RSSI 판독 모듈 구현

임재걸*, 정승환*
*동국대학교 컴퓨터멀티미디어학과
e-mail: *{yim, honourj}@dongguk.ac.kr

Implementation of a Module Scanning RSSI

Jaegeol Yim*, Seunghwan Jeong*
*Dept of Computer and Multimedia, Dongguk University

요 약

디지털 홈 네트워크가 활성화되면서 IEEE 802.11 기술 덕분에 가정은 물론 사무실과 학교, 병원, 역 등지에서도 무선 LAN을 이용한 인터넷 사용이 가능하다. 현재 가장 많이 사용되는 2.4GHz 대역의 802.11b, 802.11g 프로토콜에 대하여 RSSI를 판독할 수 있는 모듈을 C# 언어에서 구현한 사례와 이 모듈의 이용에 대하여 소개한다.

1. 서론

현재 디지털 홈 네트워크의 응용이 활발히 연구되고 실현됨에 따라 가정은 물론 사무실과 학교, 병원, 역 등지에서 무선 LAN을 이용한 인터넷 사용이 가능하게 되었다. 이것은 유선 LAN 형태인 이더넷의 단점을 보완하기 위해 고안된 IEEE 802.11 기술[1] 덕분으로, 이더넷 네트워크의 말단에 위치해 불필요한 배선작업 및 유지관리 비용을 최소화하는 효과를 가진다. 또한 Microsoft에서 개발한 RADAR[2] 시스템에서는 이 IEEE 802.11b 프로토콜의 RF 신호세기(RSSI: Received Signal Strength Indication)를 이용하여 위치인식 방식을 연구했다.

기존에 RSSI를 판독하려면 상용화 서비스로 개발된 프로그램을 이용하거나, C++ 언어로 프로그래밍된 제한적인 데모 프로그램을 이용해야함으로 판독된 RSSI 데이터를 바로 사용하기에는 불편함이 있었다. 본 논문에서는 흔히 무선 LAN, Wi-Fi(와이파이)라고 부르는 컴퓨터 무선 네트워크에 사용되는 IEEE 802.11 기술에 대하여 알아보고, 현재 많이 사용하는 802.11b, 802.11g 프로토콜에 대하여 C# 프로그래밍 언어에서 RSSI를 판독하는 방법과 그 구현 모듈을 소개한다. 여기서 C# 언어는 C++언어에 비해 프로그래밍 구현 시간이 빠르고 많은 라이브러리를 제공하는 반면, 일부 IEEE 802.11 기술 같은 라이브러리가 아직 제공되지 않았기 때문에 C# 언어에서 자유롭게 사용할 수 있도록 RSSI 판독 모듈을 구현하였다.

2. 기존의 연구

IEEE 802.11은 IEEE의 LAN/MAN 표준 위원회의 11번째 워킹 그룹에서 개발된 표준 기술을 의미한다. 보통 외부 WAN과 백본 스위치, 그리고 사무실 같은 공간에 설치된 AP 까지 이더넷 네트워크로 연결되면, AP 주변의 컴퓨터들은 무선으로 네트워크를 형성할 수 있다. 여기서 AP는 이더넷 허브와 비슷한 역할을 하며, 고유의 SSID(Service Set Identifier)와 BSSID (Basic SSID(MAC))를 가지고 있어 무선 클라이언트의 연결을 도와준다.

IEEE 802.11 네트워크 환경은 AP에 다수의 클라이언트가 연결되어 네트워크를 구성하는 Infrastructure 방식과 AP 없이 클라이언트끼리 네트워크를 구성하는 AD-HOC 방식이 있다. 그리고 이 무선 LAN의 표준 전송 방식에는 프로토콜의 초기버전에 이어 (b), (a), (g), (n) 순으로 등장하였다. 802.11 초기버전의 프로토콜은 2.4GHz 대역 전파에서 최고 전송 속도 2Mbps로 데이터를 송수신 하며, 이보다 기술을 더욱 발전시킨 802.11b는 최고 전송 속도가 11Mbps이다. 다음으로 등장한 802.11a 프로토콜은 5GHz 대역에서 최고 54Mbps 까지 전송 속도를 지원하며, 802.11g 프로토콜은

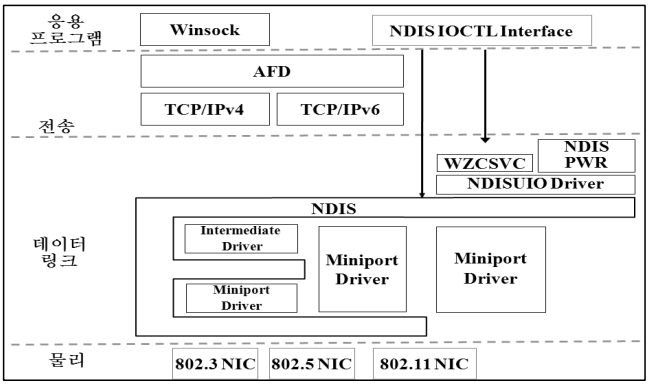
2.4GHz 대역에서 (a) 프로토콜과 동일한 전송 속도를 제공한다. 특히 (b) 프로토콜과 쉽게 호환되기 때문에 현재는 (b)와 더불어 (g) 프로토콜이 널리 쓰이고 있다. 마지막으로 등장한 802.11n 프로토콜은 현재 개발 중이며, 최고 540Mbps 까지 속도를 지원할 예정이다. RSSI의 값은 -30dBm 대에서 -90dBm 대까지 이뤄지며, AP로부터 통신 가능한 거리는 실내 45m, 실외 90m 정도로 제한된다.

3. 802.11 NIC과의 통신과정

802.11 NIC(Network Interface Card)는 네트워크 OSI(Open Systems Interconnection) 7계층 중 물리계층에 위치한 무선 LAN Card를 말한다. AP(Access Point)들이 SSID, BSSID, RSSI, Network Type등의 정보를 브로드캐스트하면 클라이언트는 802.11 NIC를 통해 정보들을 캐치하고 RSSI가 가장 강한 AP를 선택해 통신을 하게 된다.

3.1 NDIS Architecture

(그림 1)에는 OSI 7계층의 일부 계층을 보이고 있는데, 이 중 데이터링크 계층에 위치한 NDIS(Network Driver Interface Specification) 레이어는 Microsoft와 3Com이 공동으로 개발한 것으로 네트워크와 프로토콜의 겸용 드라이버이다. 물리 계층에 위치한 802.3 NIC, 802.5 NIC, 802.11 NIC 등과 응용 계층에 위치한 응용프로그램의 통신은 NDIS를 통해 이뤄진다. 응용 계층이 802.11 NIC로부터 정보를 가져온다는 것은 무선 LAN Card와 통신한다는 것을 의미하며, NDIS 드라이버를 Access 할 수 있는 NDIS IOCTL Interface를 구현함으로써 통신 문제를 해결할 수 있다. (그림 1)의 NDISUIO 드라이버는 Public Code로 작성되었고 응용프로그램이 NDIS 드라이버와 통신할 수 있도록 인터페이스를 제공하기 때문에 NDIS IOCTL Interface 구현에 큰 도움을 준다. 여기서 NDISPWR은 NDIS 드라이버의 전력 상태 구성을 제공한다.



(그림 1) NDIS Architecture

3.2 802.11 자동 구성

Windows XP는 자체 구성(Wireless Zero Configuration)을 이용하여 윈도우 XP에 내장된 기본 무선 유틸리티로 무선 LAN을 설정한다. (그림 1)에 보이는 Wireless Zero Config 서비스(WZCSVC)는 802.11 특정 OID와 드라이버 통신을 하며 Device 상태를 모니터하고 연결을 초기화 하는 서비스를 제공한다. 기본적으로 Windows XP가 WZCSVC를 이용하고 있기 때문에 이 WZCSVC를 사용하기 위해서는 먼저 Windows XP의 무선 LAN 설정 기능을 비활성화 해야 한다.

3.3 802.11 NIC과의 통신 과정

(그림 1)처럼 응용프로그램은 WZCSVC 서비스를 이용하여 NDISUIO 드라이버를 통해 802.11 NIC와 통신을 할 수 있다. <표 1>은 응용프로그램과 무선 LAN Card가 통신하는 과정을 정리한 것이다.

<표 1> 응용프로그램과 무선 LAN Card의 통신과정

① Windows XP의 WZCSVC를 비활성화 시킨다.

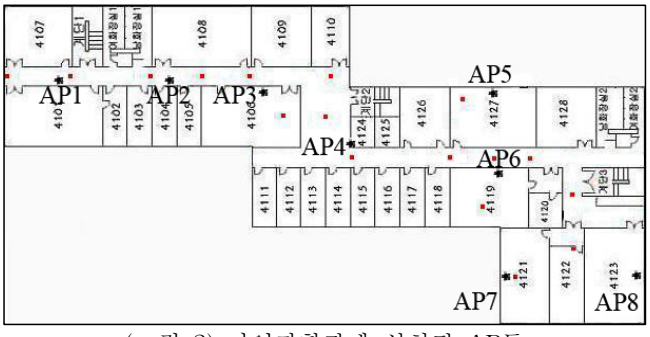
② kernel32.dll의 CreateFile() 함수로 NDISUIO 드라이버에 접근. CreteFile() 함수는 핸들을 반환한다.

③ kernel32.dll의 DeviceIoControl() 함수를 통해 Device들과 통신한다.

④ Device와 통신이 끝나면 CloseHandle() 함수로 Device와의 접근을 해제하고 Windows XP의 무선랜 설정 기능(WZCSVC)을 활성화 시킨다.

4. RSSI 판독 모듈 구현

RSSI를 판독하는 모듈을 구현하기 위하여 (그림 2)에 보이는 것처럼 자연과학관에 설치된 AP들 중 CISCO AIRONET 1131G Series Model인 AP4를 이용한다. 노트북은 삼성 SENS M40 Model이며, 무선 LAN Card는 Intel(R) PRO/Wireless 2200BG Network Connection이다. 프로그램 개발 도구로는 Microsoft Visual .NET 2005 통합 툴의 C# 언어로 Wireless-Manager Project를 생성하여 동적 라이브러리인 WirelessManager.dll 파일을 생성한다.



(그림 2) 자연과학관에 설치된 AP들

4.1 C#에서 802.11 NIC과 통신하기 위한 준비

kernel32.dll에 있는 CreateFile(), DeviceIoControl(), CloseHandle(), FormatMessage() 등의 DLL 익스포트 함수들을 C# 코드에서 호출하기 위해서는 해당 함수에 대응되는 static, extern 메소드를 선언하고 이 메서드에 DllImport 애트리뷰트를 지정해야 한다. 즉 <표 2>와 같이 PInvoke(플래폼 호출)를 정의해야 한다. 여기서 <표 2>는 여러개의 함수들 중 DeviceIoControl() 함수에 대하여 PInvoke가 정의된 것이다.

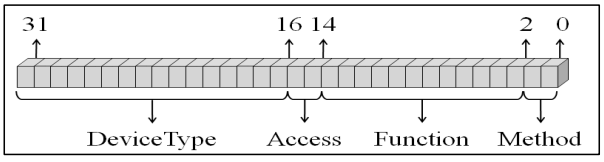
<표 2> DeviceIoControl() 함수의 PInvoke 정의

```
[DllImport("kernel32", SetLastError = true)]
private static extern bool DeviceIoControl(
    void* hDevice, IO_CTL_CODE dwIoCtlCode,
    void* lpInBuf, int nInBufSize,
    void* lpOutBuf, int nOutBufSize,
    out int lpBytesRet, void* lpOverlapped
);
```

CreateFile() 함수는 NDIS_FILE_NAME(@"\\.\Ndisuio")으로 NDISUIO 드라이버에 연결을 시도하고 연결이 성공되면 Handle을 반환한다. DeviceIoControl() 함수의 첫 번째 매개변수는 앞에서 반환된 Handle이며, 두 번째 매개변수인 dwIoCtlCode는 수행하고자 하는 IO Control Code를 말한다. Network Device와 통신할 경우 DeviceType(0x12), Access(0x1|0x2), Funtion, Method(0x0)의 속성 값에 따라 IO Control Code를 다음과 같이

((DeviceType) << 16) | ((Access) << 14) | ((Function) << 2) | (Method)

DDK(Winioctl.h, nuiouser.h)에 정의된 규칙으로 생성하며, (그림 3)은 생성된 값이 차지하는 비트의 자리를 보이고 있다.



(그림 3) IO Control Code 값의 비트의 자리

여기서 Funtion 속성의 값에 따라 IO Control Code의 수행하는 역할이 달라지는데, Funtion 속성 값에는 여러 가지가 있기 때문에 본 논문에서는 <표 3>에 보이는 5개 요소를 사용한다.

<표 3> IoCtlCode에 따른 버퍼의 사용 여부

IoCtlCode	InBuffer	OutBuffer
Bind Wait (0x204)	사용 안함	사용 안함
Query Binding (0x203)	사용	사용
Open Device (0x200)	사용	사용 안함
Set Object ID (0x205)	사용	사용 안함
Query Object ID (0x201)	사용	사용

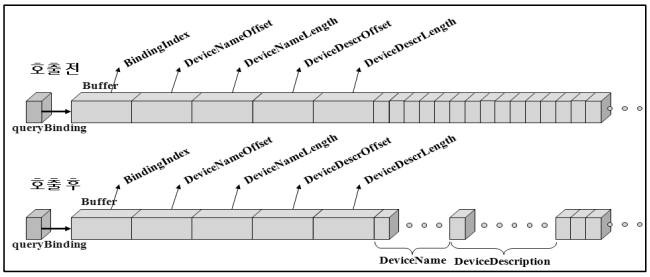
만약 통신 작업이 끝나면 CloseHandle() 함수를 통해 연결을 끝내며, FormatMessage() 함수는 통신 작업 중 발생될 수 있는 Error에 대하여 원인을 메시지로 보여준다.

4.2 DeviceIoControl() 함수

두 번째 매개변수인 IO Control Code에 따라서 DeviceIoControl() 함수의 사용 용도가 달라진다. 또한 이 용도에 따라 작업에 요구되는 데이터와 작업 후 출력되는 데이터의 존재 여부가 <표 3>과 같이 각각 다르다.

Bind Wait의 목적은 CreateFile()함수를 통해 생성된 핸들을 가지고 Network Device로의 접근을 시도하는 것이며, 이때 사용하는 버퍼는 없다.

Query Binding의 목적은 Bind Wait가 성공할 경우 모든 Network Device의 DeviceName과 DeviceDescription을 가져오는 것이다. (그림 4)의 호출 전 Buffer를 보면 이름이 명시된 공간이 입력 버퍼(4byte씩)로 사용되며, 그 뒷부분의 DeviceName과 DeviceDescription이 출력 버퍼로 사용되어 하나의 Device에 대해 한 쌍의 정보를 가진다. 두 번째 매개변수를 Query Binding으로 함수를 호출 하고 나면 Buffer에는 각 Device에 대하여 한 쌍씩 출력 결과가 연속적으로 저장된다. 입력 버퍼에서 BindingIndex는 Binding 순서로써 호출 전에 설정되고, 이를 제외한 뒤의 4개의 공간은 함수 호출 후에 출력 버퍼의 각 요소에 대하여 Offset과 Length로 갱신된다.



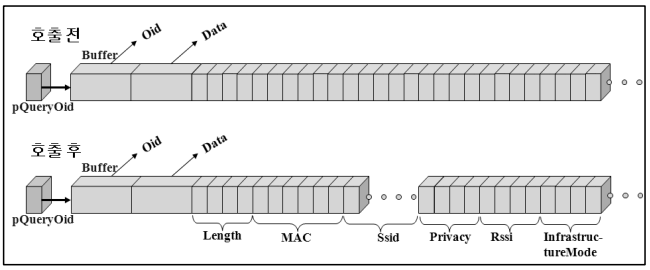
(그림 4) Query Binding에서의 Buffer 상태

Open Device의 목적은 Query Binding으로 수집한 Device Name들 중 Wireless LAN Card에 해당하는 Device Name을 선택하고, 이것을 Open Device의 Device Name으로 설정하여 열기 작업을 수행하는 것이며, 출력 버퍼는 사용하지 않는다.

Set Object ID의 목적은 Wireless LAN이 Bssid List를 수집하도록 지시하는 것으로 입력 버퍼에는Bssid List를 수집하라는 지시 내용이 담겨있으며 출력버퍼는 사용하지 않는다. 여기서 Bssid는 주변 AP 단말기의 브로드캐스팅 정보로 Mac Address, SSID, RSSI, Infrastructure Mode등을 말한다. 그리고 이 함수의 작업이 수행되면 대략 6초가 경과해야 Wireless LAN Card가 주변의 Bssid List를 충분히 수집할 수 있다.

Query Object ID의 목적은 Wireless LAN Card로부터 Bssid List들을 가져오는 것이다. 그렇기 때문에 작업에 요구되는 데이터는 지시 내용이 담긴 Code이며, 작업 후 출력되는 데이터는 Bssid List들이다.

Query Object ID를 매개변수로 DeviceIoControl()함수가 호출되면 (그림 5)의 호출 후 Buffer와 같이 Length(Bssid의 길이), Bssid가 한 쌍이 되며, 이 쌍이 Bssid의 수만큼 연속적으로 Buffer에 출력되고 입력 버퍼의 Data에 Bssid의 개수가 갱신된다.



(그림 5) Query Object ID에서의 버퍼 상태

4.3 WirelessManager Project

(그림 6)의 WirelessManager는 NDIS IOCTL Interface인 WirelessManager.dll을 생성하는 Project이다. WirelessManager Project는 C# 언어로써 포인터 연산자(*), 주소 연산자(&), 역 참조 멤버 연산자(->)를 사용하기 때문에 Project 속성에서 unsafe code 요소를 활성화 시켜야한다.

<표 4> WirelessManager Project의 각 파일

파일명	내용
ManagerDefinition.cs	◎ IO Control 타입 정의 ◎ 802.11 타입 정의
AccessPoint.cs	◎ AccessPoint 타입 정의
MacAddress.cs	◎ AccessPoint의 Mac-Address 구조체 정의
NdisDevice.cs	◎ Device 타입 정의
WirelessManager.cs	◎ 무선 LAN과의 통신 및 제어

<표 4>는 (그림 6)에 보이는 각 파일들이 하는 일을 정리한 것으로 크게 타입을 정의한 파일들과 무선 LAN과의 통신 및 제어를 구현한 파일이다. 이 Project의 네임스페이스는 “Dongguk3CSLab.Wireless”이다.

5. 구현 결과

WirelessManager Project가 생성한 Wireless-Manager.dll은 RSSI를 판독할 수 있는 NDIS IOCTL Interface역할을 한다.

<표 5> WirelessManager.dll 사용을 위한 초기화

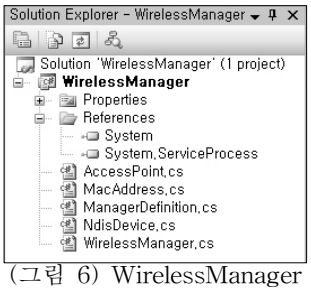
```
NdisDevice dev;

// 기존에 WirelessManager가 사용되고 있었다면,
// 기존 사용을 종료하고 다시 초기화 한다.
if(WirelessManager.CurrentDevice != null)
    WirelessManager.Finish();
WirelessManager.Start();

// 무선 LAN Card의 장치 정보를 수집하고
// WirelessManager에 현재 장치로 갱신한다.
WirelessManager.OpenWirelessDevice();
dev = WirelessManager.CurrentDevice;
```

5.1 WirelessManager.dll을 사용하여 RSSI 판독

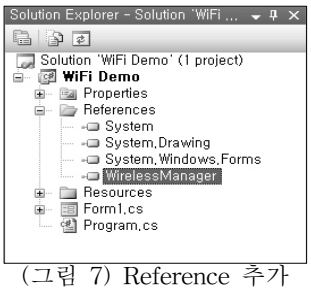
NDISUIO 드라이버를 사용하려는 C# Project에서 (그림 7)처럼 WirelessManager.dll을 References에 추가하고, Dongguk3CSLab.Wireless를 using하면 C# 코드 어디에서나 이 모듈을 사용할 수 있다. <표 5>와 같이 초기화 하고, 다음과 같이 ScanAPList() 함수를 호출하면

```
AccessPoint[] aplist;
aplist = WirelessManager.ScanAPList(dev);
```

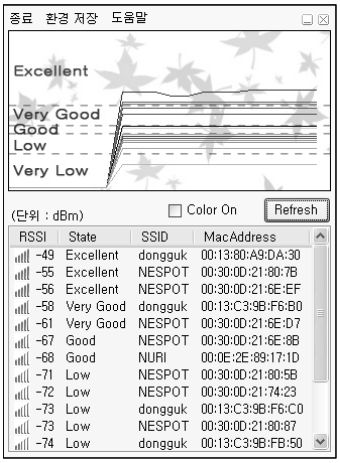
주변 AP들이 브로드캐스트하는 정보를 수신하여 RSSI를 판독할 수 있다. (그림 8)은 WirelessManager를 이용하여 임의의 지점에서 RSSI 판독 프로그램을 수행한 실행화면으로 1초 단위로 RSSI 값을 갱신한다.



(그림 6) WirelessManager Project



(그림 7) Reference 추가



(그림 8) RSSI 판독 프로그램의 실행화면

6. 결론

본 논문에서는 IEEE 802.11 프로토콜 중 2.4GHz 대역을 사용하는 802.11b와 802.11g를 대상으로 OSI 7계층의 응용프로그램 계층에서 물리계층간의 통신 절차를 알아보고, 이를 C#언어에서 WirelessManager Project를 통해 RSSI 판독 모듈을 구현하였다. 생성된 WirelessManager.dll만 있으면 C# 언어로 구성된 프로그램 어디에서나 AP의 정보를 Access할 수 있기 때문에 AP의 정보를 요구하는 관련 연구에서 많은 도움이 될 것으로 예상된다.

향후에는 구현된 RSSI 판독 모듈에 부가 기술을 추가함으로써 지금보도 더 많은 기능을 가진 다기능 모듈을 구현하고자 한다.

참고문헌

[1] http://en.wikipedia.org/wiki/IEEE_802.11

[2] Bahl, P. and Padmanabhan, V., "RADAR: An in-building RF-based user location and tracking system", INFOCOM 2000, Mar. 2000, pp. 775-784