[소프트웨어학과]

TinyOS 기반의 저전력 네트워크 프로토콜 LPP 구현 및 Broadcasting 효율성 연구

[결과보고서]

Version 1.0
2016.06.03

학번 : 201323158
이름 : 정선교
지도교수 : 고정길

목 차

<그림 목차>

요약

A power of embedded board is used efficiency in wireless embedded network. Most consuming part of power is Radio for data transmission. So treated as a core to control the radio properly in wireless embedded network environment and low power protocol provide it. The low power protocol such as LPL and LPP carry out low-power operation for a variety of customized to the environment.

The following is the algorithm of LPP. If there is no data to send, send Probing. If receiver receive Ack for probing, wait for the data leaving radio turn on. After receive data, turn off the radio and back to the owns cycle. If receiver doesn't receive Ack for probing, turn off the radio and back to the owns cycle. If there is data to send, listening for probing instead of sending probing. In other words, waiting for receiver to get the data. If there is receiver, notify to wait for data by sending Ack of probing. After sending ack successfully, send data immediately. Receiver and Sender works by 1:1 here.

I feel the limitation in that the exchange of data 1:1. It must transmit to a receiver at a time in sending data to all and is inefficient. Finally I devised the way to sending to all efficiency. The differences between original LPP and Broadcasting LPP are when sender send ack of probing, send data that you must wake up after some time. When receiver receive ack, it turn off the radio and wake up after some time. And then all receiver receive data at the same time.

1 개요

1.1 배경

임베디드 보드들과의 통신을 위해 유선 네트워크와 무선 네트워크가 형성되어 있다. 하지만 유선 네트워크는 임베디드 보드들간에 직접적으로 연결이 되어야 하고, 중간에 줄이 끊어지지 않도록 해야 하며 데이터 전송이 잘 되는 줄의 속성 및 길이를 고려해야 하는 등 현실적으로 제한이 많고 비효율적인 측면이 있어 무선 네트워크를 더 많이 사용하게 된다.

무선 임베디드 네트워크에서는 임베디드 보드가 가지고 있는 한정된 전력을 효과적으로 사용해야 한다. 전력은 곧 임베디드 보드의 수명을 뜻하며 실제로 보드들을 적절한 환경에 설치하고 방치해두며 데이터를 수집한다. 따라서 오랫동안 안정된 네트워크를 통해 데이터를 수집하기 위해서는 전력을 효과적으로 사용해야 한다. 통신을 하는 데 있어 전력이 제일 많이 소모되는 부분은 Radio 를 키는 부분이다. 즉, Radio 를 킨다는 말은 데이터를 전송하기 위해 전력을 사용한다는 것이다. Radio 를 적절히 제어하는 것이 임베디드 무선 네트워크 환경에서 가장 핵심적으로 다루어지는 부분이다. 따라서 다양한 응용상황에서 전력을 적게 사용하면서 최적의 결과를 도출할 수 있도록 하는 통신 방법을 저전력 프로토콜이라고 한다.

저전력 프로토콜에는 Low Power Listening 과 Low Power Probing 와 같은 기법이 존재하며 이를 통해 다양한 환경에 맞춤형으로 저전력 운용을 수행한다. 특히 LPP 는 수신자가 깨어있을 때까지 기다렸다가 수신자의 깨어남과 동시에 패킷을 교환하고 데이터 교환을 수행하는 방식으로 구성되어 있다. 이를 이용하여 적합한 환경에 응용될 수 있도록 TinyOS 를 기반으로 구현하고 다양한 환경에 최적한 시킬 수 있는 운용 방식 등에 대한 연구를 진행한다.

1.2 목표

무선 임베디드 네트워크에서 중점적으로 고려해야 하는 것은 전력을 얼마나 효율적으로 사용하는 가이다. 이를 해결하기 위해 저전력 네트워크 프로토콜이 존재하며 이를 구현하는 것이 목표이다. 저전력 네트워크 프로토콜에는 앞서 말했듯이 LPL 과 LPP 가 있으며 본 연구에서는 TinyOS 를 기반으로 한 LPP 구현을 수행한다. LPP 프로토콜을 구현이 완성되면 데이터를 전송하는 것 중에서도 broadcasting 해야 하는 상황에서의 효율성을 극대화 시키기 위한 연구를 진행한다. Broadcasting 해야 하는 상황을 예로 들자면 데이터 패킷이 모든 노드들에게 전송이 되어야 하는 환경이다 즉 특정 노드에게만 데이터 전송을 하는 것이 아니라 broadcast 를 통해 모든 노드들에게 같은 데이터를 전송을 해야 할 때 기존의 LPP 로 수행하면 전력을 낭비하게 된다. 이를 보완하기 위해 Receiver 노드들로부터 probe 를 받은 후 여러 개의 Receiver 노드들이 같은 시간에 일어나게 하여 한 번에 데이터 전송을 하는 프로토콜을 구축하는 것이 본 연구의 최종 목표이다.

2 관련 선행 연구 조사

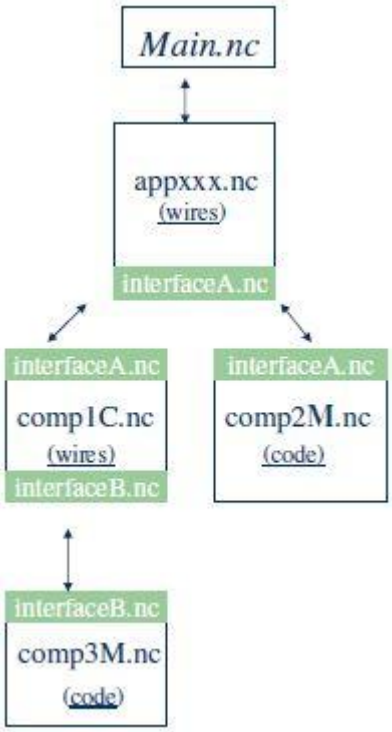
2.1 TinyOS

① Interfaces (xxx.nc)

- 함수들을 구체화한다.
- command 가 불려질 수 있다.
- events 가 다뤄져야 한다.

② Software Components

- Module (xxxM.nc) : code implementation 과 interface 함수에 대한 code 가 포함되어 있다.
- Configuration (xxxC.nc) : components 들을 linking 하고 writing 한다.



[Figure 1] Basic Structure

위의 구조는 TinyOS 에서 코드를 구현하는 것의 기본적인 구조이다. [1],[2]

2.2 Tmote Sky(Telosb)

- 임베디드 보드의 일종이다.
- IEEE 802.15.4 Radio 사용한다.
- TI MSP430 microcontroller
- 안테나와 USB interface 장착되어 있다.
- 저전력을 활용한다.

- 전력과 사이즈, 센서 노드들의 비용이 넓은 범위의 무선 센서 네트워크에서는 적절하지 않다. [2]

[Figure2] Telosb [2]

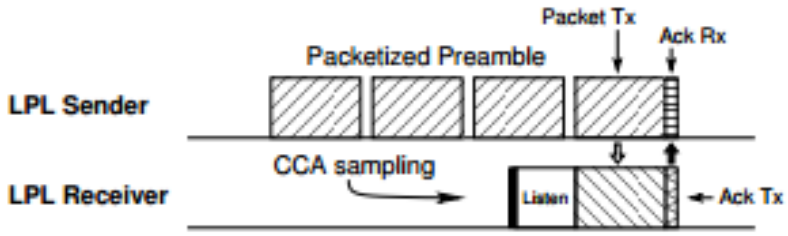
2.3 LPL

LPL 은 Low Power Listening 의 약자로 event 가 많이 발생하는 환경에서 많이 사용되는 프로토콜이다. 예를 들자면 일본은 지진이 많이 발생하기 때문에 이런 지역에서 지진에 관한 데이터 수집을 원할 때 LPL 을 적용한 임베디드 보드를 설치할 수 있다. LPL 은 보낼 데이터가 있으면 Receiver 가 깨어있는지 여부에 상관없이 Receiver 의 Ack 가 올 때 까지 계속 데이터 패킷을 전송한다. 만약 Receiver 가 깨어있고 데이터 패킷을 받으면 그에 따른 Ack 를 보내고 Sender 는 Ack 를 받으면 그 때 데이터 전송을 멈춘다.

node 들이 주기적으로 event 에 대한 sign 을 사용하고 event 를 발생을 알리기 위해 긴 preamble 을 전송하는 데 있어서 채널을 사용하는 LPL 은 non-synchronized node 들이 깨어나는 데 매력적인 대안을 제공한다. LPL 은 radio 를 기반으로 한 패킷을 적용하고 있고, preamble 은 끊임없는 패킷 스트림으로 구성되어 있다.

LPL 은 전체 네트워크가 동시에 깨는 것이 아니라 Receiver 가 깨어있는지 여부에 상관없이 보낼 데이터가 있으면 각 노드들이 주기적으로 깨는 방식으로 설계되어 있다. LPL 을 사용하는 동안 broadcast 모드에서 최대 길이의 preamble 전송이 필요 하며, 너무 많은 패킷이 존재하면 데이터를 수집 하는 데 있어 지연이 될 수 있다. 그 이유는 unicast 와는 달리 sender 는 receiver 가 깨어있는 순간을 모르고 Receiver 가 받을 때 까지 데이터를 주기적으로 보내기 때문이다. 그렇기 때문에 preamble 의 전송이 일찍 끝날 수 없고 만약 정확한 preamble 을 받지 못했다면(false negative) 엄청난 위험이 초래할 수 있다.

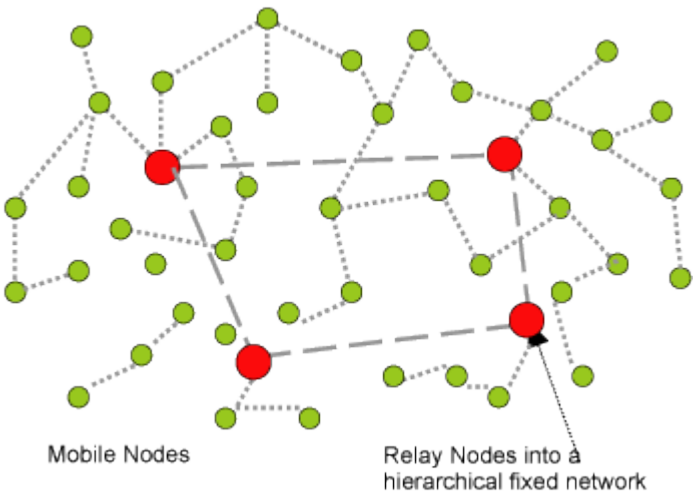
LPL 은 Receiver 가 없더라도 데이터 패킷을 전송하기 때문에 전력 낭비가 된다. 하지만 LPL 이 사용되는 전제 조건이 event 가 많이 발생하는 곳이기 때문에 Receiver 가 충분히 많고 sender 가 보낼 때마다 깨어있다고 가정한다면 sender 는 데이터 패킷이 있을 때마다 즉각적으로 전송을 완료할 수 있어서 장점이 될 수 있다. 아래 그림은 LPL 을 그림으로 묘사한 것이다.



[Figure 3] LPL [3]

2.4 Wireless Mesh Network(WMN)

Mesh 네트워크는 'flooding' 이나 'routing' 기술을 사용하여 메세지를 전달한다.[5] 무선 mesh 네트워크는 mesh 라우터들과 메시 클라이언트라는 노드들로 이루어진 네트워크이다. mesh 라우터들이 무선 mesh 네트워크의 핵심을 이룬다.각 mesh 노드들은 1 개 이상의 매개체를 거쳐 온 데이터들을 무선 전송이 되는 한도 내에서 이웃 라우터나 클라이언트에게로 전달한다.[4] 라우팅 기술을 사용하면 메세지는 목적지 노드에 전달될 때가지 노드에서 노드로 hopping 하여 전파된다. 이러한 경로의 가용성을 확실히 하기 위해서 Shortest Path Bridging 같은 self-healing 알고리즘을 사용하여 네트워크는 계속적인 연결과 연결이 끊어지거나 새롭게 연결되는 것에 대해 mesh 네트워크 구조를 재구성 해야 한다. [5],[6],[7]



[Figure 4] Mesh Network [7]

3 연구 결과

3.1 문제 정의

3.1.1 문제 제기

임베디드 보드로부터 데이터를 전송 받아 분석하고 더 나은 환경을 구축하기 위해서는 유선 또는 무선 통신 방법이 필요하다. 유선 네트워크는 현실적으로 문제가 많고 실제 사용하는 빈도수가 낮으므로 유선 보다는 무선 네트워크의 개발이 필수적이다. 더욱이 무선 네트워크를 사용하는 IoT 분야의 급부상으로 인해 무선 임베디드 네트워크에 대한 개발이 시급하다. 하지만 현재 무선 임베디드 네트워크에 대해 다양한 프로토콜이 존재하지 않고 있다. 이에 무선 네트워크 중에서도 오랜 기간 효율적으로 임베디드를 관리하기 위한 저전력 프로토콜에 대한 연구가 필요하다. 저전력 프로토콜인 LPL 과 LPP 중에서도 LPP 에 대한 연구를 진행한다.

3.1.2 연구의 필요성

LPP 프로토콜은 Receiver 로 부터 probe 가 오면 각 노드들에 대해 데이터 패킷을 보내는 방식으로 구현되어있다. 앞서 말했듯이 무선 임베디드 네트워크에서는 전력을 효율적으로 사용하는 것이 중요한데 만약 모든 노드들에 대하여 데이터 패킷을 보내야 하는 경우 문제가 발생한다. 즉 특정 노드들이 아닌 모든 노드들에 대해 데이터 패킷을 보내야 한다면 Receiver 로 부터 probe 가 올 때마다 패킷을 전송해야 한다. 이러한 상황에서는 전력을 효율적으로 사용하지 못하게 되며 무선 임베디드 네트워크의 기본 목표에 위배된다. 따라서 이러한 문제를 해결하여 보다 더 다양한 환경에서 적용되기 위해 본 연구를 진행하게 되었다.

중간보고서 이후 연구의 주제가 수정이 되었다. 중간보고서에서는 LPP 를 구현한 후 LPL 과 LPP 를 결합한 구조를 구축하고자 하였지만, LPP 에 관해 연구를 진행하던 중 위와 같은 문제점을 발견하게 되었으며 LPL 과 결합하기 전 LPP 자체만으로 완벽한 프로토콜을 구현하기 위해 연구의 방향을 변경하게 되었다.

3.1.3 기존 연구와의 차이점

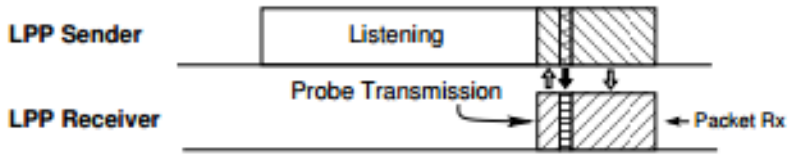
앞서 조사한 WMN 는 linked list 처럼 노드들이 연결이 되어있기 때문에 worst case 가 발생할 수 있다. 예를 들어 source node 와 destination node 들간의 거쳐야 할 노드들이 많다면 시간이 낭비되고 중간 노드들은 필요 없는 데이터를 전송 받게 된다. 또한, destination 노드로 데이터를 전송하는 도중 데이터가 손실되거나 손상된다면 수 많은 노드들을 거쳐 다시 데이터를 요청해야 한다. 이를 보완하기 위해 저전력 네트워크의 일종인 LPP 기법을 제안한다.

3.2 제안하는 기법 혹은 소프트웨어 구조

3.2.1 LPP

LPP 는 Low Power Probing 의 약자로 LPL 에 사용되는 환경에서보다 event 가 발생하는 빈도수가 적은 환경에서 사용된다. 예를 들면 한국은 일본에 비해 지진이 적게 발생하므로 이러한 지역에서 지진에 관한 데이터 수집을 원할 때 LPP 를 적용한 임베디드 보드를

설치할 수 있다. 기존의 LPL 에서 사용하던 방식을 사용하면 비효율적으로 전력을 낭비하기 때문에 LPP 같은 새로운 방식으로 접근한 것이다. LPP 는 LPL 과는 다르게 보낼 데이터가 있어도 바로 패킷을 전송하지 않고 Receiver 가 존재할 때만 전송을 한다. Receiver 의 존재를 확인 하는 방법은 Receiver 는 깨어남과 동시에 Probe 를 보냄으로써 Receiver 의 존재를 알 수 있다. Probe 패킷은 데이터 패킷보다 짧은 패킷이며 가짜 데이터 패킷이라고 생각할 수 있다. Receiver 가 Probe 패킷을 보내면 sender 는 그에 대한 Ack 를 보내며 해당 Receiver 에게 데이터 패킷을 보낸다. Receiver 는 Sender 가 보낸 데이터 패킷을 받으면 그에 대한 Ack 를 보내게 된다. Sender 는 데이터를 보낸 후 Receiver 로 부터 데이터 패킷을 잘 받았다는 Ack 를 기다리고 있는데 만약 오지 않았다고 하더라도 데이터 패킷을 재전송하지 않고 자신의 주기가 끝나면 그대로 데이터 전송은 실패하게 된다. LPL 은 어떤 Receiver 가 깨어있는지 모르기 때문에 전체 노드에 broadcast 하게 된다. 하지만 LPP 는 Probe 가 온 노드에 대해서만 데이터 전송을 하게 된다. LPP 는 보낼 데이터가 없으면 Receiver 가 되는데 주기적으로 깰 때마다 Probe 를 보내고 그에 대한 Ack 가 오지 않으면 주위 노드들로 부터 받을 데이터가 없다는 뜻이기 때문에 그대로 Radio 를 끔으로써 전력 낭비를 방지하게 된다. Sender 또한 보낼 데이터가 있더라도 Receiver 가 없다면 데이터 전송을 하지 않기 때문에 LPL 에서 sender 가 깰 때마다 데이터를 전송하면서 발생하는 비효율적인 전력 낭비 문제를 방지할 수 있다. 아래 그림은 LPP 의 데이터 교환 방법을 나타내는 그림이다.



[Figure 5] LPP [3]

아래는 LPP 에 대한 알고리즘이다.

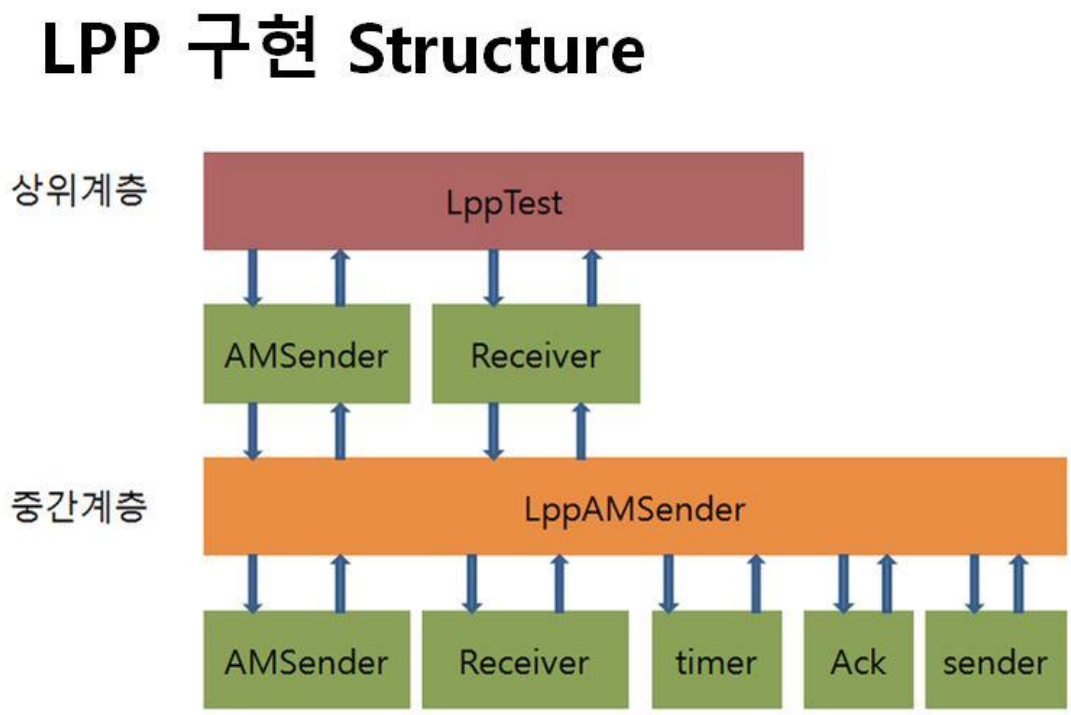
```
Algorithm 1 Lower Power Probing
Procedure PeriodicWakeUp
  loop
      TurnRadioOn
      If HaveDataToSend then
        ListeningProbe
        If ReceiveProbe then
          SendAckForProbe
          SendData
      Else SendProbe then
        If ReceiveAckForProbe then
          If ReceiveData then
            SendAckForData
      TurnRadioOff
```

[Figure 6] LPP Algorithm

아래 structure 는 주요 부분을 간추려서 묘사한 그림이다. 상위계층과 중간계층으로 나뉘며, 상위계층에서 데이터가 있을 때마다 send 를 하면 중간계층에서 데이터를 처리한다. 상위계층은 단순히 데이터를 보내는 것만 수행하고 중간계층에서 어떤 프로토콜을 사용하여 데이터를 받고 Radio 의 On/Off 모드를 조절할 수 있도록 설계하였다. 상위계층에서 해결하지 않고 중간계층을 따로 만들어준 이유는 LPP 만을 구현하면 상관없지만 만약 다른 프로토콜을 적용하려거나 데이터를 주고 받는 형태를 변경하려면 상위계층 모두를 다시 설계해야 하기 때문이다. 중간계층을 따로 만들어주면 그러한 수고를 할 필요 없이 중간계층의 설계만 변경해주거나 새로운 파일을 linking 해주면 된다.

상위계층과 중간계층에서 데이터를 전송할 때 AMSender 를 사용하며 데이터를 받을 때는 Receiver interface 를 사용한다. 상위계층에서 데이터를 전송하려고 하면 중간계층에 있는 AMSender 로 내려와서 LPP 프로토콜의 알고리즘에 맞추어 데이터를 전송한다. 쉽게 말하자면 상위 계층은 중간 계층에서 일어나는 일을 모르고 데이터를 중간계층에 넘겨주기만 하며, 중간계층은 해당 프로토콜 알고리즘에 맞추어서 데이터를 전송한다.

LPP 를 구현하는데 TImer 를 따로 관리하여 Timer 에서 Radio 의 on/off 를 관리하도록 해주었다. 데이터와 probe 의 데이터 크기를 다르게 하여 probe 의 크기를 작게 만들어 실제 전송하려는 데이터와 probe 를 구분시켜 주었다.



[Figure 7] LPP Structure

3.2.2 효율적 Broadcasting LPP

모든 노드들에게 데이터 패킷을 보내야 하는 상황에서는 기존의 LPP 기법을 사용하면 오히려 전력을 낭비하게 된다. 이러한 상황을 보완하기 위해 Receiver 의 probe 가 올 때마다 데이터 패킷을 전송하는 것이 아니라 Receiver 들이 일정 시간 이후 동시에 깨도록 timer 조정을 한다. Receiver 들이 깨는 순간 Sender 는 broadcasting 으로 데이터 패킷을 전송하게 되면 전력을 기존의 LPP 방식에서 보다 더 효율적으로 사용할 수 있게 된다. 아래 그림은 이 프로토콜의 데이터 교환 방식을 나타내는 그림이다.

[Figure 8] Broadcast data at once

Structure 는 위에 나타난 기존 LPP structure 와 동일하며 알고리즘만 다르다. 효율적 broadcasting 에 대한 LPP 알고리즘은 아래와 같다.

Algorithm 2 Efficient Broadcasting LPP

```
Procedure PeriodicWakeUp
 loop
      TurnRadioOn
      If HaveDataToSend then
        ListeningProbe
        If ReceiveProbe then
          SendAckForProbe
          SendWaitingMsg
          If AfterSomeTime then
            SendData
       Else SendProbe then
         If ReceiveAckForProbe then
           TurnRadioOff
          If AfterSomeTime then
            TurnRadioOn
            ReceiveData
      TurnRadioOff
```

[Figure 9] Efficient Broadcasting LPP Algorithm

4 성능 분석

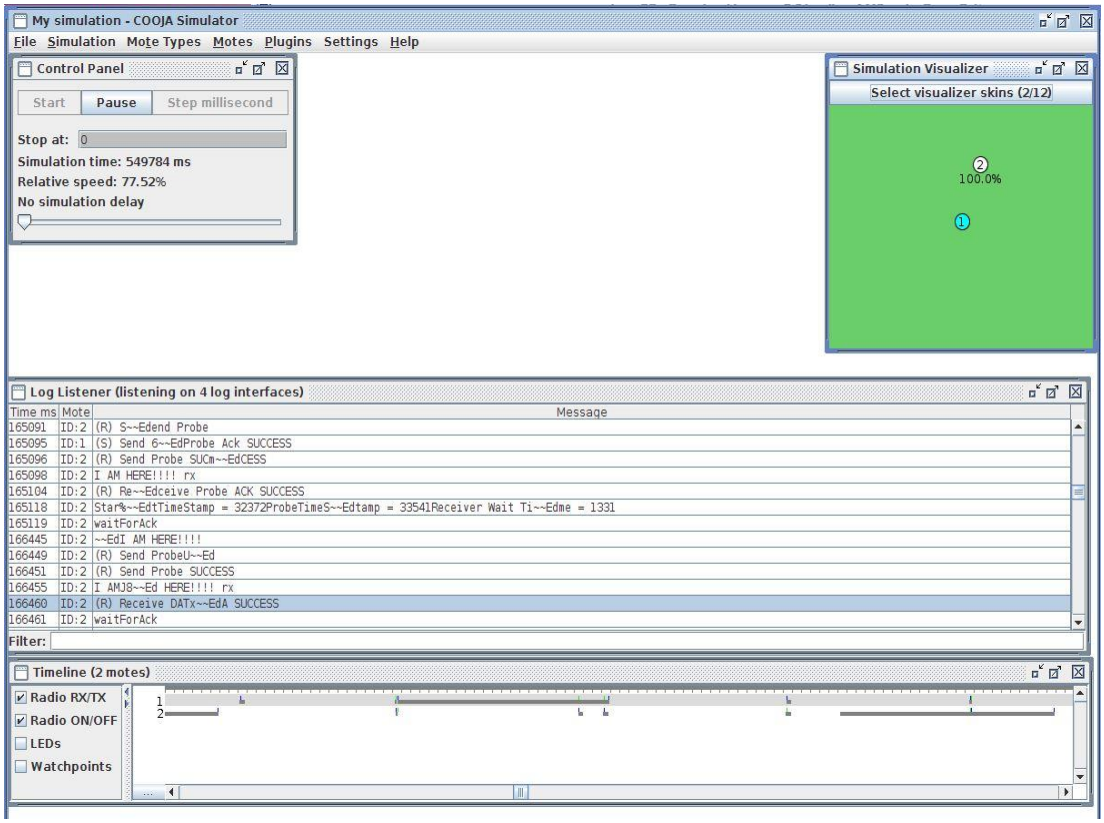
4.1 성능 분석 환경

4.1.1 Contiki

Instance Contiki 는 전체적인 contiki 개발 환경이며 VMWare 에서 운용되는 Ubuntu Linux virtual machine 이다. instance contiki 는 contiki 를 가지고 있으며 contiki 개발 환경에서 사용되고 있는 모든 개발 도구, 컴파일러 그리고 시뮬레이터들을 가지고 있다. Contiki 는 하드웨어에 직접적으로 contik 를 쉽게 운용할 수 있게 만들어주는 build system 을 가지고 있다. build system 은 서로 다른 하드웨어 플랫폼에서 동일하게 설계되어 있어 하드웨어를 전활 할 때 명령을 잘 알 수 있도록 한다. [8]

4.1.2 Cooja

Cooja 는 contiki 네트워크 시뮬레이터이다. 쿠자는 contiki 보드들 간의 크고 작은 네트워크를 허용한다. 보드들은 하드웨어 레벨 또는 덜 정확한 레벨에서 수행될 수 있다. 하드웨어 레벨이란 느리지만 시스템 behavior 의 정확한 측정이 되며, 덜 정확한 레벨에서는 빠르고 큰 네트워크 상에서 허용이 된다. cooja 는 개발자들이 자신의 코드와 타겟 보드에 실제 구축하기 전 system 을 테스트해볼 수 있기 때문에 contiki 개발환경에서 매우 유용하게 사용되는 tool 이다. [8]



[Figure 10] Cooja simulator

cooja 에서 simulation 하기 위해 수행하는 시나리오는 다음과 같다.

1. terminal : ~/contiki/tools/cooja/ 로 이동한다.
2. ant run 을 사용하여 cooja 를 실행시킨다.
3. [File]-[New simulation]-[Create] 하여 새로운 시뮬레이션을 만든다.
4. [Motes] - [Create new mote type] - [Sky mote] 를 선택하고 contiki process/Firmware 에서 Browse 버튼을 통해 실행하고자 하는 코드가 있는 폴더로 이동한다.
5. 폴더의 main.exe 파일을 open 한다.
6. Create 버튼을 누르면 simulation 이 생성이 된다.
7. Simulation visualizer 창을 통해 motes 의 거리를 조정할 수 있으며 환경 변수를 조정할 수 있다.

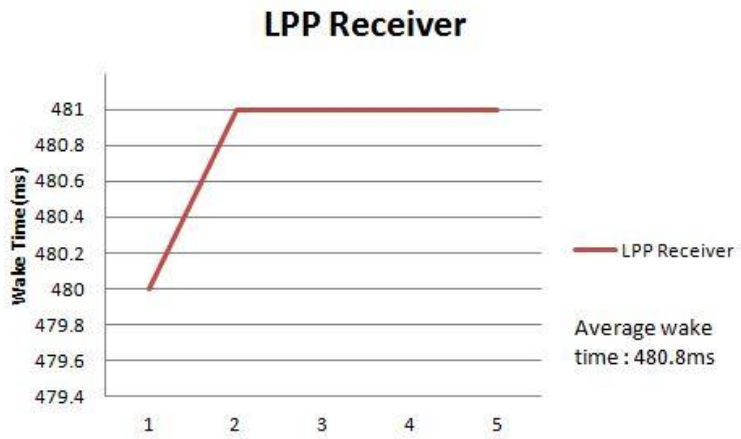
8. control panel 을 통해 실행 속도를 조정할 수 있으며 Timeline 에서 동작을 확인할 수 있다.
9. Log Listener 에서는 코드 상에서 printf 되는 것을 확인할 수 있다.
10. [Motes]에서 mote 들을 추가하거나 삭제할 수 있다.

4.2 성능 분석 결과

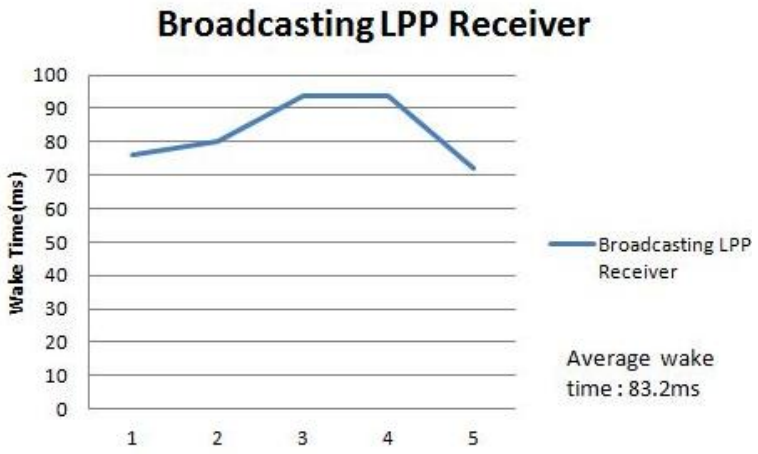
위에서 설명한 Cooja 를 이용하여 결과를 도출하였다. 기존의 LPP 의 경우 Receiver 는 깨어남과 동시에 Probing 을 하고 Probing 에 대한 Ack 가 있으면 데이터 패킷을 기다린다. Ack 가 없으면 라디오를 끈다. Sender 는 깨어남과 동시에 자기 주기 동안 Receiver 의 Probe 를 기다린다. Probe 를 받으면 Ack 를 보내고 데이터를 보낸다.

Broadcasting LPP 의 경우 Receiver 는 깨어남과 동시에 Probing 을 하며 Probing 에 대한 Ack 가 오면 라디오를 끄고 일정시간 이후에 일어나 데이터를 보낸다. Ack 가 없으면 라디오를 끈다. Sender 는 깨어남과 동시에 자기 주기 동안 Receiver 의 Probe 를 기다린다. Probe 를 받으면 Ack 를 보내고 바로 데이터를 보내는 것이 아니라 일정 시간 이후에 데이터를 보낸다. 일정 시간 이후에 보내는 이유는 모든 Receiver 가 동시에 깨어나게끔 timer 를 조정하여 데이터를 1:1 로 보내는 것이 아니라 한 번에 모든 Receiver 에게 보내기 때문이다.

Broadcasting LPP 와 기존의 LPP 의 차이점은 Receiver 가 깨어있는 시간이다. 아래 도표에서 볼 수 있듯이 LPP 의 Receiver 는 probe 를 보내고 data 를 받을 때까지 깨어있는데 평균 480.8ms 동안 깨어 있는다. Broadcasting LPP 의 Receiver 는 Probing 에 대한 Ack 를 받으면 잠시 라디오를 껐다가 다시 깨어나므로 깨어 있는 시간은 평균 83.2ms 이다. Probing Ack 를 받고 다시 깨는 시간 차이만큼 전력을 절약할 수 있다. 약 17.3%만큼 전력을 절약할 수 있다.



[Figure 11] LPP Receiver's wake Time



[Figure 12] Broadcasting LPP Receiver's wake Time

5 결론

무선 임베디드 네트워크에서는 임베디드 보드가 가지고 있는 한정된 전력을 효과적으로 사용해야 한다. 전력이 제일 많이 소모되는 부분은 데이터를 주고 받기 위해 사용하는 Radio 이다. 따라서 Radio 를 적절히 제어하는 것이 임베디드 무선 네트워크 환경에서 핵심적으로 다루어지는 부분이며 이러한 환경을 제공하는 것이 저전력 프로토콜이다. 저전력 프로토콜에는 LPL 과 LPP 와 같은 기법이 존재하며 이를 통해 다양한 환경에 맞춤형으로 저전력 운용을 수행한다.

LPP 알고리즘에 대해 설명하면 다음과 같다. LPP 에서 보낼 데이터가 없으면 Probing 을 보낸다. Probing 에 대한 Ack 를 받으면 Radio 를 켜둔 채 Data 를 기다렸다가 받는다. data 를 받으면 Radio 를 끄고 자기 주기로 돌아간다. Probing 에 대한 Ack 를 받지 못하면 Radio 를 끄고 다시 자기 주기로 돌아간다. LPP 에서 보낼 데이터가 있으면 Probing 을 보내는 대신 Probing 을 기다린다. 즉, Data 를 받을 Receiver 를 기다린다. Receiver 가 있으면 Ack 를 보냄으로써 radio 를 끄지 말고 데이터를 받도록 기다리라고 알려준다. Ack 를 성공적으로 보낸 후 바로 data 를 전송한다. 여기서 sender 와 receiver 는 1:1 로 작용한다.

1:1 로 데이터를 주고받는 점에서 한계를 느꼈으며 데이터를 모두에게 전송해야 하는 경우에는 한 번에 한 개의 Receiver 에게만 보내게 되므로 비효율적이다. 따라서 데이터를 하나의 Receiver 가 아니라 모든 Receiver 에게 전송 해야 할 때 효율적으로 보내는 방법을 고안하였다. 기존 LPP 와 다른 점은 Sender 가 Receiver 가 보낸 Probing 에 대해 Ack 를 보낼 때 일정 시간 후에 깨라는 데이터를 같이 보낸다. Receiver 는 Ack 를 받으면 Sender 가 보낸 data 만큼 radio 를 껐다가 다시 일어난다. 그리고 모든 Receiver 들이 동시에 data 를 받는다.

LPP 와 가장 밀접한 관계가 있는 LPL 과 비교해보면, LPP 는 데이터를 보내는 횟수가 적은 환경에서 효율적이며 LPL 은 데이터를 보내는 횟수가 많은 환경에서 효율적이다. LPL 은 기본적으로 Receiver 가 있든 없든 데이터를 무조건 보내기 때문이다. 이러한 기법은 Receiver 가 자주 깨어나는 상황에서 효율적이다. 반대로 LPP 는 Receiver 가 있을 때만 데이터를 보내기 때문에 LPL 과 대비하여 데이터를 보내는 횟수가 적으므로 상대적으로 Receiver 가 깨어나는 주기가 길다.

앞으로 LPL 과 LPP 를 결합한 형태의 프로토콜을 구축한다면 데이터를 보내는 횟수가 바뀌어도 상황에 따라 프로토콜이 자동으로 바뀌므로 더욱 더 효율적인 프로토콜이 될 수 있을 것 같다.

참고자료

[1] (October 27, 2006) Philip Levis, "TinyOS Programming"
[2] (2011), Greg Hackmann, "TinyOS Tutorial"
[3] Razvan Musaloiu-E, Chieh-Jan Mike Liang, Andreas Terzis, "Koala : Ultra-Low Power Data Retrieval in Wireless Sensor Network", International Conference on Information Processing in Sensor networks, 2008.
[4] https://en.wikipedia.org/wiki/Mesh_networking
[5]https://ko.wikipedia.org/wiki/%EB%AC%B4%EC%84%A0_%EB%A9%94%EC%8B%9C_%EB%84%A4%ED%8A%B8%EC%9B%8C%ED%81%AC
[6] (January, 5, 2012) Digi-Key's european eitors, "Evaluating the different protocols for low power wirelsess networks in industrial automation"
[7]https://www.google.co.kr/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwjRvquB4IjNAhVOtJQKHUyID0kQjRwIBw&url=%2Furl%3Fsa%3Di%26rct%3Dj%26q%3D%26esrc%3Ds%26source%3Dimages%26cd%3D%26cad%3Drja%26uact%3D8%26ved%3D0ahUKEwjRvquB4IjNAhVOtJQKHUyID0kQjRwIBw%26url%3Dhttp%253A%252F%252Fstakeholders.ofcom.org.uk%252Fmarket-data-research%252Fother%252Ftechnology-research%252Fresearch%252Femerging-tech%252Fmesh%252F%26psig%3DAFQjCNEyJVBNYXdXDwRM_nztXceeAOL16Q%26ust%3D1464936524785019&psig=AFQjCNEyJVBNYXdXDwRM_nztXceeAOL16Q&ust=1464936524785019
[8] http://www.contiki-os.org/start.html