VAASAN AMMATTIKORKEAKOULU
VASA YRKESHÖGSKOLA
UNIVERSITY OF APPLIED SCIENCES

Xufei Ning

XBEE API MODE PROGRAMMING AND A SOUND DETECTION APPLICATION

Technology and Communication

2015

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Information Technology

ABSTRACT

Author	Xufei Ning
Title	Xbee API Mode Programming and a Sound Detection Application
Year	2015
Language	English
Pages	47
Name of Supervisor	Chao Gao

This project is about using Xbee API mode to transmit sound detection data in a wireless sensor network. In such a sound detection wireless sensor network, we use Raspberry Pi as a sink node, and a group of Arduino Mega 2560 as sensor nodes. The wireless communication was achieved by Xbee RF modules working in API mode.

This system has a basic function run as a sound level meter. The sensor node can measure sound level in RMS (Root Mean Square) value and turn on a LED if the RMS value is over a threshold. Then the sensor nodes will send these RMS values to a sink node. The sink node display RMS value with its sensor ID, frame ID and timestamp. All these data will be recorded to log files for further use.

Keywords Sound detection, Arduino, Raspberry Pi, Xbee API mode

CONTENTS

LIST OF ABBREVIATIONS

ADC	Analog-Digital Converter
API	Application Programming Interface
AT	Transparent Mode
DI	Digital Input
DO	Digital Output
FTDI	Future Technology Devices International
GND	Ground
GPIO	General Purpose Input/Output
I/O	Input/Output
LED	Light Emitting Diode
PAN	Personal Area Network
PWM	Pulse Width Modulation
RF	Radio Frequency
RMS	Root Mean Square
RSSI	Received Signal Strength Indication
SSH	Secure Shell
UART	Universal asynchronous receiver/transmitter
USB	Universal Serial Bus

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

This chapter includes the purpose of this project and the structure of this thesis.

1.1 Purpose of Project

Sound is an important part of our lives, it allows human beings and animals to hear and communicate, and it provides us information about the world around us.

The use of a sound detection system offers a monitor of specific environment, so it may also provide a solution of surveillance and security. Furthermore, sound detection system can be used to analyse noise sources or be used in portable devices to inform person.

This project contains two parts: sound detection system and using Xbee API mode to transmit data.

A sound sensor node will detect the sound level in RMS value every 125ms, the same as a sound level meter works at Fast mode /1/. The sensor node will turn on a LED for three seconds if the RMS value is greater than a threshold. During the time LED is turned on, any detection over the threshold will extend LED for another three seconds.

A packet will send from the sensor node to the sink node every 1 second, which contains eight pieces of RMS value and a frame ID.

The sink node will unpack the packet, analysis of the source address, frame ID, RMS values, and display these data with a timestamp. Then the sink node records the original packet into a .csv log file, and records the source address, frame ID, timestamp and RMS values into another .csv file for further use.

1.2 Structure of Thesis

Chapter 1 introduces the purpose and function of this project. Then Chapter 2 illustrates the background of theoretical support, including methodologies, system structure, hardware and software. After that Chapter 3 lists all the implementation steps of sound detection system and Xbee module API mode data transmitting. Finally, Chapter 4 gives the conclusion.

2 BACKGROUND

This chapter illustrates the background of theoretical support, including methodologies, system structure, hardware and software.

2.1 System Structure

The system is divided into two parts according to its hardware design: Sensor Node and Sink Node. One Sink may support multiple sensor nodes. Each Sensor Node has its own sensor ID. The hardware is described in Table 1.

Table 1. Description of hardware.

Sensor Node:	**Sink Node:**
Microphone	Raspberry Pi
Amplifier	XBee module
Arduino	Adapter
XBee module	
LED	

The system stats with analog sound signal input, ends with display data and record data to the log file. The whole system structure is shown in Figure 1.

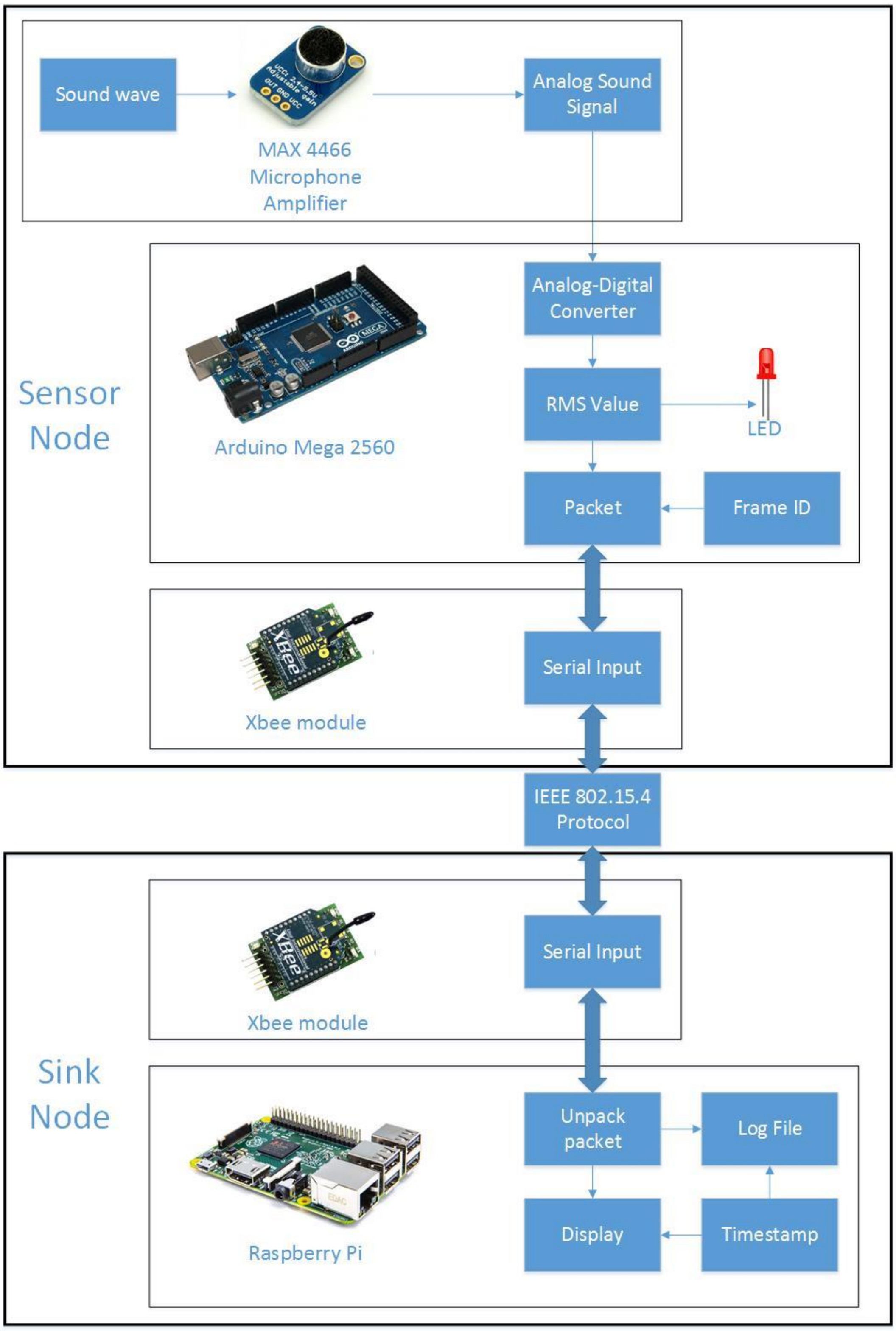


Figure 1. System Structure

2.2 Theory

2.2.1 Analog to Digital Conversion

Analog-to-digital conversion is an electronic process in which a continuously variable (analog) signal is changed, without altering its essential content, into a multi-level (digital) signal. The device that converts a continuous physical quantity to a digital number is called analog-to-digital converter.

An important factor influencing the output of an ADC is resolution. The resolution of the converter is the digital value range of the analog signal conversion. The formula of ADC converter is:

$$\frac{Resolution\ of\ ADC}{System\ Voltage} = \frac{ADC\ reading}{Analog\ Voltage\ Measured} \tag{1}$$

For example, a 10 bit (0-1023) ADC works on a 5V system, if the analog voltage given is 3V, the ADC conversion result x will be:

$$\frac{1023}{5V} = \frac{x}{3V}$$

$$\mathrm{x} = 614$$

In this project, ADC is used to convert analog sound signals to digital sound signals. Figure 2 shows the parameter of the sound signal.

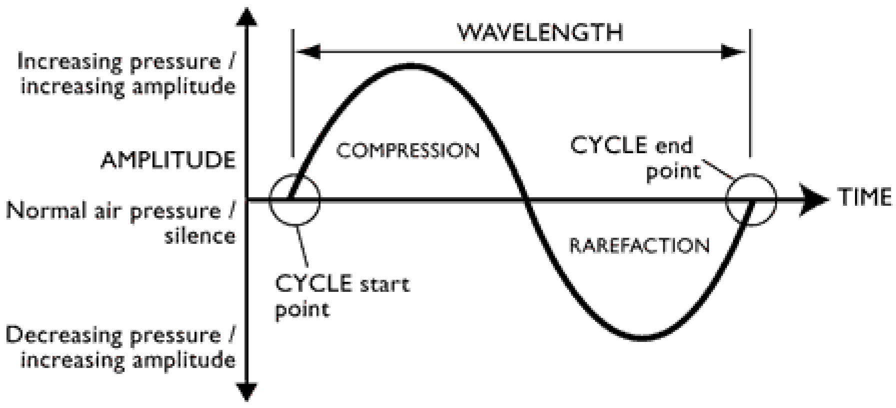

WAVELENGTH
Increasing pressure / increasing amplitude
AMPLITUDE
Normal air pressure / silence
Decreasing pressure / increasing amplitude
COMPRESSION
RAREFACTION
CYCLE end point
CYCLE start point
TIME

Figure 2. Sound wave

A sound wave includes increasing pressure (+) and decreasing pressure (-). A 10 bit ADC is used in this project to convert the sound signal, the range of ADC output is 0 to 1023. In order to represent the sound signal, ADC have to subtract the raw value by 512, the range is changed to -512 to +511.

2.2.2 Root Mean Square

The root mean square (RMS), also known as the quadratic mean in statistics, is a statistical measure defined as the square root of the mean of the squares of samples. /2/

The formula of RMS calculation is:

$$x_{rms} = \sqrt{\frac{1}{n}\left({x_1}^2 + {x_2}^2 + \cdots + {x_n}^2\right)} \tag{2}$$

Where:

$$\mathrm{x} = \mathrm{sample} - 512$$

$$\mathrm{n} = 1096$$

The value of n is calculated according to the sample rate of Arduino Mega 2560. Arduino Mega 2560 has a sample rate of 8776Hz, the RMS value is calculated 8 times per second (125ms), which means each RMS calculation contains $8776/8 = 1096$ samples.

2.2.3 Protocol, IEEE 802.15.4

IEEE 802.15.4 produces a standard that enables very low-cost, low-power communications. A system conforming to this standard consists of several components. The most basic is the device. Two or more devices communicating on the same physical channel constitute a WPAN. /3/

Several XBee modules that meet IEEE 802.15.4 standard are used in this project, one as a receiver used by the sink node, rest as transmitters used by the sensor node, The Xbee network topology is shown in Figure 3.

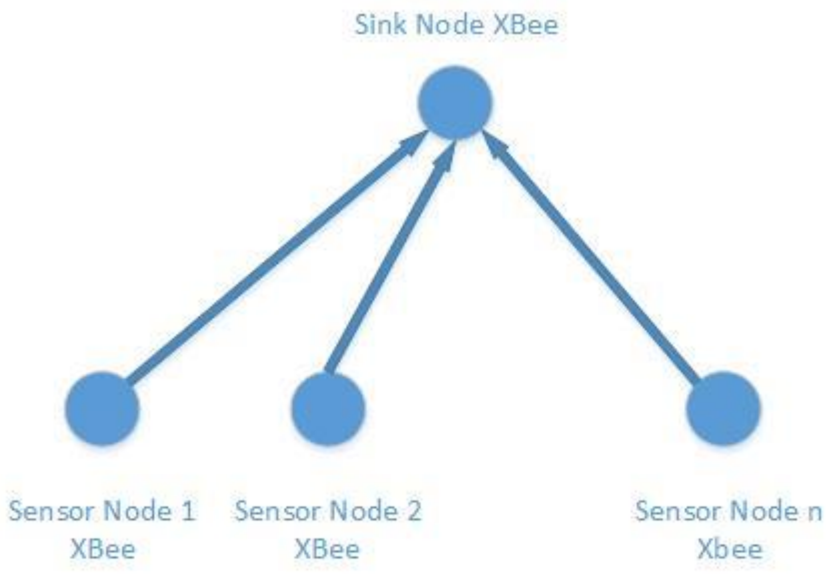


Figure 3. Xbee network topology

Devices following IEE 802.15.4 have several parameters for configuration:

PAN ID: With multiple devices using IEEE 802.15.4 standard in the same area, the PAN ID is used to distinguish which devices are in the same PAN domain.

16-bit address: Each device has a unique 64-bit identifier, and some devices may use short 16-bit identifiers within a restricted environment (eg. the same PAN domain).

2.3 Hardware

2.3.1 MAX4466 Microphone Amplifier

MAX4466 is micro power op amps optimized for use as microphone preamplifiers. The gain can set from 25X to 125X. Figure 4 shows the front view and back view of MAX4466.

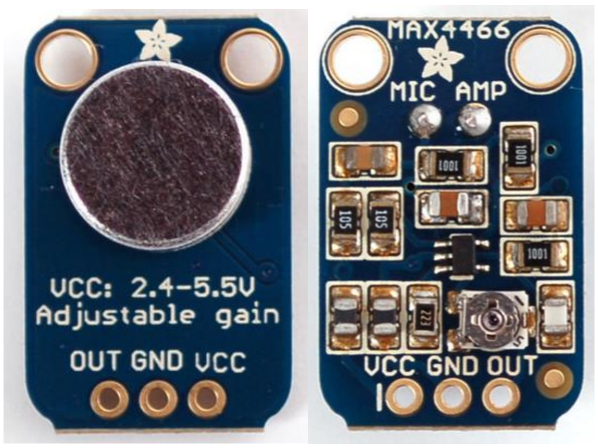


Figure 4. MAX4466

2.3.2 Arduino Mega 2560

Arduino is an open-source physical computing platform based on a simple microcontroller board, and a development environment for writing software for the board. /4/

The Arduino board used in this project is the Arduino Mega 2560. Figure 5 shows the Arduino Mega 2560, the parameters of Arduino Mega 2560 are shown in Table 2. /5/

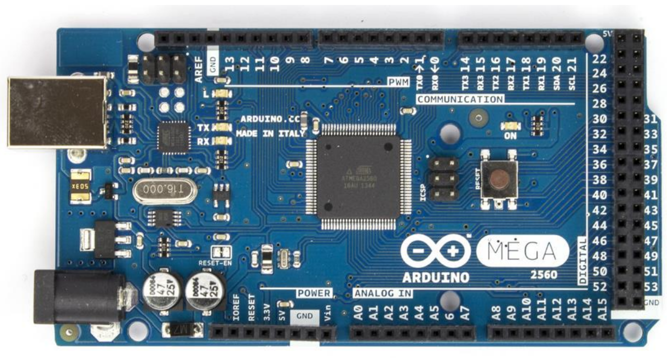


Figure 5. Arduino Mega 2560

Table 2. Parameters of Arduino Mega 2560

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16 (of which provide 10 bits of resolution)
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz

2.3.3 Raspberry Pi

The Raspberry Pi is a series of credit card-sized single-board computers developed in the UK by the Raspberry Pi Foundation with the intention of promoting the teaching of basic computer science in schools. /6/

The Raspberry Pi board used in this project is Raspberry Pi 1 Model B+. Figure 6 shows the Raspberry Pi 1 Model B+, the parameters of Raspberry Pi 1 Model B+ are shown in the Table 3. /7/

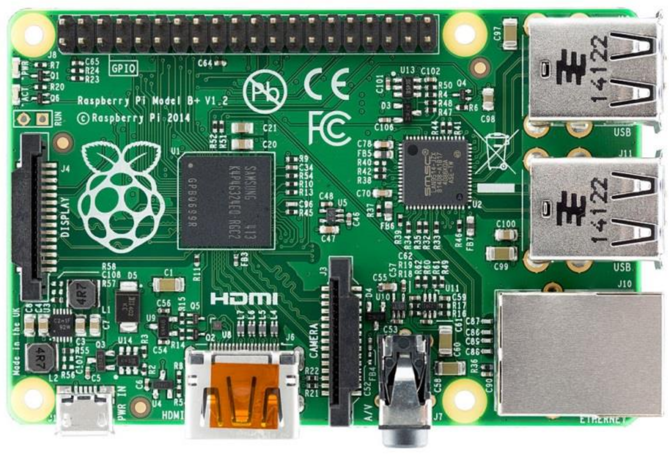


Figure 6. Raspberry Pi 1 Model B+

Table 3. Parameters of Raspberry Pi 1 Model B+

Chip	Broadcom BCM2835 SoC
Core architecture	ARM11
CPU	700 MHz Low Power ARM1176JZFS Applications Processor
GPU	Dual Core VideoCore IV® Multimedia Co-Processor Provides Open GL ES 2.0, hardware-accelerated OpenVG, and 1080p30 H.264 high-profile decode Capable of 1Gpixel/s, 1.5Gtexel/s or 24GFLOPs with texture filtering and DMA infrastructure
Memory	512MB SDRAM
Operating System	Boots from Micro SD card, running a version of the Linux operating system
Dimensions	85 x 56 x 17mm
Power	Micro USB socket 5V, 2A
Ethernet	10/100 BaseT Ethernet socket
Video Output	HDMI (rev 1.3 & 1.4) Composite RCA (PAL and NTSC)
Audio Output	3.5mm jack, HDMI
USB	4 x USB 2.0 Connector
GPIO Connector	40-pin 2.54 mm (100 mil) expansion header: 2x20 strip Providing 27 GPIO pins as well as +3.3 V, +5 V and GND supply lines
Camera Connector	15-pin MIPI Camera Serial Interface (CSI-2)
JTAG	Not populated
Display Connector	Display Serial Interface (DSI) 15 way flat flex cable connector with two data lanes and a clock lane
Memory Card Slot	SDIO

2.3.4 Xbee RF Modules

The Xbee and Xbee-PRO RF Modules were engineered to meet IEEE 802.15.4 standards and support the unique needs of low-cost, low-power wireless sensor networks. The modules require minimal power and provide reliable delivery of data between devices. /8/

The Xbee RF modules are shown in Figure 7, the datasheet of Xbee RF module is shown in Table 4. /9/

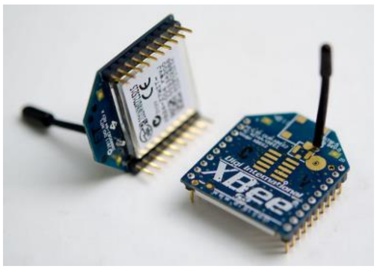


Figure 7. Xbee module

Table 4. Datasheet of Xbee module

Indoor/Urban Range	Up to 100 ft (30 m)
Outdoor RF line-of-sight Range	Up to 300 ft (90 m)
Transmit Power Output (software selectable)	1mW (0 dBm)
RF Data Rate	250,000 bps
Serial Interface Data Rate (software selectable)	1200 bps - 250 kbps (non-standard baud rates also supported)
Receiver Sensitivity	-92 dBm (1% packet error rate)
Supported Network Topologies	Point-to-point, Point-to-multipoint & Peer-to-peer
Number of Channels (software selectable)	16 Direct Sequence Channels
Addressing Options	PAN ID, Channel and Addresses

Xbee and Xbee-PRO RF modules operate in two different modes: AT mode (Transparent Mode) and API mode (Application Programming Interface)

2.3.4.1 AT Mode

When operating in the AT mode, the Xbee RF module act as the serial port replacement. All received RF data is sent to the DO pin, as well as the DI pin is receiving data for RF transmission.

2.3.4.2 API Mode

The API mode is a frame-based method for sending and receiving data to and from a radio's serial UART. When in the API mode, all data entering and leaving the module is formatted as frames that define operations or events within the module. The API allows the programmer to: /10/

- Change parameters without entering command mode (Xbee only)
- View RSSI and source address on a packet by packet basis
- Receive packet delivery confirmation on every transmitted packet

By setting the AP parameter values, the Xbee module may operate in the following modes:

- AP = 0 (default): API modes are disabled.
- AP = 1: API Operation
- AP = 2: API Operation (with escaped characters)

When the Xbee module works at the API mode (AP = 1), its data frame structure is defined as shown in Figure 8:

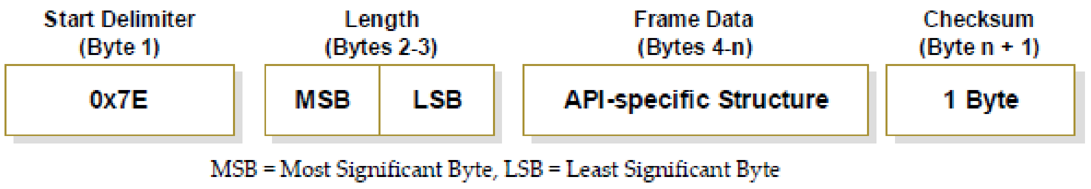


Figure 8. Data Frame Structure

When the Xbee module works in the API mode (AP = 2), its data frame structure is defined as shown in Figure 9:

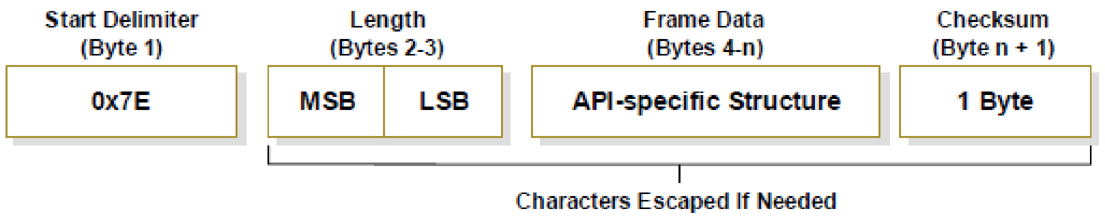


Figure 9. Data Frame Structure (with escape control characters)

Escape characters /11/: When sending or receiving a UART data frame, specific data values must be escaped (flagged) so they do not interfere with the UART or UART data frame operation. To escape an interfering data byte, insert 0x7D is inserted and followed with the byte to be escaped XOR'd with 0x20.

Escape characters needed by the data are listed in Table 5.

Table 5. Escape Characters

0x7E	Frame Delimiter
0x7D	Escape
0x11	XON
0x13	XOFF

The API mode used in this project is API mode 1. So that all packets from sensor node are in the same format and same length, the receiver will unpack packets easier.

The API packet can be defined to several specific structures to support different kinds of data frame. The cmdID frame (API-identifier) defines the API types, shown in Figure 10.

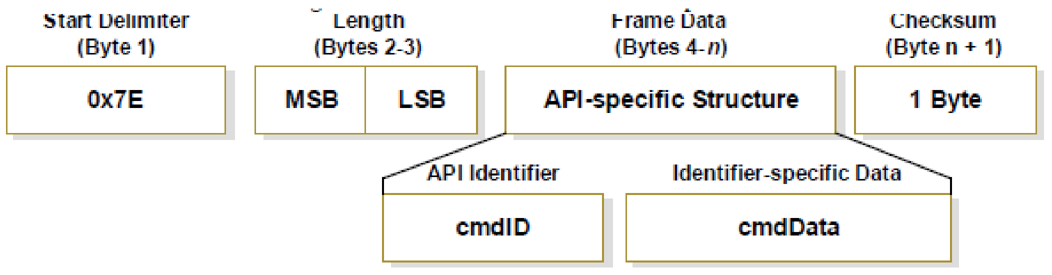


Figure 10. API identifier

The API type used in this project is RX (Receive) Packet: 16-bit Address, which has API identifier value 0x81, its structure is shown in Figure 11. The parameters of the packet are listed in Table 6.

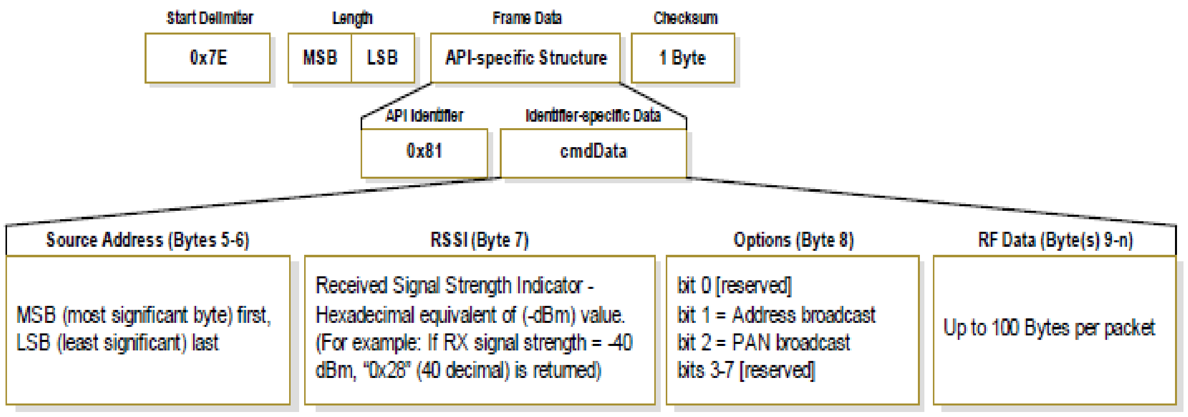

Start Delimiter
0x7E
Length
MSB
LSB
Frame Data
API-specific Structure
Checksum
1 Byte
API Identifier
0x81
Identifier-specific Data
cmdData
Source Address (Bytes 5-6)
MSB (most significant byte) first, LSB (least significant) last
RSSI (Byte 7)
Received Signal Strength Indicator - Hexadecimal equivalent of (-dBm) value. (For example: If RX signal strength = -40 dBm, "0x28" (40 decimal) is returned)
Options (Byte 8)
bit 0 [reserved]
bit 1 = Address broadcast
bit 2 = PAN broadcast
bits 3-7 [reserved]
RF Data (Byte(s) 9-n)
Up to 100 Bytes per packet

Figure 11. API type: RX (Receive) Packet: 16-bit Address

Table 6. API type: RX (Receive) Packet: 16-bit Address

Start Delimiter	7E (data before start delimiter 7e is discarded)
Length	length of frame data
API identifier	0x81 (RX (Receive) Packet: 16-bit Address)
Source Address	16bit source address
RSSI	Received Signal Strength Indicator
Options	0
RF data	sent data up to 100 bytes
Checksum	packet is discarded if the checksum unqualified

2.3.5 Xbee Adapter

The Xbee RF module needs adapters to connect with other devices. There were three different kinds of adapters used in this project:

The Raspberry Pi Sink Node uses Xbee to the USB adapter because it is easier than using GPIO pins. Arduino Sensor Nodes use Xbee to the FTDI cable adapter or Xbee to the Arduino shield for the RS232 serial communication.

Figures 12 – 14 show the three kinds of Xbee adapters:

Figure 12. Xbee to USB adapter

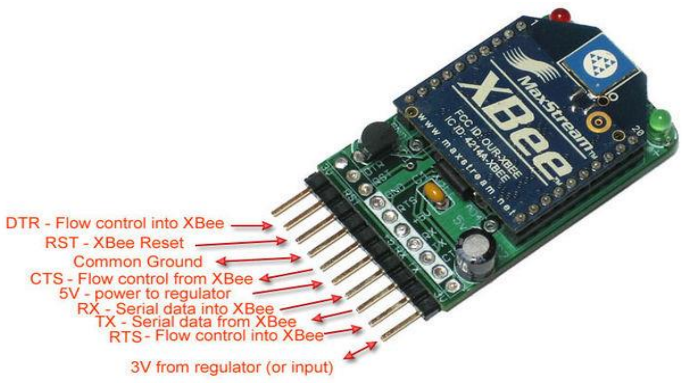

DTR - Flow control into XBee
RST - XBee Reset
Common Ground
CTS - Flow control from XBee
5V - power to regulator
RX - Serial data into XBee
TX - Serial data from XBee
RTS - Flow control into XBee
3V from regulator (or input)

Figure 13. Xbee to FTDI cable adapter

Figure 14. Xbee to Arduino shield

2.3.6 Design of Hardware

This system can support multiple Sensor Nodes. In this project, two Sensor Nodes were used. Figure 15 and Figure 16 show the Sensor Node and the Sink Node.

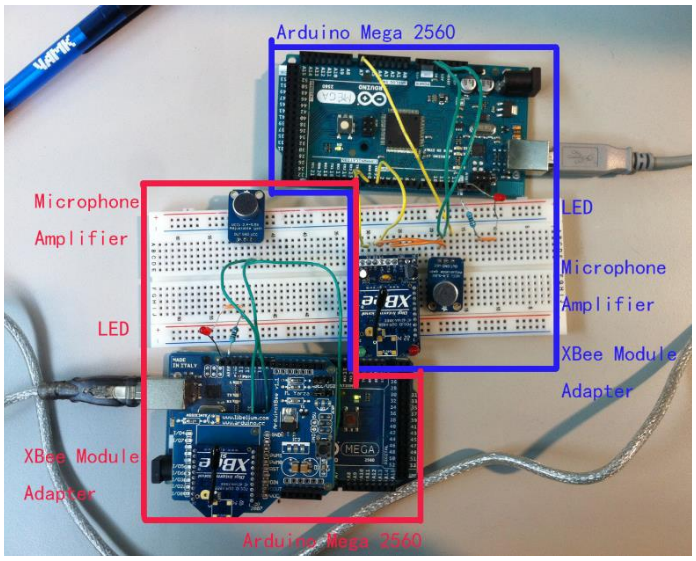


Figure 15. Sensor Node

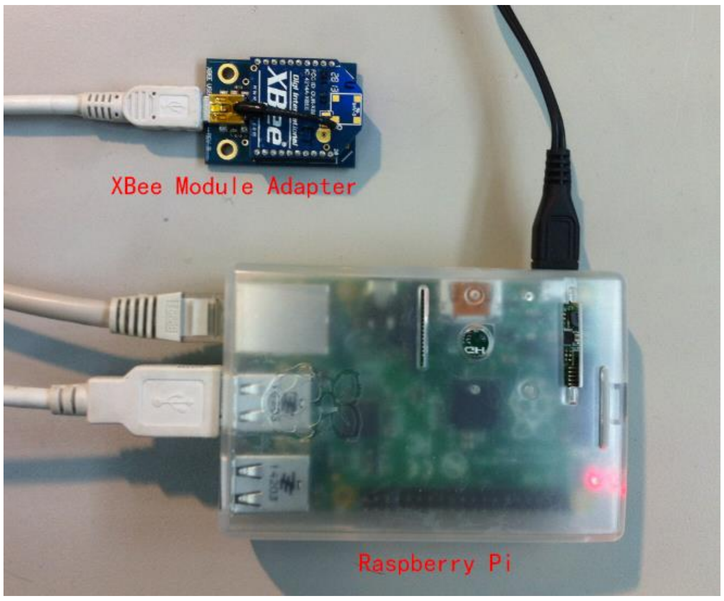


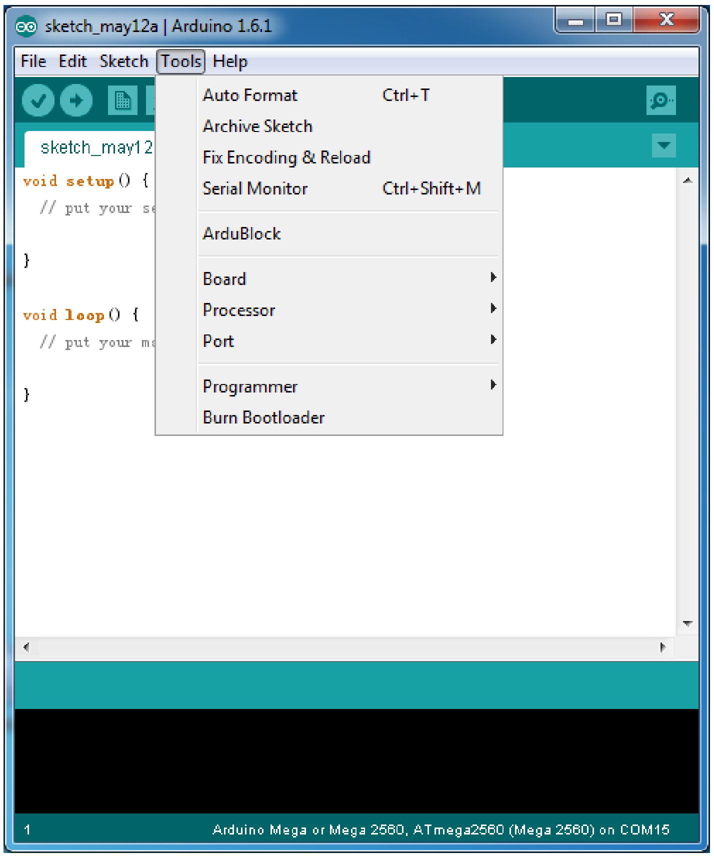
Figure 16. Sink Node

2.4 Software

2.4.1 Arduino Software

Arduino software is an open-source integrated development environment (IDE) for writing code and uploading to Arduino board. The programming language for Arduino board is C and C++.

Figure 17 shows the window of Arduino Software, in the toolbar, board, processor and port can be selected.



```
void setup() {
  // put your se

}

void loop() {
  // put your ma

}
```

```
1          Arduino Mega or Mega 2560, ATmega2560 (Mega 2560) on COM15
```

Figure 17. Arduino Software

2.4.2 X-CTU

X-CTU is a free multi-platform application designed to enable developers to interact with XBee RF Modules through a simple-to-use graphical interface.

In this project, X-CTU was used to configure the Xbee modules, the parameter includes: PAN ID, 16bit source address, baud rate, API mode. Three Xbee modules were used here, two as sensor node, and one as sink node. Their parameters are listed in Table 7.

Table 7. Xbee configuration

	Sensor Node 1	Sensor Node 2	Sink Node
PAN ID	1234	1234	1234
16bit address	2222	3333	1111
Baud rate	57600	57600	57600
API enable	1	1	1

3 IMPLEMENTATION

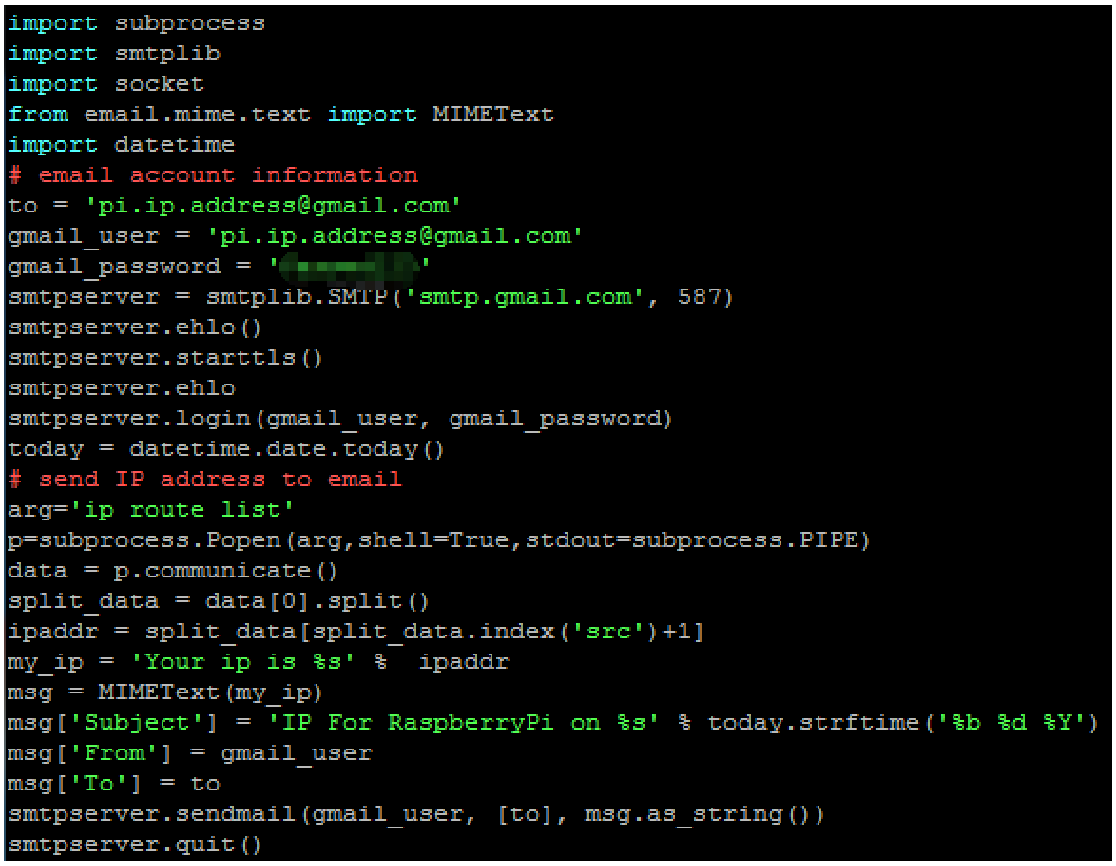
This chapter lists all implementation step of this project.

3.1 Raspberry Pi Configuration

The power supply was plugged in and Raspberry Pi was connected to the router as the control computer. PuTTY was run, the IP address of Raspberry Pi was filled and then logged in.

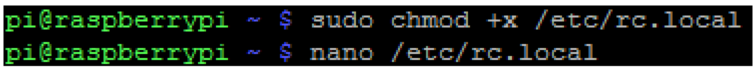
Here an easy method was provided to get the IP address of Raspberry Pi: Raspberry Pi sent its IP address to a specific email once it booted.

A necessary program was used to send IP address of Raspberry Pi to a specific email:



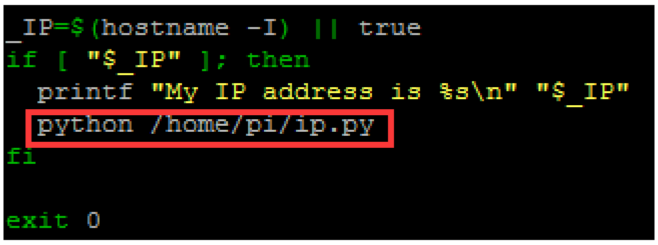
```
import subprocess
import smtplib
import socket
from email.mime.text import MIMEText
import datetime
# email account information
to = 'pi.ip.address@gmail.com'
gmail_user = 'pi.ip.address@gmail.com'
gmail_password = '[redacted]'
smtpserver = smtplib.SMTP('smtp.gmail.com', 587)
smtpserver.ehlo()
smtpserver.starttls()
smtpserver.ehlo
smtpserver.login(gmail_user, gmail_password)
today = datetime.date.today()
# send IP address to email
arg='ip route list'
p=subprocess.Popen(arg,shell=True,stdout=subprocess.PIPE)
data = p.communicate()
split_data = data[0].split()
ipaddr = split_data[split_data.index('src')+1]
my_ip = 'Your ip is %s' %  ipaddr
msg = MIMEText(my_ip)
msg['Subject'] = 'IP For RaspberryPi on %s' % today.strftime('%b %d %Y')
msg['From'] = gmail_user
msg['To'] = to
smtpserver.sendmail(gmail_user, [to], msg.as_string())
smtpserver.quit()
```

This file was named as ip.py. The rc.local file was edited to set ip.py run automatically when Raspberry Pi was connecting to the network, the command line to edit re.local was:



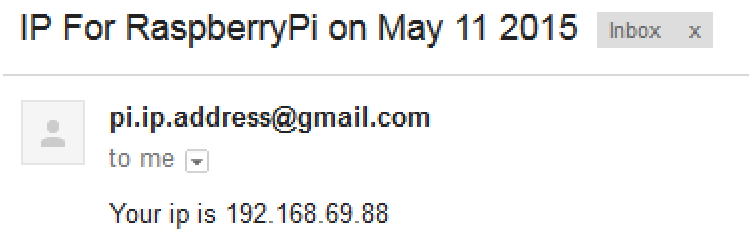
```
pi@raspberrypi ~ $ sudo chmod +x /etc/rc.local
pi@raspberrypi ~ $ nano /etc/rc.local
```

The line python /home/pi/ip.py was added.



```
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
  printf "My IP address is %s\n" "$_IP"
  python /home/pi/ip.py
fi

exit 0
```

Once the Raspberry Pi booted and connected to the network, it sent its IP address to the specific email, shown in Figure 18.



IP For RaspberryPi on May 11 2015 Inbox x

pi.ip.address@gmail.com
to me

Your ip is 192.168.69.88

Figure 18. Raspberry Pi IP address

3.2 Measure Arduino Sample Rate

An easy way to measure Arduino sample is to use oscilloscope. Arduino was allowed to sample 1000 times, then an I/O pin was flipped, using the oscilloscope to measure the duty cycle of the I/O pin.

In real situation, the sample rate depends on how fast ADC makes one conversion and how many other codes are in the loop. Any extra code such as floating point calculation and serial output will affect the sample rate.

In this project, the sample rate of Arduino Mega 2560 was measured as 8776.

3.3 Converting Analog Sound Signal to Digital Values

The first part of this project was to measure the sound level in RMS values every 125ms. Arduino Mega 2560 has sample rate of 8776 and 10bits of analog input resolution. /12/

This means that the analog value can be presented in digital value from 0 to 1023, and each RMS values is calculated according to 1096 samples.

An example is shown in Figure 19. A 125ms analog sound wave can be converted to digital value with 1096 samples.

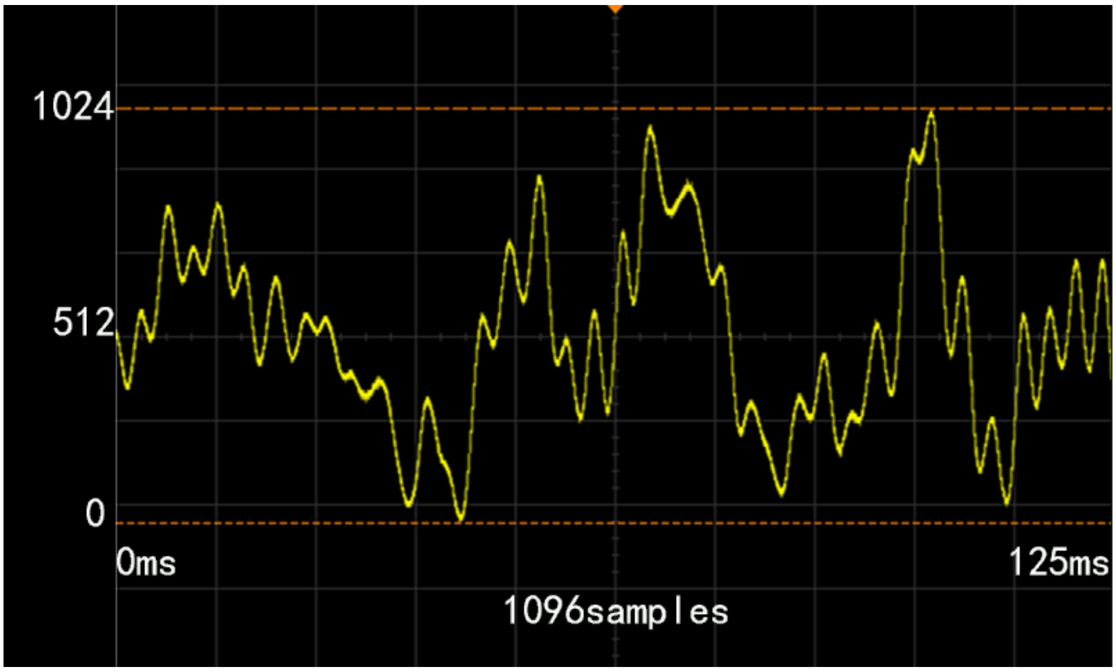


Figure 19. Analog signal to digital signal

3.4 Calculating RMS Value

In this project, a RMS value presents the change value of the sound wave with 125ms by formula (2).

3.5 Sound Control on LED

The sensor node will turn on a LED for three second if the RMS value is greater than a threshold. During the time LED is turned on, any detection over the threshold will extend LED for another three seconds.

The LED control pin used here is PIN 13. The timer function used is millis(): Returns the number of milliseconds since the Arduino board began running the current program.

When connecting a LED to PIN 13 and GND on the board, it is necessary to use a resistor, otherwise the board could have damaged.

3.6 Frame ID

When the Sensor Node sends RMS values to the Sink Node, it mark packet with a frame ID. The frame ID is using unsigned long integer type.

3.7 Data Packet and Transmit

The Arduino board needs a library Xbee.h to communicate with Xbees in API mode. This library includes support for the majority of packet types, including: TX/RX, AT Command, Remote AT, I/O Samples and Modem Status. /13/

The function to transmit data with Xbee in API mode used in this project was

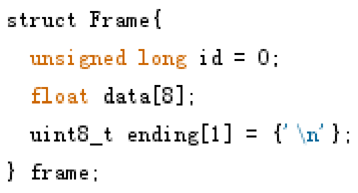
```
Tx16Request(uint16_t addr16, uint8_t *payload, uint8_t payloadLength);
```

This function has three parameters: 16 bit destination address, data frame, and length of data frame.

When sending the packet, frame ID is of the integer type, eight pieces of RMS values are of the float type, and the frame needs an ‘enter’ (‘\n’) at the end to inform the sink node that the frame ends. So a struct type was necessary:

In C language, structure provides a way to combine data items of different kinds under one name in a block of memory. Using structure in C language programming makes it a more modular program. In this project, struct type stores unsigned long integer type, float type and uint8_t (a type of unsigned integer of length 8 bits) type under the name Frame, then a single pointer is used to access these variables.

Here is an example of how the struct type was used in this project:



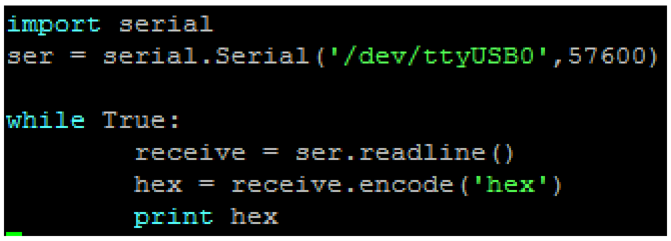
```
struct Frame{
  unsigned long id = 0;
  float data[8];
  uint8_t ending[1] = {'\n'};
} frame;
```

The Xbee module should be disconnected when the code is upload to the Arduino, otherwise the serial will conflict.

The Xbee module was connected with the Arduino board, then the Arduino board detected the sound level in RMS value and sent the RMS value in a packet through the Xbee module in the API mode.

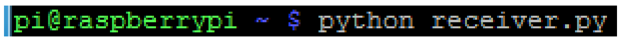
3.8 Data Receiver and Unpacking

The data receiver was done in the Raspberry Pi Sink Node. The method readline() reads one entire line from input. The following code gives an example of readline() method:



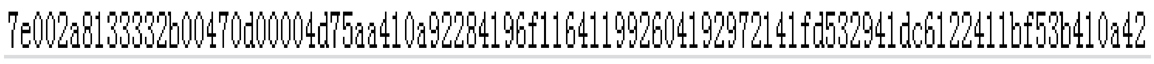
```
import serial
ser = serial.Serial('/dev/ttyUSB0',57600)

while True:
        receive = ser.readline()
        hex = receive.encode('hex')
        print hex
```

Save this code to file receiver.py and run receiver.py:



```
pi@raspberrypi ~ $ python receiver.py
```

The receiver data may like:



```
7e002a8133332b00470d00004d75aa410a92284196f116411992604192972141fd532941dc6122411bf53b410a42
```

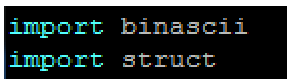
See Figure 11. API type: RX (Receive) Packet: 16-bit Address to analyse the packet. The RMS value is presented in IEEE 754 format. The results are shown in Table 8.

Table 8. Unpack packet

Start: 7E	RMS1: 4d75aa41 (21.307)
Length: 00 2a (42)	RMS2: 0a922841 (10.535)
API type: 81	RMS3: 96f11641 (9.433)
Source address: 3333	RMS4: 19926041 (14.035)
RSSI: 2b	RMS5: 92972141 (10.099)
Option: 00	RMS6: fd532941 (10.583)
ID: 470d0000 (3399)	RMS7: dc612241 (10.148)
RMS X 8	RMS8: 1bf53b41 (11.747)
Enter: 0a	
Checksum: 42	

3.9 Unpacking IEEE 754 Format

In the Python language, the IEEE 754 binary was converted to the float type that needs the library *binascii* and *struct.*



```
import binascii
import struct
```

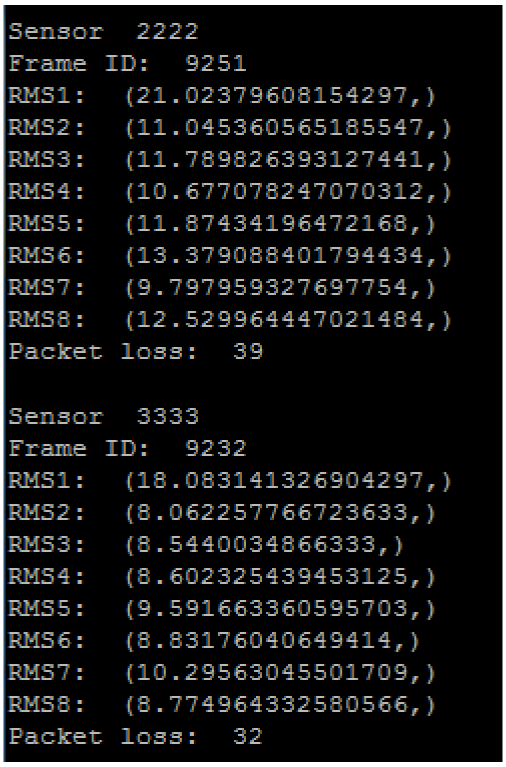
3.10 Packet Loss

According to the frame ID, the packet loss can be calculated easily.

The idea to calculate the packet loss is to mark the ID of the first frame as Start ID.

Once a packet is in, its frame ID should be analysed and the next ID is predicted, if the new coming frame ID is not the same as the predicted one, the difference between of the predicted ID and the real ID is the packet loss.

An experiment was made to measure the packet loss rate: the system ran around 2.5 hours and showed its packet loss. The result is shown in Figure 20.



```
Sensor  2222
Frame ID:  9251
RMS1:  (21.02379608154297,)
RMS2:  (11.045360565185547,)
RMS3:  (11.789826393127441,)
RMS4:  (10.677078247070312,)
RMS5:  (11.87434196472168,)
RMS6:  (13.379088401794434,)
RMS7:  (9.797959327697754,)
RMS8:  (12.529964447021484,)
Packet loss:  39

Sensor  3333
Frame ID:  9232
RMS1:  (18.083141326904297,)
RMS2:  (8.062257766723633,)
RMS3:  (8.5440034866333,)
RMS4:  (8.602325439453125,)
RMS5:  (9.591663360595703,)
RMS6:  (8.83176040649414,)
RMS7:  (10.29563045501709,)
RMS8:  (8.774964332580566,)
Packet loss:  32
```

Figure 20. Packet loss

Sensor 2222 received 9251 packets and lost 39 packets, the packet loss rate is 0.42%.

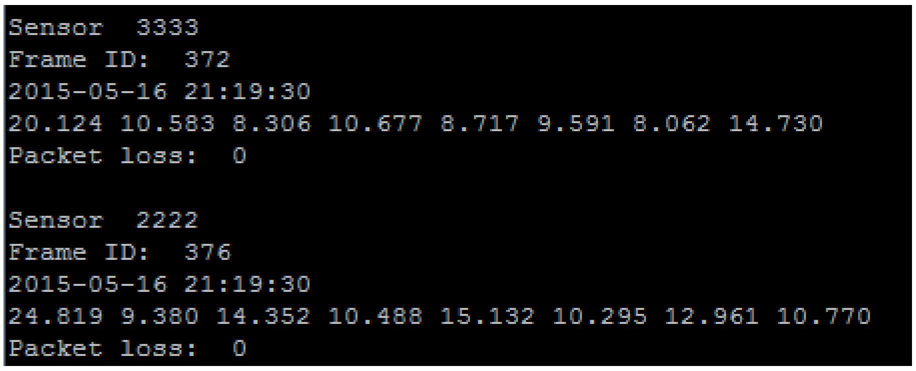
Sensor 3333 received 9232 packets and lost 32 packets, the packet loss rate is 0.34%.

3.11 Packet Time Stamp

The sink node will display the frame with its timestamp. The time is in ISO 8061 format.

3.12 Displaying Data

The sink node can display data to the screen, including: sensor ID, frame ID, timestamp, RMS values and packet loss, as shown in Figure 21.



```
Sensor  3333
Frame ID:  372
2015-05-16 21:19:30
20.124 10.583 8.306 10.677 8.717 9.591 8.062 14.730
Packet loss:  0

Sensor  2222
Frame ID:  376
2015-05-16 21:19:30
24.819 9.380 14.352 10.488 15.132 10.295 12.961 10.770
Packet loss:  0
```

Figure 21. Display data

3.13 Record to Logfile

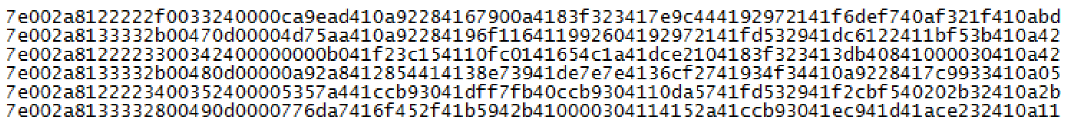
The received data will be recorded into two .csv file:

The first one is the log file, which records the packet in the hex value. The second one is to record the unpacked and decoded data, including: sensor ID, frame ID, received time, and RMS values.

In Python, a library csv was imported to realize this function,

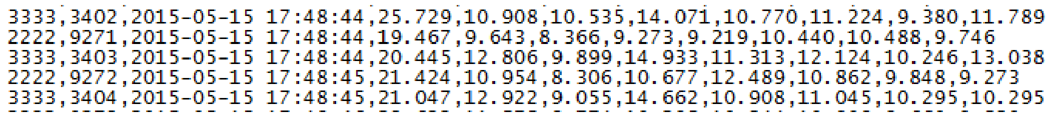
```
import csv
```

The results of log files are shown in Figure 22 and Figure 23.



```
7e002a8122222f0033240000ca9ead410a92284167900a4183f323417e9c444192972141f6def740af321f410abd
7e002a8133332b00470d00004d75aa410a92284196f1164119926041929721 41fd532941dc6122411bf53b410a42
7e002a812222330034240000000b041f23c154110fc0141654c1a41dce2104183f323413db40841000030410a42
7e002a8133332b00480d00000a92a8412854414138e73941de7e7e4136cf2741934f34410a9228417c9933410a05
7e002a8122223400352400005357a441ccb93041dff7fb40ccb9304110da5741fd532941f2cbf540202b32410a2b
7e002a813333280049 0d0000776da7416f452f41b5942b41000030411415 2a41ccb93041ec941d41ace232410a11
```

Figure 22. Logfile



```
3333,3402,2015-05-15 17:48:44,25.729,10.908,10.535,14.071,10.770,11.224,9.380,11.789
2222,9271,2015-05-15 17:48:44,19.467,9.643,8.366,9.273,9.219,10.440,10.488,9.746
3333,3403,2015-05-15 17:48:44,20.445,12.806,9.899,14.933,11.313,12.124,10.246,13.038
2222,9272,2015-05-15 17:48:45,21.424,10.954,8.306,10.677,12.489,10.862,9.848,9.273
3333,3404,2015-05-15 17:48:45,21.047,12.922,9.055,14.662,10.908,11.045,10.295,10.295
```

Figure 23. Record file

4 CONCLUSION

In this project, a low-cost wireless sound detection system was build based on two parts: Sound detection application and Xbee API mode programming.

This project provides a prototype of sound based monitor system. This system can be used to monitor a specific environment or call the recorded data for further analysis.

We can draw the following conclusions:

1. In this project, several fields were involved: acoustics, hardware design, software programming, and wireless communication.
2. A component of devices may affect the whole system.
3. The remote sensor system can be widely developed for its low cost and easy built.

This system can be improved in several ways:

1. Filter function: According to the devices used in this project, the Arduino's sample rate is 8776Hz, which means the system only supports sound frequency less than 4338Hz. So a low path filer can be used here.
2. Improve packet loss calculation: In this project, the packet loss was calculated according to the frame ID generated by the Sensor Node, the frame ID may overflow and reset to zero.
3. Data analysis interface: The log file was recorded in a .csv file in the Sink Node. The non-admin user may not access the log file, so a web based interface can be made to public the records.

REFERENCES

/1/ Time Constants and Time Weightings, Acoustic glossary

http://www.acoustic-glossary.co.uk/time-weighting.htm

/2/ A Dictionary of Physics (6 ed.). Oxford University Press. 2009. ISBN 9780199233991.

http://www.oxfordreference.com/view/10.1093/acref/9780199233991.001.0001/acref-9780199233991-e-2676

/3/ IEEE 802.15.4 standards, IEEE Standards Association

https://standards.ieee.org/getieee802/download/802.15.4-2011.pdf

/4/ Overview, Arduino Mega 2560 introduction

http://arduino.cc/en/guide/introduction

/5/ Summary, Arduino Mega 2560 introduction

http://arduino.cc/en/guide/introduction

/6/ Cellan-Jones, Rory (5 May 2011), BBC News.

http://www.bbc.co.uk/blogs/legacy/thereporters/rorycellanjones/2011/05/a_15_computer_to_inspire_young.html

/7/ Raspberry Pi data sheet

https://www.adafruit.com/datasheets/pi-specs.pdf

/8/ Introduction, XBee/XBee-Pro RF modules data sheet, Page 4, Sparkfun

https://www.sparkfun.com/datasheets/Wireless/Zigbee/XBee-Datasheet.pdf

/9/ Specifications, XBee/XBee-Pro RF modules data sheet, Page 5, Sparkfun

https://www.sparkfun.com/datasheets/Wireless/Zigbee/XBee-Datasheet.pdf

/10/ What is API, Knowledge Base, Digi

http://knowledge.digi.com/articles/Knowledge_Base_Article/What-is-API-Application-Programming-Interface-Mode-and-how-does-it-work

/11/ API Operation, XBee/XBee-Pro RF modules data sheet, Page 57, Sparkfun

https://www.sparkfun.com/datasheets/Wireless/Zigbee/XBee-Datasheet.pdf

/12/ Input and Ourput, Arduino Mega 2560 introduction

http://www.arduino.cc/en/Main/ArduinoBoardMega2560

/13/ Overview, xbee-arduino, Google Code

https://code.google.com/p/xbee-arduino/

APPENDIX 1. SENDER

```
// seneor
int sample;
long int acc = 0;
double rms = 0;
int i;

// LED
int flag = 0;
int time0 = 0;
int time1 = 0;
int differ = 0;

// XBee
#include <XBee.h>
XBee xbee = XBee();
int j = 0;
struct Frame{
  unsigned long id = 0;
  float data[8];
  uint8_t ending[1] = {'\n'};
} frame;

void setup() {
  xbee.begin(57600);
  pinMode(13, OUTPUT);
}

void loop() {
  // sensor part
  for(i=0;i<1096;i++){
    sample = analogRead(A7) - 512;
    acc = acc + sample * sample;
  }
  rms = sqrt(acc / 1096);
  acc = 0;

  // frame
  frame.data[j] = rms;
  j = j + 1;
  // turn LED
  if (rms > 30){
    digitalWrite(13,HIGH);
    flag = 1;
    time0 = millis();
  }
  time1 = millis();
  differ = time1 - time0;
  if (differ > 3000 && flag == 1){
    digitalWrite(13,LOW);
    flag = 0;
  }

  // XBee
  if(j==8){
    frame.id = frame.id + 1;
    uint8_t *payload;
    payload = (uint8_t *)&frame;
    Tx16Request packet = Tx16Request(0x1111, payload, sizeof(frame));
    xbee.send(packet);
    j = 0;
  }

}
```

APPENDIX 2. RECEIVER

```
#XBee
import serial
ser = serial.Serial('/dev/ttyUSB0',57600)

#IEEE754 to float
import binascii
import struct

#packet loss
#node2
idflag2 = 0
startid2 = 0
currentid2 = 0
nextid2 = 0
loss2 = 0
#node3
idflag3 = 0
startid3 = 0
currentid3 = 0
nextid3 = 0
loss3 = 0

#time
from time import strftime

#file
import csv
c = csv.writer(open("record.csv", "a"))
l = csv.writer(open("logfile.csv", "a"))

#first string
while True:
    #receiver
    receive = ser.readline()
    hex = receive.encode("hex")
    #print hex

    #track
    start = '7e002a81'
    check =    hex.find(start,1)
```

```
        #print check

        if check == 2:
                string1 = hex[0:check]
                string2 = hex[check:len(hex)]
                buffer = string2
                loss2 = 1
                loss3 = 1
                break

while True:
        #receiver
        receive = ser.readline()
        hex = receive.encode("hex")
        #print hex

        #track
        start = '7e002a81'
        check =    hex.find(start,1)
        #print check

        if check ==2:
                string1 = hex[0:check]
                string2 = hex[check:len(hex)]
                buffer = buffer + string1
                #print 'Received: ' + buffer
                sensor = buffer[8:12]
                idstring = buffer[22:24] + buffer[20:22] + buffer[18:20] + buffer[16:18]
                id = int(idstring,16)
                RMS1 = struct.unpack('<f', binascii.unhexlify(buffer[24:32]))
                RMS1value = str(RMS1)[1:len(str(RMS1))-2]
                RMS2 = struct.unpack('<f', binascii.unhexlify(buffer[32:40]))
                RMS2value = str(RMS2)[1:len(str(RMS2))-2]
                RMS3 = struct.unpack('<f', binascii.unhexlify(buffer[40:48]))
                RMS3value = str(RMS3)[1:len(str(RMS3))-2]
                RMS4 = struct.unpack('<f', binascii.unhexlify(buffer[48:56]))
                RMS4value = str(RMS4)[1:len(str(RMS4))-2]
                RMS5 = struct.unpack('<f', binascii.unhexlify(buffer[56:64]))
                RMS5value = str(RMS5)[1:len(str(RMS5))-2]
                RMS6 = struct.unpack('<f', binascii.unhexlify(buffer[64:72]))
                RMS6value = str(RMS6)[1:len(str(RMS6))-2]
                RMS7 = struct.unpack('<f', binascii.unhexlify(buffer[72:80]))
                RMS7value = str(RMS7)[1:len(str(RMS7))-2]
```

```
RMS8 = struct.unpack('<f', binascii.unhexlify(buffer[80:88]))
RMS8value = str(RMS8)[1:len(str(RMS8))-2]
if RMS1value.find('.',1) == 2:
        rms1 = RMS1value[0:6]
if RMS1value.find('.',1) == 1:
        rms1 = RMS1value[0:5]
if RMS2value.find('.',1) == 2:
        rms2 = RMS2value[0:6]
if RMS2value.find('.',1) == 1:
        rms2 = RMS2value[0:5]
if RMS3value.find('.',1) == 2:
        rms3 = RMS3value[0:6]
if RMS3value.find('.',1) == 1:
        rms3 = RMS3value[0:5]
if RMS4value.find('.',1) == 2:
        rms4 = RMS4value[0:6]
if RMS4value.find('.',1) == 1:
        rms4 = RMS4value[0:5]
if RMS5value.find('.',1) == 2:
        rms5 = RMS5value[0:6]
if RMS5value.find('.',1) == 1:
        rms5 = RMS5value[0:5]
if RMS6value.find('.',1) == 2:
        rms6 = RMS6value[0:6]
if RMS6value.find('.',1) == 1:
        rms6 = RMS6value[0:5]
if RMS7value.find('.',1) == 2:
        rms7 = RMS7value[0:6]
if RMS7value.find('.',1) == 1:
        rms7 = RMS7value[0:5]
if RMS8value.find('.',1) == 2:
        rms8 = RMS8value[0:6]
if RMS8value.find('.',1) == 1:
        rms8 = RMS8value[0:5]

time = strftime("%Y-%m-%d %H:%M:%S")
print 'Sensor ', sensor
print 'Frame ID: ', id
print time
print rms1, rms2, rms3, rms4, rms5, rms6, rms7, rms8
if len(buffer) == 92:
        l.writerow([buffer])
if len(buffer) == 184:
```

```
                l.writerow([buffer[0:92]])
                l.writerow([buffer[92:184]])
        c.writerow([sensor, id, time, rms1, rms2, rms3, rms4, rms5, rms6, rms7, rms8])

        buffer = string2

        #packet loss
        #node2
        if sensor == '2222' and idflag2 == 0:
                startid2 = int(id)
                currentid2 = int(id)
                nextid2 = int(currentid2) + 1
                idflag2 = 1
        if sensor == '2222' and idflag2 == 1:
                #print '2 start id: ', startid2
                currentid2 = int(id)
                #print '2 current id: ', currentid2
                if int(currentid2) != int(nextid2):
                        loss2 = loss2 + int(currentid2) - int(nextid2)
                print 'Packet loss: ', loss2
                nextid2 = int(currentid2) + 1
                #print '2 next id: ', nextid2
                print ''
        #node3
        if sensor == '3333' and idflag3 == 0:
                startid3 = int(id)
                currentid3 = int(id)
                nextid3 = int(currentid3) + 1
                idflag3 = 1
        if sensor == '3333' and idflag3 == 1:
                #print '3 start id: ', startid3
                currentid3= int(id)
                #print '3 current id: ', currentid3
                if int(currentid3) != int(nextid3):
                        loss3 = loss3 + int(currentid3) - int(nextid3)
                print 'Packet loss: ', loss3
                nextid3 = int(currentid3) + 1
                #print '3 next id: ', nextid3
                print ''

if check == -1:
        string3 = hex
        buffer = buffer + string3
```

```
if check != 2 and check != -1:
        string4 = hex[0:check]
        string5 = hex[check:len(hex)]
        buffer = buffer + string4
        #print 'Received: ' + buffer
        sensor = buffer[8:12]
        idstring = buffer[22:24] + buffer[20:22] + buffer[18:20] + buffer[16:18]
        id = int(idstring,16)
        RMS1 = struct.unpack('<f', binascii.unhexlify(buffer[24:32]))
        RMS1value = str(RMS1)[1:len(str(RMS1))-2]
        RMS2 = struct.unpack('<f', binascii.unhexlify(buffer[32:40]))
        RMS2value = str(RMS2)[1:len(str(RMS2))-2]
        RMS3 = struct.unpack('<f', binascii.unhexlify(buffer[40:48]))
        RMS3value = str(RMS3)[1:len(str(RMS3))-2]
        RMS4 = struct.unpack('<f', binascii.unhexlify(buffer[48:56]))
        RMS4value = str(RMS4)[1:len(str(RMS4))-2]
        RMS5 = struct.unpack('<f', binascii.unhexlify(buffer[56:64]))
        RMS5value = str(RMS5)[1:len(str(RMS5))-2]
        RMS6 = struct.unpack('<f', binascii.unhexlify(buffer[64:72]))
        RMS6value = str(RMS6)[1:len(str(RMS6))-2]
        RMS7 = struct.unpack('<f', binascii.unhexlify(buffer[72:80]))
        RMS7value = str(RMS7)[1:len(str(RMS7))-2]
        RMS8 = struct.unpack('<f', binascii.unhexlify(buffer[80:88]))
        RMS8value = str(RMS8)[1:len(str(RMS8))-2]
        if RMS1value.find('.',1) == 2:
                rms1 = RMS1value[0:6]
        if RMS1value.find('.',1) == 1:
                rms1 = RMS1value[0:5]
        if RMS2value.find('.',1) == 2:
                rms2 = RMS2value[0:6]
        if RMS2value.find('.',1) == 1:
                rms2 = RMS2value[0:5]
        if RMS3value.find('.',1) == 2:
                rms3 = RMS3value[0:6]
        if RMS3value.find('.',1) == 1:
                rms3 = RMS3value[0:5]
        if RMS4value.find('.',1) == 2:
                rms4 = RMS4value[0:6]
        if RMS4value.find('.',1) == 1:
                rms4 = RMS4value[0:5]
        if RMS5value.find('.',1) == 2:
                rms5 = RMS5value[0:6]
```

```
if RMS5value.find('.',1) == 1:
        rms5 = RMS5value[0:5]
if RMS6value.find('.',1) == 2:
        rms6 = RMS6value[0:6]
if RMS6value.find('.',1) == 1:
        rms6 = RMS6value[0:5]
if RMS7value.find('.',1) == 2:
        rms7 = RMS7value[0:6]
if RMS7value.find('.',1) == 1:
        rms7 = RMS7value[0:5]
if RMS8value.find('.',1) == 2:
        rms8 = RMS8value[0:6]
if RMS8value.find('.',1) == 1:
        rms8 = RMS8value[0:5]

time = strftime("%Y-%m-%d %H:%M:%S")
print 'Sensor ', sensor
print 'Frame ID: ', id
print time
print rms1, rms2, rms3, rms4, rms5, rms6, rms7, rms8
if len(buffer) == 92:
        l.writerow([buffer])
if len(buffer) == 184:
        l.writerow([buffer[0:92]])
        l.writerow([buffer[92:184]])
c.writerow([sensor, id, time, rms1, rms2, rms3, rms4, rms5, rms6, rms7, rms8])

buffer = string5

#packet loss
#node2
if sensor == '2222' and idflag2 == 0:
        startid2 = int(id)
        currentid2 = int(id)
        nextid2 = int(currentid2) + 1
        idflag2 = 1
if sensor == '2222' and idflag2 == 1:
        #print '2 start id: ', startid2
        currentid2 = int(id)
        #print '2 current id: ', currentid2
        if int(currentid2) != int(nextid2):
                loss2 = loss2 + int(currentid2) - int(nextid2)
        print 'Packet loss: ', loss2
        nextid2 = int(currentid2) + 1
        #print '2 next id: ', nextid2
        print ''
#node3
if sensor == '3333' and idflag3 == 0:
        startid3 = int(id)
        currentid3 = int(id)
        nextid3 = int(currentid3) + 1
        idflag3 = 1
if sensor == '3333' and idflag3 == 1:
        #print '3 start id: ', startid3
        currentid3= int(id)
        #print '3 current id: ', currentid3
        if int(currentid3) != int(nextid3):
                loss3 = loss3 + int(currentid3) - int(nextid3)
        print 'Packet loss: ', loss3
        nextid3 = int(currentid3) + 1
        #print '3 next id: ', nextid3
        print ''
```