 구별되는 두 가지 특징을 지닌다. 하나는 비컨 주기가 노드별로 동적으로 변동한다는 점이고, 다른 하나는 누적 효과로 비컨 주기가 클수록 그 오차가 커진다는 점이다. 이런 상황에서는 바로 전에 비컨 동기화가 져서 코디네이터가 다음 비컨을 정확한 주기로 전송한다고 해도 수신노드 상에서 인터럽트 처리 지연이나 유실 등의 빈도 증가로 비컨 주기가 커지면 수신기를 미처 ON 상태로 켜놓지

(a) Experimental Setting

(b) SN-100 Sensor Node



(c) Embedded Node Software

Fig. 4 Experimental Environment

못해 비컨을 유실하는 상황이 발생할 수 있다.

4. 비컨 동기화 저해 영향 평가

본 절과 이어지는 5절은 실험 내용과 결과를 다룬다. 우선, 간략히 전체 공통 실험환경을 소개하며, 세부 실험별 특이한 사항은 해당 소절에서 설명한다. Fig. 4에서 볼 수 있듯이 공통 실험환경은 SN-100 센서노드와 내장 소프트웨어, 소프트웨어 오실로스코프로 구성되어 있다. SN-100 센서노드는 ATmega128과 CC2420을 장착하고 있으며, 이 센서노드에는 본 연구실에서 개발한 자체 경량 멀티스레드 OS 커널과 802.15.4 MAC을 포함한 ZWeaver 스택이 탑재되어 있다. ZWeaver는 ZigBee 얼라이언스 국제공인인증(인증번호 ZIG12021ZCP26823-24)을 취득하여 ZigBee 사양을 완벽하게 구현하고 있는 ZigBee 스택이다. 경량 임베디드 OS 커널은 소프트웨어 타이머 서비스를 제공하고 있으며, MAC, NWK, APS 등 ZWeaver의 모든 구성요소는 이 소프트웨어 타이머 서비스를 사용하고 있다. 네트워크는 한 개의 코디네이터가 8개의 디바이스로 구성된다. 실험에서는 8개의 엔드 디바이스 중에서 필요한 개수만 취해 반경 30cm의 스타형 토폴로지를 구성했다. 한편, 동기화 파형을 관찰하기 위해 노드의 DIO 포트 출력을 소프트 오실로스코프의 CH1과 CH2에 연결한 후 노드의 DIO 포트에 동기화 정보를 출력해 파형을 도시했다.

본 절에서는 소프트웨어 타이머의 부정확성이 비컨 동기화에 미치는 영향을 측정하기 위한 다양한 실험을 통해 비컨 동기화의 정확도를 평가한다. 디바이스와 코디네이터와 동기화 수행에는 MLME_sync_req 프리미티브를 사용했다. 대부분의 실험에서 비컨의 주기를 결정하는 BO(Beacon Order)와 슈퍼프레임의 크기를 결정하는 SO(Superframe Order)의 설정값은 각각 10과 9를 사용한다. 이때, 비컨 인터벌은 15.728sec이고 활성 구간의 길이는 7.864 sec이다.

4.1 비컨 전송 주기의 증가

이 실험에서는 비컨의 전송 주기를 측정한 후에 이론값과 비교함으로써 타이머의 부정확성이 비컨주기에 미치는 영향을 살펴본다.

실험에는 코디네이터 노드 하나를 사용하며, 타이머 주기 간의 지연에 의한 효과를 최대한 반영하기 위해 코디네이터 노드는 여타 메시지를 전송하지 않고 단순히 비컨 전송 기능만을 수행하도록 하였다. 비컨 주기를 측정하기 위하여 IEEE 802.15.4의 비컨 발생 프리머티브인 MLME_start_req를 사용했다. MLME_start_req의 입력 파라미터인 BO의 모든 범위에 대한 비컨 주기를 측정하였다. 이 실험에서 SO 값은 BO와 동일한 값을 사용하여 비활성화 구간이 존재하지 않도록 설정했으며, 주기 측정도구로는

Table 1 Measurements of Beacon Interval Deviation

BO	이론값(A) (sec)	측정값(B) (sec)	오차(B-A) (sec)	오차율 (%)
14	251.65824	251.923	+0.26476	0.10
13	125.82912	125.964	+0.13488	0.11
12	62.91456	62.954	+0.03944	0.06
11	31.45728	31.494	+0.03672	0.12
10	15.72864	15.748	+0.01936	0.12
9	7.864	7.876	+0.01200	0.15
8	3.93216	3.94	+0.00784	0.2
7	1.96608	1.971	+0.00492	0.3
6	0.98304	0.987	+0.00396	0.4
5	0.49152	0.495	+0.00348	0.7
4	0.24576	0.249	+0.00324	1.3
3	0.12288	0.126	+0.00312	2.5
2	0.06144	0.065	+0.00356	5.8
1	0.03072	0.034	+0.00328	10.6
0	0.01536	0.019	+0.00364	23.7

DainTree 패킷 분석기를 이용하였다.

Table 1은 비컨 주기 측정 결과를 보여준다. BO 값별로 측정값, 이론값, 오차, 오차율이 나타나 있다. 오차는 측정값에서 이론값을 뺀 값이며, 그 주된 원인은 소프트웨어 타이머의 부정확성이다. 앞에서 설명한 바와 같이 소프트웨어 타이머의 부정확성은 하드웨어 타이머 인터럽트를 바탕으로 가상 타이머를 제작·관리하는 데서 발생하는 오버헤드의 누적과 타이머 인터럽트 지연 및 유실의 두 가지 요인으로 초래된다. 이 실험에서는 측정환경에서 비컨 이외의 메시지 송수신과 상위 응용 애플리케이션이 제거된 상태이다. 따라서 하드웨어 타이머 인터럽트 유실보다는 소프트웨어 타이머 관리 오버헤드의 누적이 비컨 주기의 오류의 실질적인 원인이 된다.

측정결과를 살펴보면, 모든 비컨 주기에서 측정치가 이론치보다 크다는 것을 확인할 수 있다. 이런 현상은 소프트웨어 타이머 인터럽트 주기가 증가했다는 점을 나타낸다. BO 값이 5 이하인

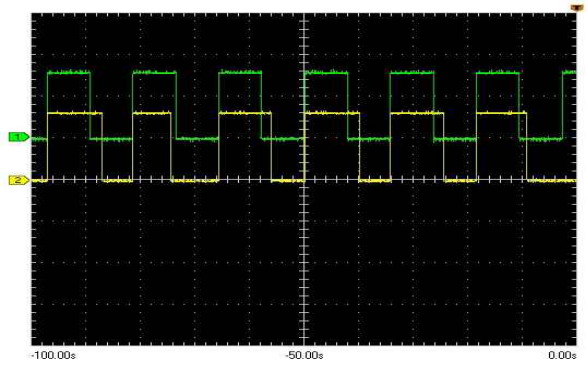


Fig. 5 Superframes of End-device 1

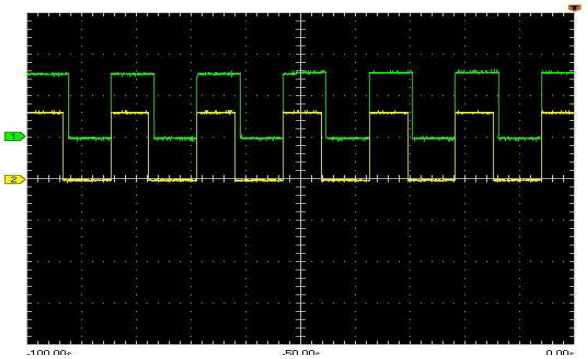


Fig. 6 Superframes of End-device 2

작은 크기의 비컨 주기에 대해서는 오차의 크기가 약 3.5msec로 균일한 값을 보이지만, 일정 크기, 즉 BO 값이 5 이상에서는 오차가 점차적으로 증가하고 있는 것을 볼 수 있다. 이런 현상은 비컨의 이론적 크기가 500msec 이하인 경우에는 소프트웨어 타이머의 부정확성에 소프트웨어 타이머 관리 오버헤드가 주된 영향을 미치지만, 그 이상일 경우에는 누적효과도 본격적으로 영향을 미치는 것으로 해석할 수 있다.

4.2 소프트웨어 타이머 관리 오버헤드 분석

이 실험을 위한 네트워크는 코디네이터와 2개의 엔드 디바이스로 구성된다. 코디네이터는 비컨 이외의 다른 메시지 송수신과 응용 애플리케이션 부하를 배제해 단순히 비컨 전송 기능만을 수행하도록 설정되었다. 엔드 디바이스 상에서는 0.1 msec 주기로 대기(Waiting) 상태와 활성화(Computing) 상태를 반복하는 애플리케이션이

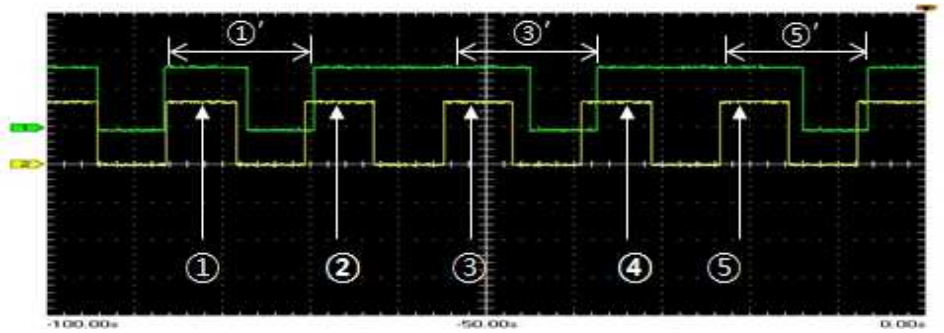


Fig. 7 Beacon Synchronization Failure (Device : 1, Coordinator : 2)

구동한다.

두 엔드 디바이스 상의 애플리케이션은 동일한 기능을 갖지만 대기와 활성화 시간을 조정을 위한 라이브러리 함수에 서로 다른 버전을 채택하고 있다. 첫 번째 엔드 디바이스의 경우에는 OS에서 제공하는 delay_until_event()를 사용한다. 이 함수는 OS 내부의 소프트웨어 타이머를 사용하며, 앞 소절에서 지적한 바와 같이 하드웨어 타이머를 직접 사용하는 버전보다 타이어 정확성에 떨어지는 단점을 지닌다. 참고로, 이 함수는 비컨 관리에 사용하는 것과 동일한 소프트웨어 타이머를 사용한다. 소프트웨어 타이머 주기의 증가를 초래할 가능성이 있다. 두 번째 엔드 디바이스 상의 애플리케이션은 이 실험을 위해 여분의 하드웨어 타이머를 직접 사용해서 구현한 self_delay_until_event() 함수를 사용한다.

위와 같은 환경에서 엔드 디바이스 각각에 대해 슈퍼프레임의 활성화와 비활성화 구간의 크기를 측정하였으며, 그 결과는 각각 Fig. 5와 Fig. 6에 나타냈다. 그림 내에는 두 개의 파형이 나타나 있는데, 각 파형은 슈퍼프레임의 시계열 전개를 나타낸다. 이미 앞에서 설명한 바와 같이 각 슈퍼프레임의 주기는 활성구간과 비활성구간으로 구성된다. 그림에 나타난 상단의 파형(녹색)은 하단의 그래프(노란색)는 각각 코디네이터와 디바이스의 슈퍼프레임을 나타낸다.

OS의 내부의 소프트웨어 타이머 기반으로 구현한 OS 라이브러리인 delay_until_event() 함수를 사용한 첫 번째 디바이스의 경우에는 거의 모든 슈퍼프레임의 활성구간(Active Period)이 코디네이터의 활성구간보다 비정상적으로 큰 것을 볼 수 있다. 반면에 self_delay_until_event() 함수를 사용하는 두 번째 디바이스 상에서는 슈퍼프레임의 활성화 영역이 코디네이터의 슈퍼프레임과 높은 정합성을 보이며, 동기화가 순조롭게 이루어지고 있다는 점을 볼 수 있다.

이미 앞에서 언급한 바와 같이, 부하 면에서 두 엔드 디바이스의 차이는 애플리케이션의 소프트웨어 타이머 사용 여부이다. 실험환경의 설정에서 의도한 바처럼 애플리케이션 작업부하와 문맥교환 요구량 등에서는 거의 동일하다고 볼 수 있다. 한편, 데이터 송수신으로 인한 인터럽트는 비컨 프레임에 대해서만 존재하며, 이 두 노드는 프레임 송수신 면에서도 동일한 부하를 지닌다. 따라서 이번 실험의 결과는 소프트웨어 타이머 관리에 오버헤드의 누적으로 인한 내부 소프트웨어 타이머의 주기 증가 현상이 비컨 동기화에 부정적인 영향을 미칠 수 있다는 점을 분명히 확인시켜준다.

4.3 하드웨어 타이머 인터럽트 처리 지연 및 유실의 영향 평가

3절에서 하드웨어 타이머 인터럽트 처리 지연과 유실은 소프트웨어 타이머의 정확성을 저하시킨다는 점을 지적했다. 비컨 수신을 위해서는 정확한 시간에 수신기를 켜놓아야 한다. 따라서 소프트웨어 타이머가 부정확해지면 궁극적으로 비컨 동기화도 영향을 받을 수밖에 없다. 이번 실험에서는 그러한 시나리오를 실험을 통해 검증한다.

이 실험을 위해서는 타이머 인터럽트의 처리

Table 2 Beacon Reception Rate (BO=10, SO=9)

전송 주기 (msec)	Beacon Compensation Offset (msec)				
	Fixed size			SD-dependent size	
	0	50	100	SD/32 symbol =245	SD/16 symbol =491
100	50.25%	49.91%	98.68%	99.33%	99.33%
50	50.33%	58.47%	99.33%	99.33%	99.00%
10	50.08%	54.15%	63.02%	85.47%	99.00%

지연이나 유실이 발생할 수 있는 환경의 구축이 요구된다. 이런 환경은 실환경과 근접한 경우가 더욱 바람직하므로 데이터 전송 부하를 통해 구축했다. 일반적으로 이웃 노드에 프레임을 전송하면, 수신 디바이스는 곧바로 송신 디바이스에 MAC 계층 ACK(Acknowledge) 메시지를 전송한다. 수신된 ACK 메시지는 메시지 수신 인터럽트를 통해 처리되므로 ACK 인터럽트의 빈도는 데이터 전송 부하의 크기에 비례한다. 이러한 인터럽트의 처리 빈도의 증가는 상대적으로 타이머 인터럽트의 처리 지연이나 유실의 원인으로 작용한다.

실험을 위한 네트워크는 코디네이터와 엔드 디바이스의 한 쌍으로 구성되는데, 엔드 디바이스는 활성구간에 코디네이터로 127바이트 크기의 MAC 데이터 프레임을 주기적으로 전송한다. 전송주기는 데이터 전송 부하를 나타내며, 10, 50, 100msec의 세 가지 값을 사용했다. BO와 SO의 값은 각각 10과 9로 설정하였으며, 각 부하 상황에서 코디네이터와 디바이스 간의 비컨 동기화 정도를 측정하였다.

이 실험 결과로 인터럽트 부하가 가장 큰 경우인 10msec 주기에 엔드 디바이스에서 비컨을 유실하는 상황이 자주 발생하였다. Fig. 7은 비컨의 유실로 인한 동기화 실패의 경우의 슈퍼프레임의 파형 사례를 보여준다. 이 그림에는 두 개의 파형이 나타나 있는데, 각 파형은 슈퍼프레임의 시계열 전개를 나타낸다. 한 가지 유의할 점은 이전 그림과는 달리 상단 파형(녹색)과 하단의 파형(노란색)은 각각 디바이스와 코디네이터의 슈퍼프레임을 나타낸다. 이 그림은 코디네이터에서 발생하는 5개의 비컨 중에서 2번과 4번 비컨은 엔드 디바이스에서 수신에 실패한 경우를 나타내고 있다. 디바이스의 슈퍼프레임은 1번 슈퍼프레임의 종료로 활성구간에는 ON 비활성구간에 OFF 상태로 변경된다. 하지만, 2번 비컨의 경우, 비컨 수신의 실패로 수신기가 계속 ON 상태를 그대로 유지하다가 3번 비컨의 수신으로 정상적인 슈퍼프레임을 구성하는 것을 볼 수 있다.

5. 비컨 대기시간 보정기법을 통한 비컨 동기화 성능 향상

앞에서 소프트웨어 타이머의 정확성 결여로 인해 비컨 동기화가 실패할 수 있다는 점을 실험을 통해 검증했다. 여기서는 이러한 상황에서 비컨 동기화의 신뢰성을 높이기 위한 비컨 대기시간 보정기법을 제안한다. 실험을 통해 이 기법이 비컨 동기화의 신뢰성 향상에 기여하는 정도를 측정한다.

비컨 대기시간 보정기법은 비컨을 수신하는 디바이스에서 비컨 대기시간을 예정 시간보다 보다 빨리 시작하는 방식을 말한다. 이전에 설명했듯이 디바이스가 비컨 수신을 대기한다는 것은 라디오 송수신기를 수신모드로 설정해 놓는 것을 뜻한다. 당초 대기시간과 실제로 설정하는 대기시간 간의 차이를 비컨 보정 오프셋(Beacon Compensation Offset)이라 부르며, 이 값이 100msec이면 디바이스가 비컨 대기를 위한 수신모드 설정을 당초 비컨 대기시간보다 100msec 빨리 설정하게 된다. 이 실험에서는 전송부하를 변화시키며 비컨 보정 오프셋 값의 크기에 따른 비컨 수신율을 측정하였다. 전송부하는 4.3절의 실험과 같이 127바이트의 MPDU 프레임을 각각 10, 50, 100msec 주기로 엔드 디바이스가 코디네

이터에게 데이터를 전송하는 세 가지 종류로 구성하였다.

한편, 비컨 보정 오프셋의 크기에는 두 가지 종류를 사용했는데, 하나는 단순 고정값으로 0, 50, 100msec을 사용했으며, 다른 하나는 슈퍼프레임 길이에 비례하는 값을 사용했다. 후자의 경우는 SO 값에 따라 결정이 되는 슈퍼프레임의 주기가 수십 밀리 초부터 수백 초에 이르며 타이머 오차는 그 크기에 따라 함께 커지므로 비컨 보정 오프셋 값도 고정된 크기보다는 슈퍼프레임 주기의 크기에 따라 함께 증가시키는 것이 필요하다는 점을 고려했다. 여기서는 SD(Superframe Duration)의 1/32(=245msec)과 1/16(=491msec)의 비컨 보정 오프셋에 대해 실험을 수행했다.

Table 2는 실험환경별 엔드 디바이스의 비컨 수신율을 보여준다. 우선, 전송 부하의 크기와 비컨 보정 오프셋의 관계를 살펴볼 필요가 있다. 앞 소절의 타이머의 정확성에 관한 논의를 통해 비컨 수신율은 데이터 전송부하가 클수록 감소할 것으로 예측할 수 있다. 이 실험에서도 비컨 보정 오프셋 값의 크기에 상관없이 데이터 전송 부하가 커질수록 비컨 수신율은 낮아지는 것을 볼 수 있다. 한편, 비컨 수신율은 비컨 보정 오프셋이 커지면 당연히 커질 것이란 점도 예측할 수 있다. Table 2에서 보는 바와 같이 실제로도 고정 크기 대기 오프셋이나 슈퍼프레임 길이에 비례하는 값 모두 그 값이 커질수록 비컨 수신율이 높아진다.

이 절을 마치기 전에 한 가지 주목해야 할 점이 있다. 비록 비컨 보정 오프셋의 크기를 늘리면 비컨 동기화의 정확도를 증가시킬 수 있지만, 반대로 비컨 대기시간의 증가에 따른 전력의 소모도 수반된다는 점이다. 실제로 BO와 SO가 각각 10과 9인 본 실험의 설정상태에서는 비활성화 구간이 7.864sec이므로 100msec의 비컨 보정 오프셋을 적용했을 때 에너지 소모 증가율은 약 1.28%가 된다. 이 절의 실험은 비컨 보정 오프셋 기법이 비컨 동기화의 신뢰성을 향상시킬 수 있다는 점을 실증하는 데에 있으므로 BO와 SO 각각 10과 9에 대해서만 비컨 수신율을 측정했다. 하지만 Table 1에서 볼 수 있듯이 비컨주기(BO)와 슈퍼프레임(SO) 설정값에 따라 오차가 3msec에서 265msec까지 각각 다른 값을 나타내므로 비컨 보정 오프셋은 각각 그 값에 맞게 적절하게 조정하면 된다.

6. 결 론

일반적으로 경량 MCU를 장착한 임베디드 디바이스 상에서는 하드웨어 타이머의 수적 제약으로 운영체제의 소프트웨어 타이머를 사용해서 네트워킹 스택을 구현하는 것이 일반적이다. 이러한 경우에 시분할 동기적 통신의 신뢰성은 소프트웨어 타이머의 오차에 매우 취약할 수 있다. 하위 하드웨어 타이머의 처리 지연이나 유실, 소프트웨어 타이머 관리 오버헤드 등이 소프트웨어 타이머의 오차 발생의 주된 요인이다. 이 논문에서는 IEEE 802.15.4 비컨 모드에 대해 그러한 비컨 동기화의 신뢰성 저하 현상을 실험을 통해 검증하였다. 아울러 비컨 대기시간 보정기법을 적용해서 이러한 상황을 개선할 수 있다는 점도 실험을 통해 확인하였다. 하지만, 전력 소모 증가때문에 비컨 대기시간 보정이 비컨 동기화의 근본적 해결 방안은 될 수 없다. 향후 OS 내의 소프트웨어 오버헤드를 극소화한 전용 글로벌 타이머를 바탕으로 한 IEEE 802.15.4 비컨 동기화의 신뢰성 개선 방안에 대한 후속연구가 요구된다.

References

[1] IEEE Std 802.15.4-2011: Part 15.4: Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs), IEEE Computer Society, 2011.

[2] Kim, H., et al., “Experimental Research Testbeds for Large-Scale WSNs: A Survey from the Architectural Perspective,” International Journal of Distributed Sensor Networks, Vol. 11, No. 3, Article No. 2, 2015.

[3] Park, J., et al., "A Development Plan for Co-creation-based Smart City through the Trend Analysis of Internet of Things," Journal of the Korea Industrial Information Systems Research, Vol. 21, No. 4, pp. 67-78, 2016.
[4] Sethi, P., and Sarangi, S., "Internet of Things: Architectures, Protocols, and Applications," Journal of Electrical and Computer Engineering, Article ID 9324035, 2017.
[5] Yoo, S., "A Software Framework for Verifying Sensor Network Operations and Sensing Algorithms," Journal of the Korea Industrial Information Systems Research, Vol. 17, No. 1, pp. 53-60, 2012.
[6] Gonzalez, S., et al., "The Sticking Heartbeat Aperture Resynchronization Protocol," 26th International Conference on Computer Communication and Networks, pp. 1-8, 2017.
[7] Khoufi, I., et al., "Beacon Advertising in an IEEE 802.15.4e TSCH Network for Space Launch Vehicles," 7th European Conference for Aeronautics and Aerospace Science (EUCASS), 2017.
[8] Wu, Y., et al., "Clock Synchronization of Wireless Sensor Networks," IEEE Signal Processing Magazine, Vol. 28, No. 1, pp. 124-138, 2011.
[9] Kim, T. and Ahn, K. "Mitigating Hidden Nodes Collision and Performance Enhancement in IEEE 802.15.4 Wireless Sensor Networks," Journal of Korea Information Processing Society, Vol. 4, No. 7, pp. 235-238, 2015.
[10] Stanislowski, D., et al. "Adaptive Synchronization in IEEE802. 15.4 e networks," IEEE Transactions on Industrial Informatics, Vol. 10. No 1, pp. 795-802, 2014.
[11] Bernhard H., et al, "Timing Synchronization of Low Power Wireless Sensor nodes with Largely Differing Clock Frequencies and Variable Synchronization Intervals," 20th International Conference on Emerging Technologies & Factory Automation(ETFA), pp. 1-7, 2015.
[12] Nadas. P., et al., "Energy Efficient Beacon based Synchronization for Alarm Driven Wireless Sensor Networks," IEEE Signal Processing Letters, Vol. 23, No. 3, pp. 336-340, 2016.
[13] Severino, R., et al., "An Open-source IEEE 802.15.4 MAC Implementation for TinyOS 2.1," European Conference on Wireless Sensor Networks, 2011.
[14] Kim, H. and Yoo, S., "Implementation and Analysis of IEEE 802.15.4 Compliant Software based on a Vertically Decomposed Task Model," Journal of the Korea Industrial Information Systems Research, Vol. 19, No. 1, pp. 53-60, 2014.

김 희 철 (Hiecheol Kim)

- 정회원
- 연세대학교 전자공학과 학사
- 남가주대학교 컴퓨터공학과 석사
- 남가주대학교 컴퓨터공학과 박사
- 대구대학교 정보통신대학 정보통신공학부 교수
- 관심분야 : 임베디드 OS, 무선센서네트워크, 머신러닝