The Pennsylvania State University

The Graduate School

Department of Computer Science & Engineering

SENSOREAR: A SENSOR NETWORK BASED VOICE EAVESDROPPING SYSTEM

A Thesis in

Computer Science and Engineering

by

Ge Ruan

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

December 2008

The thesis of Ge Ruan was reviewed and approved* by the following:

*Signatures are on file in the Graduate School

ABSTRACT

SENSOREAR: A SENSOR NETWORK BASED VOICE EAVESDROPPING SYSTEM

by

Ge Ruan

Master of Science in Computer Science & Engineering,
Pennsylvania State University, University Park
Professor Sencun Zhu, Research Advisor

In recent years, the research and application on sensor network became a very hot topic. People use sensors to collect information from environment, which is not accessible or not convenient to access. In this thesis, by building up a sensor network based real time eavesdropping system, called *SensorEar*, we show that sensor network can bring up very serious and detailed threat to personal privacy. We implement the system on Mica2 of Crossbow Inc., which is a small, cheap, resource limited, commercial-off-the-shelf, and TinyOS based platform. The *SensorEar* system features on its high data rate, which exceeds the limitation of the Mica2. It is achieved by the application of a self-designed multi-rate compression algorithm, and a multi-hop and multi-channel transmission scheme. The performance of this system is evaluated on several aspects such as sampling rate, transmission rate, packet losing rate and sound quality.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

The author would like to thank my adviser Professor Sencun Zhu for his help and instruction, committee member Professor Guohong Cao for his review and suggestion, and graduate student Soumya Jain for his help in sound quality evaluation.

.

Chapter 1

Introduction

Improvements in wireless communication and electronic have enabled the development of low cost, low power sensor nodes which are small in size, light in weight and possess computation and communication ability and multi functionalities in sensing. The above-described functionalities of sensors develop into a wide range of applications of sensor networks.

Generally, sensors are deployed in inaccessible environment to collect some kinds of information such as light, temperature, humidity, velocity and so on. At first, sensor network was basically applied in military and research. For example, in military use [1], Gyula Simon et al. used pre-deployed sensors to sense a gunshot and estimate the location of the sniper. The information collected by all the sensors will be combined together to provide an estimation of the sniper's location. Alan Mainwaring et al. applied sensor networks in ecology research [2]. In their project, environmental sensor networks are implemented to monitor the habitat, identify, track and measure the population of birds and other species.

Later on, researchers started to exploit the use of sensor network in people’s daily life. In 2005, Victor Shnayder et al.[3], developed a combined hardware and software platform for medical sensor network, called *CodeBlue*. By outfitting every patient with tiny, wearable wireless sensors, nurses and doctors can continuously monitor the status of the patients remotely.

Certainly, sensor network can bring more convenience to our life before long. But as every coin has two sides, sensor network also brings privacy threat as well, since it also provides convenience for malicious people to spy on others and it is too small to be noticed.

For example, the Mica2 mote of *Crossbow Inc.*, which we use in our project, is 58 x 32 x 7 (mm) big and weighs 18 grams, excluding the battery pack. And with the development of electronic technologies, they are getting smaller and cheaper. In 2001, Kris Pister et al. finished a project named *Smart Dust* [4], and they also set up a company named Dust Network to commercialize this technology. A Smart Dust is around the size of a grain and it contains sensors, computational ability, bi-directional wireless communications, and a power supply. Dr. Pister envisioned that in the future the smart dust would be as cheap as around $1.

With the price of sensor reduced to low enough, it is monetarily possible for any person to build his own sensor network. Since we can use sensors to monitor birds, patients and enemies, of course others can use them to monitor our activities.

Imagine the following scenario. A student deploys some programmed smart dusts in his advisor’s office as sensing nodes and drops some smart dusts along the path from the advisor’s office to his office as intermediate nodes. Then he is able to monitor the advisor from his office through a computer, connected with a gateway device as a base station of the sensor network for data collection. Contrast to the traditional concept of privacy in computer science, this is a direct threat to personal privacy from the development of sensor technology. In this thesis, we are interested in how severe this direct threat can be.

What if some people can use sensors to record your conversation in real-time? That is such a serious threat to privacy that we can not ignore. But among all the sensor network applications, audio is the least exploited direction. This is because of that audio applications feature in high data rate and massive data volume, while the sensors’ resources are very limited.

Generally, the microphone on the sensor board is supposed to be used to detect the existence of a sound, for example in [1]. The default provided microphone’s sampling module of TinyOS1.x only returns a Boolean value to indicate whether a sound is heard. To some extent, we believe that the current popular mote is not designed for sound recording.

To prove sensor network’s threat to personal privacy, we show the feasibility of this system in the following aspects such as size, cost, scalability, longevity, etc. And we pick Mica2 as our platform, which is lower-end in all aspects comparing to other sensors.

The contribution of this thesis is that we proved the feasibility of using sensor network to eavesdrop on somebody’s conversation by implementing a sensor network based eavesdropping system. We address the research challenges stemming from the contrast between high data volume of audio applications and relative low platform transmission rate. We design a multi-rate compression algorithm and a multi-channel-multi-hop network model for this application. These new designs help us to achieve a high data rate that exceeds 19.2 kbps, the maximum transmission rate of Mica2.

The rest of this thesis is organized as follows. In Chapter 2, we introduce the platforms, and research challenges. In Chapter 3, we introduce the related works. In Chapter 4, we introduce the MRC (Multi Rate Compression) algorithm, which is specifically designed by us for this project. In Chapter 5, we introduce the design consideration, system architecture, modules, control flow and all the details in the implementation. And in Chapter 6, we analyze the performance of the experiment. In Chapter 7, we discuss the bottleneck of this experiment and future work. Finally, in Chapter 8, we make the conclusion.

Chapter 2

Platform and Research Challenge

2.1 Mica2

The Mica2 mote is commercially available from Crossbow Technology. It is the 3rd-generation mote module used for enabling low-power wireless sensor networks. The Mica2 mote measures 58 mm x 32 mm x 7 mm, as it is shown in Figure 2.1. Usually, it is sold with an attached battery pack containing 2 AA batteries, which are used to power the Mica2 circuitry. Using no power conservation algorithms, a typical continuous application on the Mica2 lasts just under one week. With power management, it is possible to work up to one year.

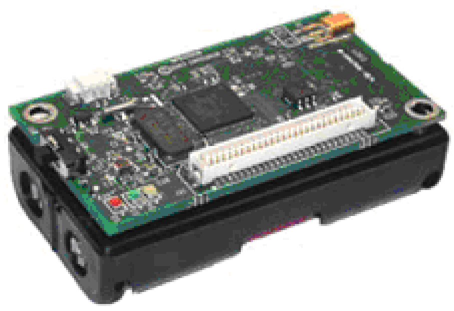


Figure 2.1 Appearance of Mica2 (Scale: 1:1)

Mica2 provides communication, most of the processing and most of the sensor interfaces. Designed as an inexpensive, off-the-shelf platform for wireless sensor networks, it is based on the Atmel ATmega128L microcontroller [14], which is a low-power AVR 8-bit processor with 128 Kbytes of flash program memory, 4 Kbytes of internal SRAM. The ATmega128L also includes an 8-channel 10-bit ADC, three hardware timers, and several bus interfaces including SPI, I2C, and two USARTs.

Besides the Atmega128L, Mica2 is also equipped with a 512 k bytes external flash memory and a CC1000 chip to provide larger storage space and radio communication ability. The block diagram of Mica2 is shown as below.

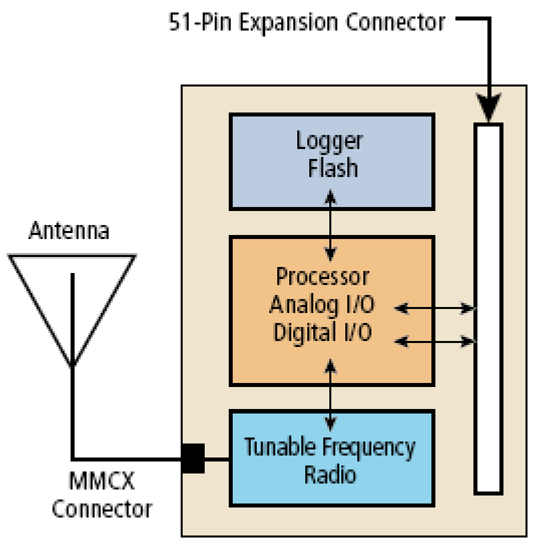


Figure 2.2 Block Diagram of Mica2

The integrated ChipCon CC1000 radio is one of the unique features of the Mica2 motes. It is available in 315, 433, or 868/916 MHz bands for low-power communication.

The CC1000 has a maximum bandwidth of 76.8 Kbaud and ranges up to 1000 ft, which have been demonstrated outdoors. Indoor ranges depend heavily on the surrounding environment but ranges of 50-100 ft are not uncommon. There are up to 50 radio channels that are available. And the RF output power and mode is adjustable to increase or decrease the communication radius and adjust the life cycle of the battery.

Received signal strength is also readable on the CC1000 and its use for distance sensing has been attempted in the past [15]. The radio may also be used to remotely program the Mica mote [16]. When using more than fifty robots for an application, network programming is essential for the sanity of the programmer and provides a substantial ease-of-use advantage.

Three LEDs are also provided for visual debugging. The 51-pin expansion connector provides interface for sensor board. By attaching different sensor board, we can get various sensing functionalities. In this project, we use the basic sensor board, which provides photo-sensor, temperature sensor, microphone and a 4 kHz buzzer.

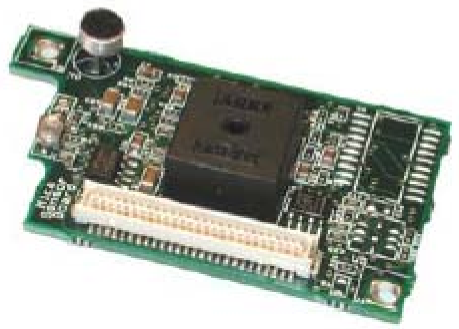


Figure 2.3 Appearance of Basic Sensor Board (Scale 1:1)

2.2 TinyOS

TinyOS [17], which is originally developed at UC Berkeley, is an open-source operating system designed for wireless embedded sensor networks. It features a component-based architecture, which enables rapid innovation and implementation while minimizing code size as required by the severe memory constraints inherent in sensor networks.

TinyOS is an event-driven and component-based operating system. Its event-driven nature is intended to help it interact with an uncertain environment – a useful feature for sensor nodes that interact heavily with their environment. Event driven interface simulates the way people work in the real world. It helps the developer to write well-structured object-oriented codes.

The component-based aspect of TinyOS results in applications that are built from reusable software components, which can also be used to abstract lower level functionality. Many pre-written components along with software tools are provided online at the TinyOS Sourceforge repository [18].

But a disadvantage of event-driven systems is their difficulty satisfying real-time requirements. All the tasks run one by one in a serial mode.

2.3 Research Challenges

Comparing to other platforms like MicaZ, Mica2 is much weaker in several aspects. And the limitations of the hardware and software platforms make our project more interesting. And the solutions on these limitations are the selling points of our design.

First of all, the TinyOS doesn’t provide in-build module for voice recording. The microphone on the sensor board is only supposed to sense the existence of sound. The in-build microphone-control module only provides a binary function returning true or false. So we should implement a module to control microphone for recording.

Secondly, in telephony, the usable voice frequency band ranges from approximately 300 Hz to 3400 Hz and the sampling rate is 8k Hz. While the resolution of the sensor's application layer timer is only 1 millisecond, far from enough to reach the required sample rate. This is the other challenge of this project. A more precise hardware clock based timer is necessary.

Thirdly, the memory of sensor is small i.e. 4k. With a 5k sampling rate and 1 byte/ sample, we will get 5k bytes in one second what is bigger than the RAM. So we should buffer the sampling data in the flash before transmitting it. How to read and write the flash in a most efficient way is the third problem to solve.

Even we can use the external flash for temporary data storing, its size is still very small, which is 512k bytes. The assumption of a 512 Mbytes external storage of [6] is not feasible currently. Ergo we should send out these sampling data in time before next sampling round starts so that we can reuse the limited storage space. It requires a transmission rate as high as or even higher than the sampling rate, i.e. 40k bps when the sampling rate is 5k bytes per second. But the transmission rate of Mica2 is at most 38.4 k baud under Manchester Code modulation, i.e. the actually maximum data rate is 19.2 k. Obviously, the transmission rate is much lower than sampling rate. This is the major challenging part of this project. So we come up with the compression idea that all the data are compressed before sending. We designed a specific data compression algorithm for this system. The implementation of compression algorithm greatly reduces the number of bytes to be sent and consequently saves the power of the sensors. As we all know, power is one of the most crucial aspect in sensor network design and data transmission is the most power-consuming task in sensor network.

It should be noticed that the mote is not able to sample when it is compressing or sending data. Because the mote is a serial system, the tasks work one by one and switching between different modules will be time costing. And the radio module is activated in every byte period, which is approximately 26 microseconds, to check the channel vacancy. To get high sampling rate, we have to shutdown the radio. It implies that a single sensor is not enough to realize continuous recording since it's not able to cover the whole sampling period. So Collaboration of multi sensors is required.

Finally, after we use multi sensors in sampling, we get a new problem. Because all the motes share the wireless channel, it becomes very "crowded" if some motes try to send packets at the same time in the same channel. Collision will results in packet losing and packet losing means worse sound quality. On the other hand, to achieve such a high data rate, the motes are transmitting data almost at their highest speed that it's not possible to apply hand-shaking, resending or other reliable transmission control mechanism. So our system model should provide relatively reliable transmission as well as high transmission rate. To avoid collision, we use multi channels to alleviate the high traffic on the default channel.

The details will be discussed later in the chapter of implementation.

Chapter 3

Related Works

Currently, there is not much personal privacy relevant research in the area of sensor network. Jun Han et al.’s research in [5] is a typical example. They detect the human presence by sensing the environment humidity. But information of presence is kind of brief, which may not be crucial enough to catch people’s attention. That’s why we implement *SensorEar* to show that privacy threat from sensors can be very serious.

In 2007, Liqiang Luo et al [6] presented EnviroMic, which is a novel distributed acoustic monitoring system. Their idea is somewhat similar to ours. But the prototype and platform are both different. Their project is based on an assumption that base station is out of reach and the mote is equipped with a 512 M flash memory to work in disconnected mode. In their scenario, all the data sampled are stored in the memory and retrieved later. And our system is a real time system. All the steps including sampling, compression and transmission are fulfilled in real time. The system can be controlled remotely through multi-hop communication.

Instead of MicaZ, which is the platform of their system, we use Mica2. The differences between these two platforms are significant. Mica2 only has a 512 kbytes flash memory, which makes it impossible to save all the sampled data in memory if the recording time is long enough.

So all the data collected must be transmitted in time before next sampling round starts. But Mica2 has a maximum transmission rate 38.4 k baud, i.e. 19.2 kbps since Manchester encoding is applied. And the default maximum sampling rate of Mica2 is 3.53 kHz.[1] Performance of MicaZ is much better. According to [6], MicaZ has a maximum transmission rate 250 kbps and the sampling rate is at least 4 kHz.

Obviously, MicaZ is more suitable for high data rate application than Mica2. We choose Mica2 as the platform in order to make the project more challengeable and meaningful. If our prototype can work well on Mica2, then it works better on other more powerful platforms.

[1] We can increase the maximum sample rate to 7k by double the ADC clock's frequency at a cost of worse clock resolution. In our project, we use the sample rates of 3.3 k and 5k, due to the limitation of transmission rate.

Chapter 4

Multi-Rate Compression

4.1 Motivation of compression

In the experiment, the minimum sampling rate we use is 3.3 k. It means we will get 3.3 k bytes per second since every sample possesses one byte [2]. Thus the input data rate is 26.4 kbps, which exceeds 19.2 kbps, the upper bound of Mica2's transmission rate. We use multi sensors in sampling so that when one mote is sending data, others can cover its sampling gap. But it still has to send out all the data in time before next mote start sending data. In other words, the output speed must be faster than the input speed. Otherwise the data will accumulate and the system will be jammed.

Audio compression algorithms always have high compression rate. It comes from the feature of audio files. So we come up with the idea of compression. It saves time in transmission. And on the other hand, it saves the power, which is the most important factor in sensor network application, since radio transmission is the most dominant part of power consuming in sensor network.

4.2 Sound's Frequencies

Sound is vibration transmitted through a solid, liquid, or gas. Particularly, sound means

[2] Actually, each sample got by ADC is 10 bits. To increase the efficiency, we just right shift it twice to get the first 8 bits.

those vibrations composed of frequencies capable of being detected by ears. [19]

Generally, the accepted standard range of audio frequencies is from 20 to 20000 Hz. Frequencies above 20000 Hz can sometimes be sensed by young people. The following Table 3.1 may help us to have a clear idea of the frequencies range of normal sounds.[20]

MIDI Note	**Frequency (Hz)**	**Description**
C-2	4.09	Lowest note for Gregg Bailey's 64' PVC subcontrabass clarinet
C-1	8.18	Lowest organ note
C0	16.35	Lowest note for tuba, large pipe organs, Bösendorfer Imperial Grand Piano
C1	32.70	Lowest C on a standard 88-key piano.
C2	65.41	Lowest note for cello
C3	130.81	Lowest note for viola, mandola
C4	261.63	Middle C
C5	523.25	Lowest note for a piccolo.
C6	1046.50	Approximately the highest note reproducible by the average female human voice.
C7	2093	Highest note for a flute.
C8	4186	Highest note on a standard 88-key piano.
C9	8372	
C10	16744	Approximately the tone that a typical CRT television emits while running.

Table 4.1 Frequencies of MIDI notes

In the experiments, we use two sampling rate: 3.3 kHz and 5 kHz. According to Nyquist-Shannon Sampling Theorem, the frequency range of sound we can get with this system is between 1.65 kHz and 2.5 kHz. From Table 3.1, we know that the highest note reproducible by human voice is approximately 1046 Hz. So the standard of our system seems not too bad as long as it is applied for human speech. Table 3.2 shows the quality levels of sounds with different frequency upper bounds.

Frequency (Hz)	**Octave**	**Description**
16 to 32	1st	The human threshold of feeling, and the lowest pedal notes of a pipe organ.
32 to 512	2nd to 5th	Rhythm frequencies, where the lower and upper bass notes lie.
512 to 2048	6th to 7th	Defines human speech intelligibility, gives a horn-like or tinny quality to sound.
2048 to 8192	8th to 9th	Gives presence to speech, where labial and fricative sounds lie.
8192 to 16384	10th	Brilliance, the sounds of bells and the ringing of cymbals. In speech, the sound of the letter "S" (8000-11000 Hz)

Table 4.2 Frequency Ranges and Sound Quality

We can see that the frequency range of sound we recorded lies across the border of the third and the fourth ranges, i.e. 512~2048 and 2048~8192. So the sound is supposed to be intelligible but horn-like. This is verified by our experiment later.

And there’s something interesting that English speech has higher frequency than Chinese speech. Because in Chinese, every word is one syllable, while in English every word has multi syllables. In other words, sound of one English word has more variations than a Chinese word. And English has more words that make labial and fricative sounds, such as “thin” or “then”. Consequently, English speech contains higher frequency parts in general.

That’s why we use two sampling frequencies. 3.3 kHz is enough for Chinese speech and still reserve enough time margins for transmission. But English speech recorded at 3.3 kHz is barely clear enough. 5 kHz provides better sound quality at the cost of less time margins, which may result in packet loss occasionally.

4.3 Common compression algorithm

Usually, the performance of a compression algorithm is judged in three aspects: compression rate, compression speed, and compression latency. Compression rate is defined as the size of input divides the size of output. Compression speed is described as the proportional operations needed by the algorithm. And compression latency refers to the numbers of samples, which must be analyzed before a block of audio is processed.

According to the requirement of different scenarios, different compression algorithms are created. For example, .zip, which is the most widely used format of compressed file, is excellent in good compression rate and lossless compression. It applies the DEFLATE algorithm which is a combination of LZ77 [7] and Huffman [8]. LZ77 features in duplicate string elimination. Within compressed blocks, if a duplicate series of bytes is spotted (a repeated string), then a back-reference is inserted, linking to the previous location of that identical string instead. And the method used in the second stage of Zip compression is Hoffman coding. It works by replacing commonly used symbols with shorter representations and less commonly used symbols with longer representations. Both algorithms require knowledge on a block of data. The bigger the size of block is, the higher compression rate we can achieve. Generally the block is set to 32 k bytes. It means the latency is 32 k bytes. It is not applicable in sensor network, because the mote only has a 4 k RAM. Of course, we can use smaller block but it will reduce the compression rate too.

When referring to audio compressing, audio compression algorithms usually yield higher compression rate than general compression algorithms. These algorithms take advantage of a perceptual limitation of human hearing called auditory masking. In 1894, Mayer reported that a tone could be rendered inaudible by another tone of lower frequency.[9] These compression algorithms work by reducing accuracy of certain parts of sound that are deemed to beyond the auditory resolution ability of most people. So most of them are lossy compression. AC-3 and MP3 are typical examples of them. Comparing to PCM (Pulse Code Modulation), these algorithms' quantification is performed in frequency domain. MDCT (Modified Discrete Cosine Transform) [10] is always applied for the time domain to frequency domain transformation. The time cost of this algorithm is O(k n), where n is the input size and k is the size of the block or sliding window. Usually, k is set to 32000 bytes.This is too much for a resource limited system as sensor. So we don't use any generic algorithm or audio compression algorithm. Instead, we design a specific multi-rate compression algorithm based on the characters of our samples. It works in linear time, i.e. O(*n*), where *n* is the size of the input. And the latency is 1 byte.

4.4 Scheme of the MRC algorithm

This algorithm is a combination of RLE (run length encoding) [11] algorithm and PCM (Pulse Code Modulation)[12][3]. It borrows the ideas from both RLE and PCM. Like many other dedicated compression algorithm, it's specially designed based on the feature of the sampled data.

The original sampled data has 10 bits. And it is shifted twice to get the dominant 8-bits to fit in a byte. Thus the range of the sample is 0~255. But by observation, most of the samples lie between 116 and 122. When there's no sound, the sample values are 119. So we regard 119 as zero and subtract all the samples by 119. Then we get the values between –119 to 137. These new values are the data we are going to transmit. Table 3.3 shows the portion of these new sample data out of 60000 samples.

Value	**Number**	**Portion(%)**
0	46941	78.2
-1,1	11346	18.9
-3,-2,2,3	1434	2.4
[-119,-4], [4, 137]	251	0.5

Table 4.3 Distributions of Sample Data

From the above information, we know that 0s is the major value in the samples, and many of them are consecutive. It makes sense, because during a conversation, silence is

[3] We didn't see the original articles on these algorithms. We just referred it from some other books.

always the major part. That’s the motivation of applying RLE. In this case, we can compress 63 consecutive zeros into 1 byte at most.

But RLE doesn’t compress nonzero values at all. Nonzero values always appear consecutively. So we use the PCM to compress the nonzero values. On the other hand, most nonzero values lie between -1 and 1. For these samples, we use 3 quantification levels 0,1,2, and compress four bytes into one. For those values, which lie between -3 and 3, we use 7 quantification levels 0,1,2,3,4,5,6, and compress two bytes into one.[4] So in this case, PCM gets a compression rate between 2 and 4. It determines the possible range of total compression rate as 2~32. Generally, during a conversation, the compression rate is around 3~5. Multiplying the transmission data rate as 14~16 kbps, we can get an information rate at 42~80 kbps, which is good enough to send out the data we collected.

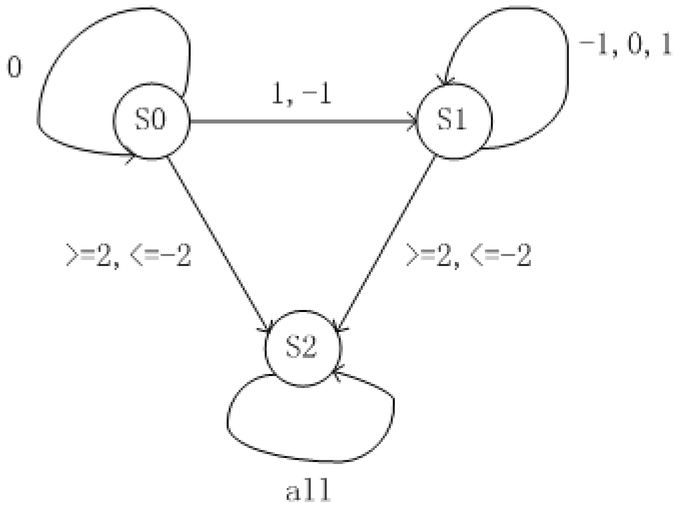


Figure 4.1 Status Switching in MRC

[4] For those samples that larger than 3 or smaller than –3, we quantified them as 3 or –3.

The procedure of the algorithm is described as below,

1. Define three statuses S0, S1, S2 which correspond to RLE algorithm, 3-level PCM and 7-level PCM. The maximum compression length of S0, S1 and S2 is 63, 4, and 2 separately. Initially, the process stays in S0 status and maintains two 1-byte variables: *result* and *counter*. *Counter* records the number of samples processed in the current status. *Result* records the current compression result of the previous several samples in count of *Counter*. Both are initialized to be 0.
2. Read the next sample, and switch the status according to Figure 4.1. If it jumps out of the current status, then maps and outputs the current *result*; resets the *counter* and the *result* before jumping. Otherwise, check whether the compression length exceeds the maximum length of the current status. If yes, map and out put the current *result*; reset the *counter* and the *result* before jumping, then jump to status S0. Else, update the *counter* and the *result*, and stay in the current status, then repeat Step 2.

Each output of the RLE algorithm lies between 1 ~ 63. Each output of the 3-level PCM lies between 0~80. And each output of the 7-level PCM lies between 0~48. Then we mapped these outputs into different range by adding different numbers to them. The output of RLE is added to 192. The output of 3-level PCM is added to 0. And the output of 7-level PCM is added to 128.

Thus the compressed data yields from different compression algorithms are mapped into different ranges before output. As such, when we received the compressed data, by checking the first two bits we can easily tell which compression algorithm is applied and how many samples we can extract from it, and then run the corresponding uncompressing algorithm on it to retrieve the original data.

Input	0 0 0 0	1 0 -1	3 -1
Status	S0	S1	S2
Intermediate Step		(3-ary) 2 1 0	(7-ary) 6 2
Result (Binary)	4 (0000 0100)	21 (0001 0101)	44 (0010 1100)
Mapping (+)	192 (1100 0000)	0 (0000 0000)	128 (1000 0000)
Output	196 (1100 0100)	21 (0001 0101)	172 (1010 1100)

Figure 4.2 An Example of MRC

Figure 4.2 illustrates how MRC works on an input sequence 0,0,0,0,1,0,-1,3,-1. The input size is 9 bytes, while the output size is 3 bytes. So the compression rate is 3.

Basically the MRC algorithm is simple and works in linear time. Since it is specially designed for this resource limited and time critical scenario, it is better in compression speed and compression latency, comparing to other compression algorithms. Of course, it is not better in all aspects. It gets faster compression speed and low compression latency at the cost of lower compression rate. Table 4.4 shows the difference between Winzip, mp3, and MRC.[5], where *n* is the input size and *k* is the block size.

[5] Mp3 generator cannot generate a mp3 with a low data rate as 3.3 k or 5k. The compression rate is the average of 10 experiment results.

Properties	**Winzip**	**Mp3**	**MRC**
Time Cost	$O(kn)$	$O(kn)$	$O(n)$
Latency (byte)	32000	32000	1
Compression Rate	3.6	--	3.4

Table 4.4 Comparison of Winzip, Mp3, and MRC

Chapter 5

Implementation

5.1 Design Consideration

As we discussed before, to study sensor network's threat to personal privacy, we should show the feasibility of this system in the following aspects such as size, cost, scalability, longevity, and so on.

5.1.1 Size

To be used as an eavesdropping system, *SensorEar* should be small in size to avoid being noticed. In the future, with powerful sensor, which is as small as or even smaller than *Smart Dust* [4], we can easily reach this requirement.

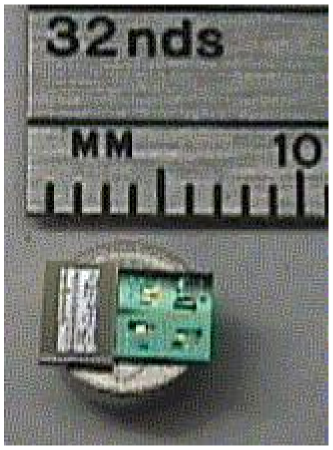


Figure 5.1 Appearance of Smart Dust

Mica2, the platform we use, is 58 x 32 x 7 (mm) big and weighs 18 grams, excluding the battery pack. As it is shown in Figure 2.1, Mica2 is small for most applications but not small enough to be unnoticeable. Anyway, it still shows the feasibility of building a real-time eavesdropping system with sensor network.

5.1.2 Cost

Budget is always important for a task. Currently, each MPR400CB, i.e. Mica2, costs $115.00. And 10 motes are in use with three of them equipped with sensor board MTS310CB another product of CrossBow Inc., which costs $185.00 each.

Part	**Price ($)**	**Qty**	**Cost ($)**
MPR400CB	115	10	1150
MTS310CB	185	3	555
Total	N/A	N/A	1705

Table 5.1 Hardware cost of a 3-hop *SensorEar*

$1705 is affordable to a malicious guy who is going to eavesdrop on others. And we all know, with the development of sensor manufacturing, there will be an obvious increase in sensor's performance as well as a drop in sensor price.

5.1.3 Scalability

The system is designed under the structure, which is very easy to be expanded. We can add new motes to extend the transmission distance without updating the existing nodes.

5.1.4 Life cycle

In normal sensor network applications, without frequent message transmissions, 2 AA batteries of Mica2 are enough to last up to several weeks.

In our system, if we let the system keep on recording continuously, the system will last for 22 hours.

In reality use, it may not be necessary to run the system continuously for such a long time since a speech won't last that long and on the other hand we can control the system remotely. Furthermore, we can improve the lifetime of the system by applying voice detection and power saving mechanism.

5.2 System Architecture

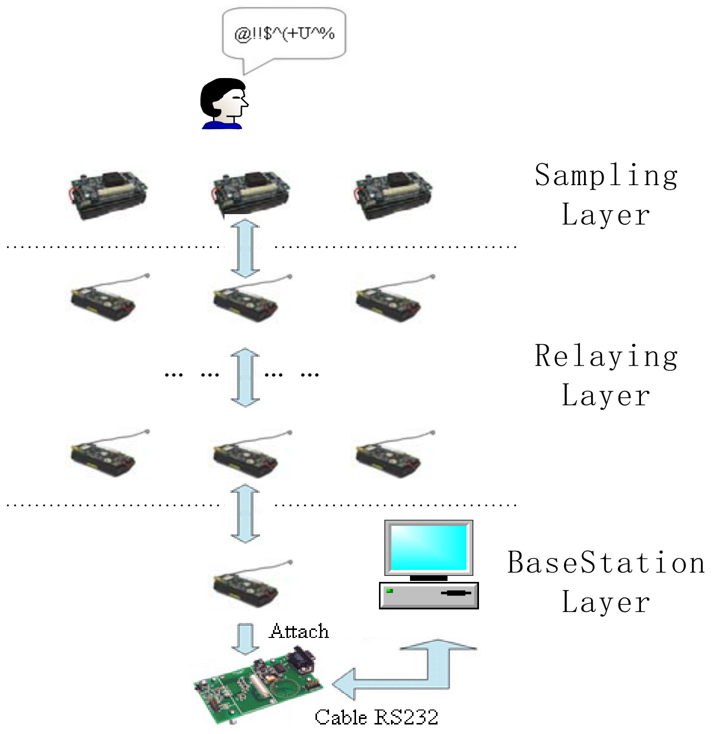


Figure 5.2 System's Architecture

First, let’s take a bird view at the whole system’s architecture, which is shown in Figure 5.2.

This system contains three layers: the sampling layer, the relaying layer and the base-station layer. On the base station layer, there is only one mote, which is connected to the computer through the Gateway board (MIB510) and a serial cable. We use the base station to send out commands and receive data messages.

There are three motes on sampling layer, each of which is attached with a basic sensor board (MTS310). With the microphone on this sensor board, the sampling mote can sample sounds and send the collected data back to the base station.

The relaying layer is the intermediate layer between sampling layer and base station layer. Motes on this layer are responsible for forwarding commands to the sampling layer and data messages to the base station layer. This layer is extendible. In this project, we use two sub-layers to show the whole system’s scalability. And each sub layer of the relaying layer also has three motes.

5.3 Demonstration of Control Flow

In this section, we will introduce the control flow of the procedure, and all details in designing..

5.3.1 Sending command

We developed a GUI program on PC side, which we can use to send out commands including parameters to the mote that acts as the base station through the serial port. On receiving the command, the base station will switch to the channel of its successor and send out the command. Motes on the relaying layer will also switch to the channel of their successor and send out the command after they receive the command. Then hop-by-hop, the command will be sent to the sampling layer.

After sending out the *SampleRequest* Command, the GUI program will listen to the serial port and wait for the data message.

The relaying motes which receive the command, will save it in the RAM, switch to the channel of the next layer and forward it. Hop by hop, the command will finally reach the sampling layer.

5.3.2 Sampling

On the sampling layer, mote 1 works as the coordinator of the layer. Only mote 1 will respond to the command from outer layer. It will start sampling according to the sampling rate and sample numbers specified by the *SampleRequest* command. Before it starts sampling, it turns off the radio to ensure the high rate sampling procedure will not be interrupted.

Because when the radio is on, the mote will detect the channel in every clock period by default. As we all know that the mote is a serial system, switching between different components will greatly decrease the throughput and efficiency.

We use the HFSSampling module, which is provided by TinyOS, to control the sampling procedure. But TinyOS's microphone control only provides a 1-bit sampling function. So we write a new microphone control module to get a 10-bit data at each sampling.[6]

Each sampled byte will be written to the flash memory for further processing. Function fastappend() of module BufferedLog provides efficient data logging. And during the sampling period of mote 1, mote 2 will stay in listening mode and wait for the *SampleRequest* command from mote 1. Mote 3 is the next one to sense after mote 2. When mote 3 finishes sampling, it will send the *SampleRequest* to mote 1 again to start a new round. In other words, these three motes work in turns.

5.3.3 Compressing

After finish sampling, mote 1 will send a *SampleRequest* command to mote 2. And mote 2 will start sampling immediately. But there is still a sampling gap between two motes. Its length is about 0.3~0.5 second by estimation. It is acceptable and ignorable comparing to the sampling period of each mote, which is 18 seconds in our experiments.[7]

After sending out the *SampleRequest* command, mote 1 will go back to its own job, i.e. compressing the data, which has been collected.

[6] The 10-bit sample data will be right shifted twice to be fit in one byte.
[7] The sampling number is set to 60000. So the sampling period is 18 seconds when the sampling rate is 3.3 k and 12 seconds when the sampling rate is 5k.

Every mote maintains two logs each with size 65535 bytes. One is used for original data storage and the other for compressed data. The Compress module reads data from the original data log, compresses them and then saves them to the compressed data log. The whole process of compressing 60000 samples takes about 4 seconds.

5.3.4 Transmitting

5.3.4.1 CSMA/CA

Transmission is the most vital part of this project, since throughput is the bottleneck of the whole system.

When the compression is finished, it's time to send out the data. According to the data sheet of Mica2, its transmission rate is 19.2 kbps. This is actually the upper bound of transmission rate.

Although many papers mentioned 19.2 kbps when talking about transmission rate, actually, in the experiment with the default MAC layer mechanism of TinyOS, the transmission rate never reaches this high. The transmission rate that we can achieve is far below this value, which is around 3 kbps. (We transferred 60000 bytes in about 162 seconds without any optimization.) In the following part, we will optimize the throughput in two ways: modifying the data length and the MAC layer protocol setting.

As it is shown in Figure 5.3, the default packet's length is 36 bytes (7 bytes of header information and 29 bytes of data) and the default preamble signal is 28 bytes. It means in every 64 bytes, only 29 bytes carry the data. The transmission efficiency is 45%, which is relatively low. Through increasing the packet size to 247 bytes, we get an efficiency of 87%, which is much higher. And the transmission rate is increased to 8 kbps, still far below the upper bound.

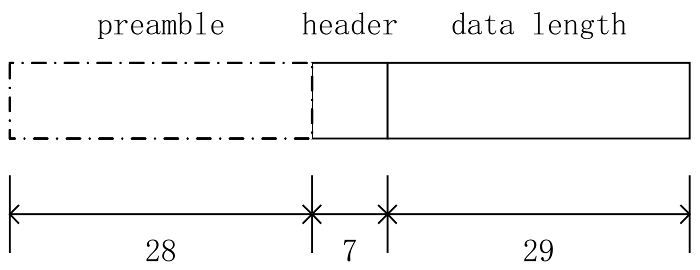


Figure 5.3 Default Packet Length

This results from the CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) protocol applied in TinyOS. In CSMA/CA, if the channel is sensed busy before transmission then the transmission is deferred for a "random" interval. This reduces the probability of collisions on the channel.

The default back-off delay is a sum of the packet length, which is 247 in our application, and a random number distributed evenly between 0 and 127. In other words, to avoid collision, the mote will wait for around 247~373 bytes time to send a 247-byte long packet. The efficiency is less than half. This expectation is consistent with the experiment result, which we mentioned before, i.e. when the data length is 240 and no modification on CSMA/CA the transmission rate is 8kbps.

Since in our system, all motes work in turn orderly, this CSMA mechanism is not indispensable. So we modified the setting of MAC protocol by reducing the back-off delay in the module *CC1000RadioIntM.nc*.

To figure out the most appropriate setting, we tried two experiments to get the results in Table 5.2 and Table 5.3. In the first experiment, we got Table 5.2 by setting the back-off delay to a random number between 1~32 and varying the data length of the packet size, which is 29 by default. Table 5.3 is the result of second experiment in which we set the data length to 240 bytes then adjusts the back-off delay to test the transmission rate.

Data Length (Bytes)	240	210	180	150	120	90	60	29 (default)
Transmission Rate (kbps)	14.8	14.4	13.8	13.2	12.2	11.1	9.1	5.9

Table 5.2 Data Length and Transmission Rate (back-off delay: 1~32)

Back-off Delay	1~32	33~64	65~96	97~128	129~160	161~192	193~224	225~256	247~374(default)
Transmission Rate (kbps)	14.8	13.4	12.2	11.3	10.4	9.7	9.1	8.5	8.1

Table 5.3 Back-off Delay and Transmission Rate (Data Length: 240 bytes)

From the experiments, we can see that either reducing the data length or increasing the back-off delay will reduce the transmission rate. How high should the transmission rate be enough to this project? If we set the sampling rate to 5kHz, each mote generates a 40 kbps data stream. At this rate, sampling 60000 bytes costs each sampling mote 12 seconds. It means each sampling mote has around 12 seconds to send out the compressed data. The compression rate of the MRC is at least 2. Then each sampling mote has at most 30000 bytes to send. To make sure that two sampling motes' transmission time will not overlap, we should ensure a transmission rate at least 20 kbps (30000*8/12 = 20 k). None of the above settings satisfies this requirement. But usually, the average compression rate is about 3.4 as it is shown in Table 4.2, and generally larger than 2.8, although no guarantee. Assume 2.8 is the minimum compression rate, then each sampling mote has at most 21429 bytes to send. And the requirement on transmission rate is lower down to 14.3. The first several settings barely satisfy the requirement. So we pick the current setting as data length set to 240 bytes and the random back-off delay range set to 1~32.[8] If we reduce the data length and increase the back-off delay, we are possible to reduce the collision rate. But we reduce the transmission rate at the same time, if a mote fails to transfer its data in its own time frame, there will be more severe packet loss. Plus, under the current setting, the packet loss rate is 0~0.4% in one hop transmission and 0~4% in three hop setting. So we can live with that.

5.3.4.2 Multihop and Channel Switching

At such a high data rate, the sender is almost on full load and not able to do anything else during transmission. And on the other side, the receiver takes similar time to receive a packet as for the sender to send a packet. So the receiver is also almost on full load.

Actually, in the experiment, the receiver on relaying layer is not able to forward a packet in time before it receives the next packet. Obviously, there's no way that the receiver can handle receiving and forwarding at the same time. By intuition, there are two solutions. The first one is saving all the received data and forwarding them later. And the second is using multi motes to share the burden of relaying.

[8] The maximum data length is 248 bytes, i.e. 255 – 7. And the minimum back-off delay can be 0. But it won't bring big improvement. So we just leave a little margin for the experiment.

Can we save these data in the RAM or flash memory? The answer is no. Because the size of the RAM is only 4 k bytes, which is much smaller than 60 k, the size of the data we collected. And the flash memory is slow in accessing speed. It will be irresponsible for about 3 milliseconds after each write, in which 16 bytes are stored at a time. So it is faster for the sender to read from the log and send the message than the receiver to receive the message and write to the log. It means the first solution does not work.

Obviously, multi relaying motes are required. That's why we use 3 motes on each sub-layer of the relaying layer. And for each mote in the relaying layer it only relays one packet in every three packets. So every relaying mote has plenty of time to send out the received packet before the next assigned packet arrived. On receiving a packet, it just sends it out and waiting for next expected packet, which is 3 packets' later than the current one.

So the relaying mote just either ignore or forward the packet it received. But there's still a new problem. When it tries to forward a packet, it can still hear the messages from the sampling layer. It means the relaying mote will always find the channel is busy when it tries to send out that packet, then it has to back-off for a random interval. Then here is collision again.

Since so many motes are sending packets without synchronization, collision becomes the biggest problem. Collision results in packet lost. It can be even more serious if collision happens when a *SampleRequest* command is in transmission. Failure in receiving

SampleRequest command will result in halt of the whole system. For example, if mote 1 is still sending out data messages when mote 2 sends out the *SampleRequest* command, then mote 3 will miss the command and still stay in waiting mode. Ergo no motes will continue sampling until a *SampleRequest* command comes from outer layers. The whole sampling procedure will stop here.

To avoid collision, we also tried handshaking or resending mechanism but all failed. Because the system is working on the edge of its limitation, making the mechanism more complicate will decrease the throughput and result in data jamming.

We solve this collision problem by designing a multi channel network model. We use multi channels in the whole system and each layer has its own channel. Now the sampling layer get its own channel and that guarantees the channel is vacant when a sampling mote tries to sends out the *SampleRequest* command to the next mote.

Before sending, the sender will switch to the receiver's channel before sending out packets. For example, when a relaying mote, say Relayer_A, is about to send out the packet it received, it will switch to the channel of the next layer, which is closer to the root, and send the message out. During this time, the mote in the sampling layer is still sending out packets in a different channel. So it will not affect Relayer_A in forwarding. After finishing forwarding the packet, Relayer_A will switch back to its original channel and listen to new messages.

In other words, in communication between different layers we apply FDMA (Frequency Division Multi Access) and in communication among motes of the same layer we apply TDMA (Time Division Multi Access) to solve conflict and keep all the motes working in order.

On the base station layer, there is only one base station. Because the mote acts as base station receives through radio channel and sends through the serial port. The transmission through serial port is much faster than the radio channel. It ensures the mote to send out the packet to the computer in time.

In all, there are 3 layers in the whole system and among which the relaying layer has two sub layers. Each of them has their own channel. According to the datasheet of Mica2 [13], Mica2 has up to 50 channels between 868 MHz and 916 MHz. It implies that the average distance between adjacent channels is 960 kHz.

Actually, the channels or available frequencies are more than 50. The number 50 provided by the datasheet is just an approximate number of available channels, which provide sufficient gaps between each other to ensure reliable transmission immune to adjacent channel interference.

The interface *CC1000Control* provides an access to adjust the central frequency of the mote. The frequencies are synthesized from the basic frequencies of the oscillator. So the frequency cannot be arbitrary and only some specific frequencies can be generated. While setting the central frequency, we should provide a value as the parameter of the function. We may not necessarily know the exact value of the eligible frequency. We can provide an arbitrary number in the range, and the synthesizer will finally generate the available frequency which is the nearest to the specified value.

But the synthesizer's performance varies in its working range. In other words, on some frequencies it works less steadily than on others. When the frequency drift increases to some extent, it results in packet loss. Then we conduct the Channel Test experiments to check which channels are appropriate for reliable high data rate transmission.

The test is simple. We make the mote keep on sending out packets at its maximum speed in the specified channel and count the number of lost packets. Figure 5.4 shows the relation between packet loss rate and channel ID. The channel ID and their frequency values are shown in Table 5.4.

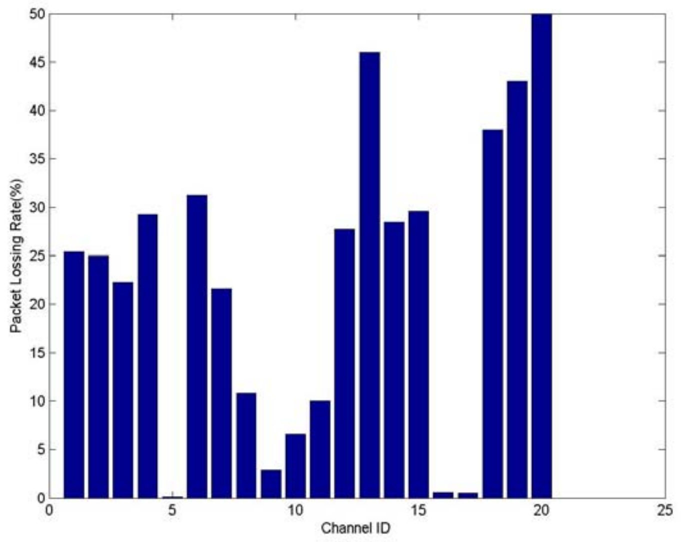


Figure 5.4 Channel Test on Packet Loss Rate

ChannelID/ Frequency(MHz)				
1/906.704	2/907.838	3/908.715	4/909.653	5/910.391
6/911.396	7/912.234	8/913.156	9/914.077	10/914.998
11/915.920	12/916.758	13/917.763	14/918.614	15/919.439
16/920.397	17/921.450	18/922.187	19/923.088	20/924.083

Table 5.4 Channel ID and Channel Frequency

From Figure 5.4 and Table 5.4, we can find that Channel 9 and 10 are two default channels of Mica2. They are not best channels. And Channel 5,16, and 17 are obviously better than any others. So we pick 4 channels for this system, Channel 5, 16, 17 and Channel 9. Channel 9 is assigned to the sampling layer and the other three are assigned to other layers. If more hops are needed, these channels can be reused as long as the layers using same frequency are far enough from each other, just like SDMA (Spatial Division Multi Access) in mobile communication.

Channel 9's performance is a little worse than others but it should be fine. Because the sampling layer only receives command messages, it means there's no high data rate transmission in this channel.

5.3.5 Start a new round

After finish sending all the data, mote 1 will switch back to waiting mode. Once it receives the *SampleRequest* command from mote 3, it will start a new round by jumping to sampling status. Then a new round starts.

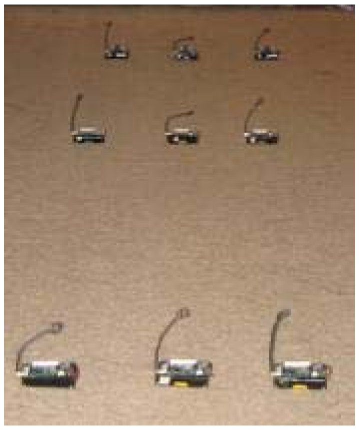


Figure 5.5 Scene of the Experiment

5.4 System Modules

TinyOS is an event driven system. An application is composed of components and these components communicate with each other through interface. There are two kinds of components in nesC: configuration and module. Configuration is like a setting component, which wires other modules together. And module is the component that provides specific functionalities.

As we mentioned before, motes on different layers run different modules. Corresponding to three layers, there are three major modules: MyHFS, MyRelay, MyTOSBase. In the following part of this section, we will introduce the modules in this application.

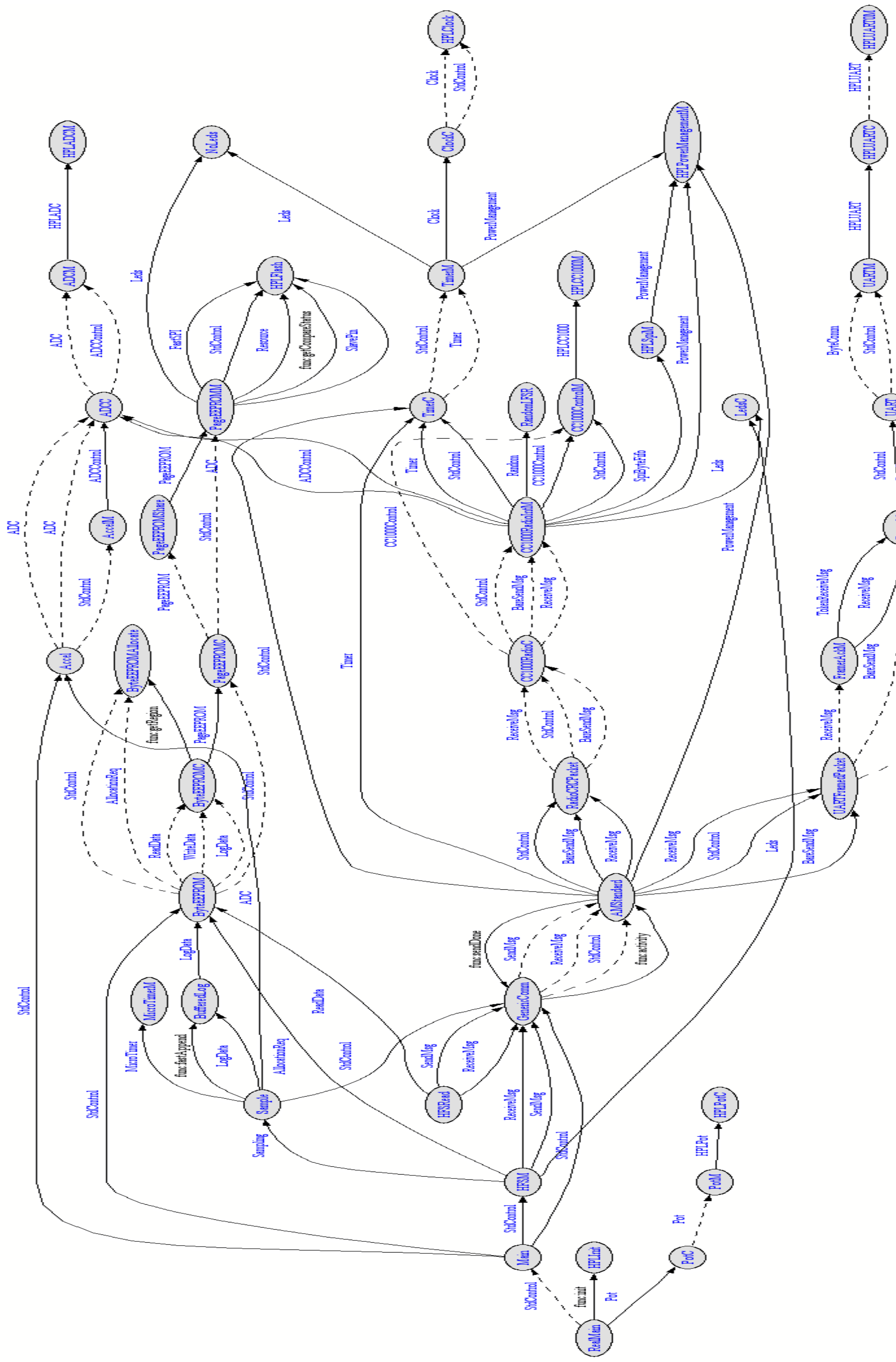


Figure 5.6 Generated Component Structure Graph of MyHFS

5.4.1 MyHFS

Based on the provided application HighFrequencySampling, we develop our own high frequency sampling application, which features in the new designed microphone-controlling module, multi-rate compression algorithm. This application is installed in the motes on the sampling layer. It is composed of the following components: HFSM, GenericComm, MicroTimer, MyMic, BufferedLog, Sample, ADC, Compress, HFSRead, etc. The system generated component structure graph is shown in Figure 5.6.

It's very complicated. So we will introduce them one by one through illustration with brief and summarized graph. In these summarized graphs, a box means a module and a diamond means an event. Arrows mean control flow.

5.4.1.1 Sample

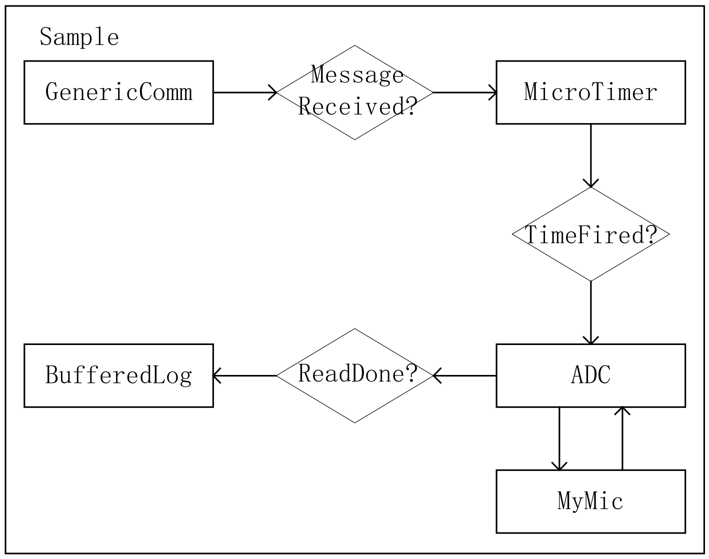


Figure 5.7 Flow Chart of Module Sample

Sample is a self-designed module that controls the relevant modules to get sample from the outer environment and save to the log. The following is the flow chart of Sampling

5.4.1.1.1 GenericComm

This module is a communication module provided by TinyOS to send or receive messages through radio or serial.

5.4.1.1.2 MicroTimer

This module is a microseconds interval timer provided by TinyOS. The common Timer module's minimum interval is 1 millisecond, which is far below the requirement of high sampling rate. The MicroTimer only supports repeat timer mode. And it cannot be used together with radio. So we turn off radio before sampling and then start the MicroTimer. When the sampling is finished, we will stop the MicroTimer and turn on the radio.

The default minimum interval of MicroTimer is 283 microseconds, which is corresponding to a maximum sampling rate of 3.53 kHz. To increase the maximum sampling rate, we double the ADC clock's frequency at the cost of lower ADC precision, then we get a maximum sampling rate of 7 kHz, i.e. the minimum interval of MicroTimer is set to 142 microseconds.

In our experiment, we usually set the interval to 300 microseconds or 200 microseconds, corresponding to sampling rate 3.3 kHz or 5 kHz.

5.4.1.1.3 ADC

ADC is a system provided module. It provides access to control the ADC component and get the sampled data. At each sampling call, ADC will return a 10-bit value.

It provides two interfaces: *ADC*, *ADCControl*. Interface *ADC* is used to wire the specific sensors such as photo sensor or microphone, etc. In this project, it is wired to the MyMic module. Interface *ADCControl* is the access to modify the settings of the sensor, such as microphone gain, clock frequency, etc.

5.4.1.1.4 MyMic

This module is a modified version of module Mic. The sampling function of the original module Mic only returns a 1-bit value that is used to detect exist of sound. That’s why we have to develop our own Mic module.

MyMic is modified from Mic. The difference is that it can return a 10-bit value on each sample. To save communication overhead, we only use the most dominant 8-bit of the sample by right shifting the sample twice. The module also provides function gainAdjust() for gain adjustment.

5.4.1.1.5 BufferLog

BufferLog is a system provided module that support high-speed logging. This component uses two buffers. While one buffer is filled by application, the other is written to EEPROM. The size of each buffer is 128 bytes

5.4.1.2 Compress

Once the specified amount of data is collected, a SampleRequest command will be sent out to the next mote through the GenericComm module. After sending the command, the Compress module starts to work.

We allocate two logs in the EEPROM for the storage of original data and compressed data. Compress module reads the original data from the first log, compresses them and saves them to the second log.

We design a data compression algorithm for this module, called Multi-Rate Compression (MRC). It is specifically designed based on the feature of the sample of this application.

5.4.1.3 HFSRead

This module is also a modified version of the original *HFSRead*. In the previous application, *HFSRead* is used when the mote hear the ReadRequest.

In this application, it works in a different way. The motes will send data automatically right after they finish compressing. And *HFSRead* doesn’t read from the log that stores the original sample, instead it reads from the log of compressed data.

5.4.1.4 HFSM

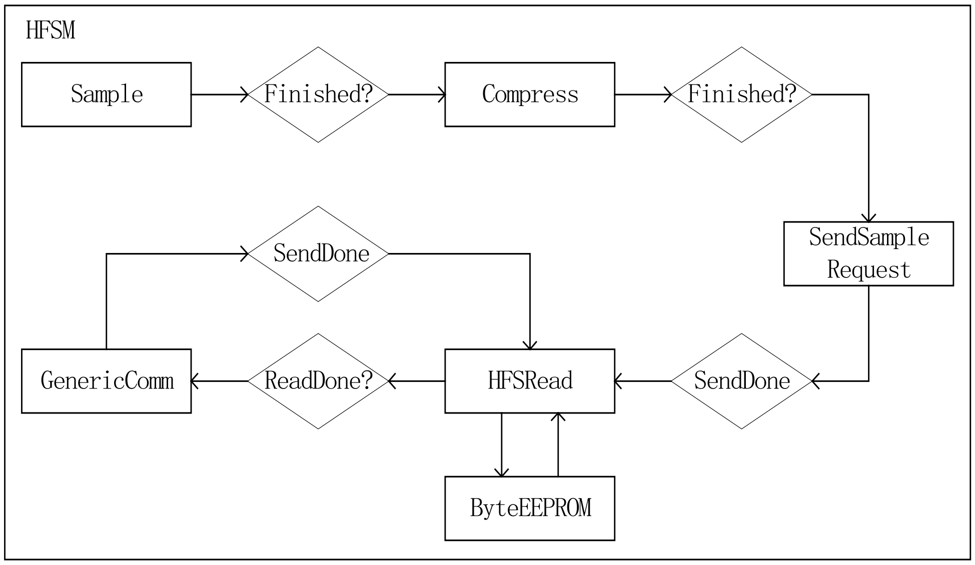


Figure 5.8 Flow Chart of HFS

This module is the platform of the whole sampling application. It wires all modules together and take care of the intermediate operations such as memory allocation, event triggering, channel switching and so on.

5.4.2 MyRelay

MyRelay is the application on the relaying layer. This application is responsible for message relaying. It receives messages from successor layer, switch to the predecessor layer's channel and forwards messages to the predecessor layer, and vice versa.[9] The platform module is *MyRelayM*. It wires the components together. Besides the *GenericComm* component, we also use the *CC1000ControlM* component, a radio control module.

The component *CC1000ControlM* is a system module through which we can modify the

[9] We define the base station as the root. So the predecessor layer means the neighbor layer which is closer to root.

radio settings. In this project, we use it as the access to switch the radio channels.

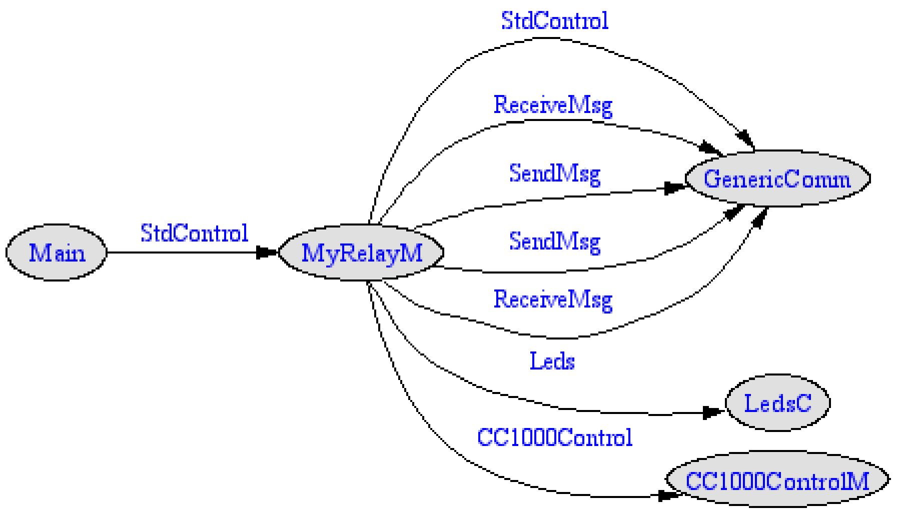


Figure 5.9 Generated Structure Graph of MyRelay

5.4.3 MyTOSBase

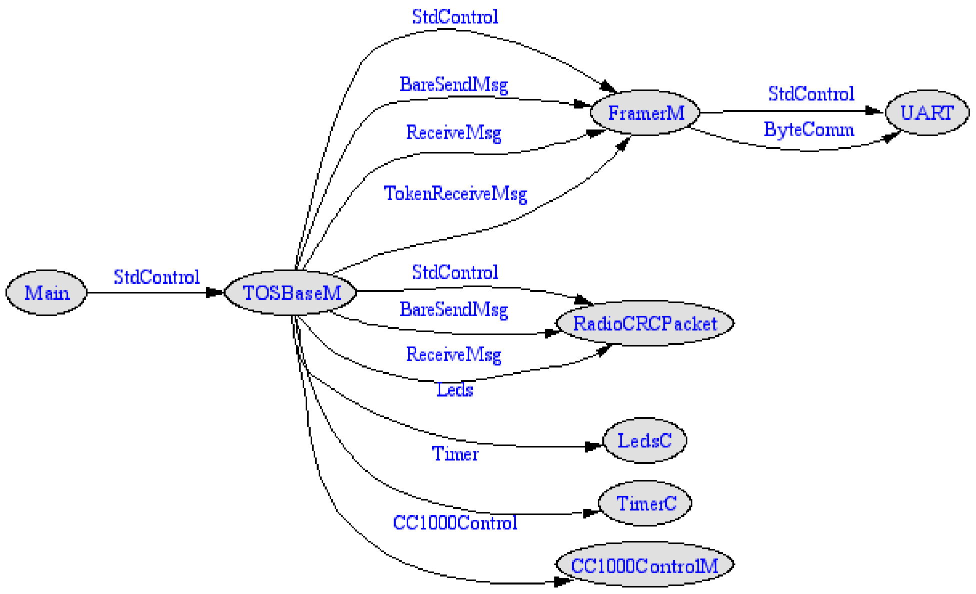
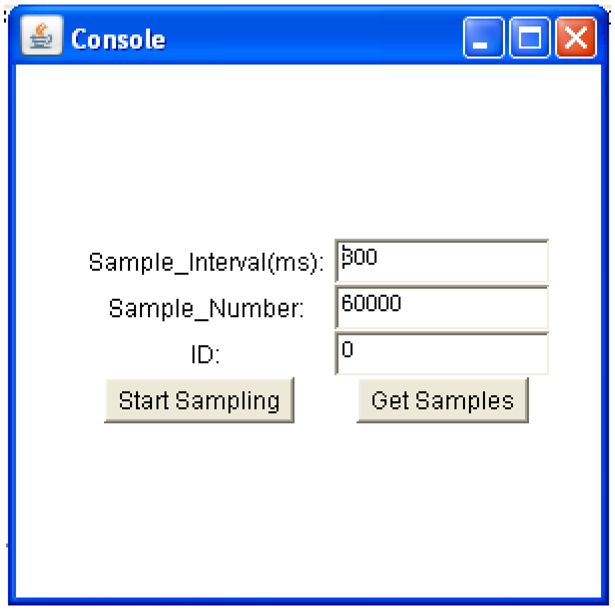


Figure 5.10 Generated Structure Graph of MyTOSBase

MyTOSBase is modified from the system provided application TOSBase. The only difference is the channel switching mechanism, which is also one of the features of this project.

On receiving message from the computer, MyTOSBase will switch to the channel of its nearest relaying layer before forwarding it. After it sends the command out, it will switch back and stay in its pre-specified channel waiting for data packets from the relaying layer. Other motes trying to send message to the base station should also switch to the channel of base station before sending.

5.4.4 Console



Sample_Interval(ms): 300
Sample_Number: 60000
ID: 0

Figure 5.11 Console-The GUI on PC Side

We also provide Console, which is a Graphic User Interface on the computer side for command injecting and data receiving. As it is shown in Figure 5.11, the sampling parameters are set through the input field. When the “Start Sampling” button is clicked[10], these parameters will be encapsulated into a command packet *SampleRequest* and send out to the base station. Through the relaying nodes’ forwarding, the packet will finally arrive at the sampling layer.

After send out the command, the program will open a socket and listen to packets, which are received by the base station and forwarded through serial port. Then the GUI will decapsulate the packets and save the data to a pre-specified file. The file will be uncompressed later and converted into a .wav file.

[10] The “Get Samples” button is used to retrieve the data from the mote in the previous version. But in the new version of this application, it is not in use, because the mote will send data automatically.

Chapter 6

Performance Evaluation

To evaluate the quality of this speech recording system, we use the PESQ (Perceptual Evaluation of Speech Quality) [21] to measure the similarity between the original speech and the recorded audio. We will analyze the impact on recording quality from different aspects.

6.1 Compression and Noise Reduction

Compression and noise are two sources of sound quality degradation. First of all the compression algorithm is a lossy algorithm. We achieve higher compression rate at a cost of sound quality degradation. All samples that larger than 3 or smaller than –3 will be quantified to 3 or –3. This brings in extra distortion, as it is shown in Figure 6.1. Fortunately, the reduction in quality is tolerable and not drastic.

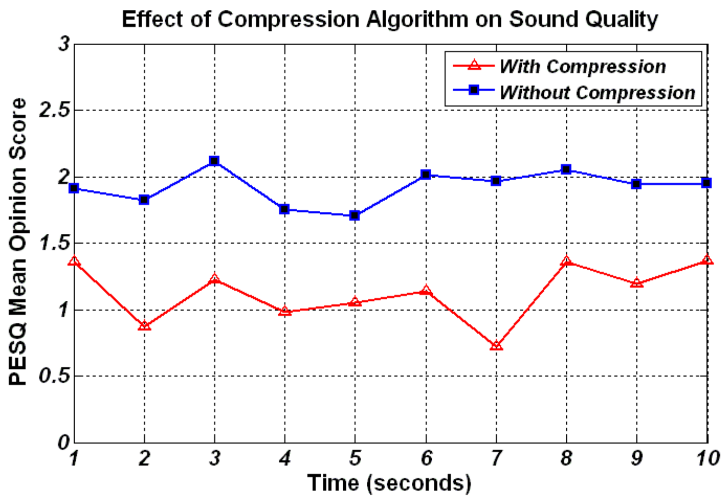


Figure 6.1 Effect of Compression Algorithm on PESQ

With the noise reduction algorithm Ephraim-Malah MMSE [22], which is a widely used background noise reduction algorithm, the degradation caused by the compression algorithm will be reduced a little bit, as it is shown in Figure 6.2.

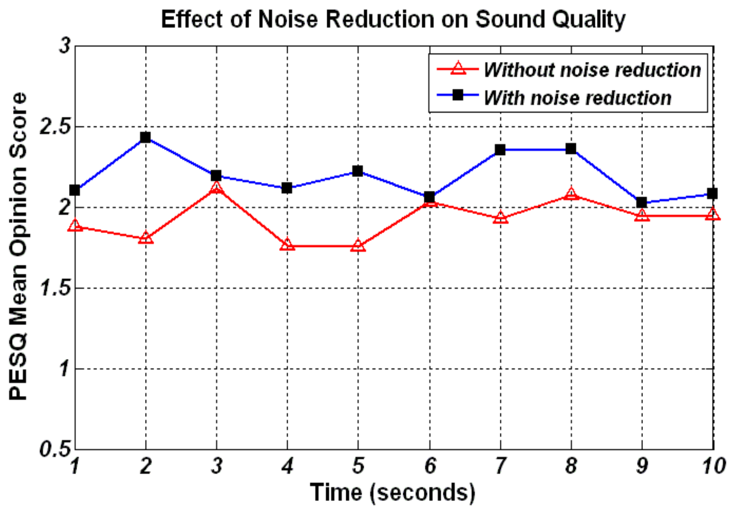


Figure 6.2 Effect of Noise Reduction Algorithm on PESQ

6.2 Distance

From Figure 6.3, we can see that the quality drops as the distance increases. It is normal because when distance increases the signal strength of sound will drop and it makes the effect of background noise considerable. That's why the noise reduction algorithm is more helpful in longer distance.

On the other hand, when the distance is too close, the sound will be "louder" to the microphone. Our compression algorithm clips samples larger than 3. So when the mote is nearer to the source, more samples will be clipped, which introduce extra distortion.

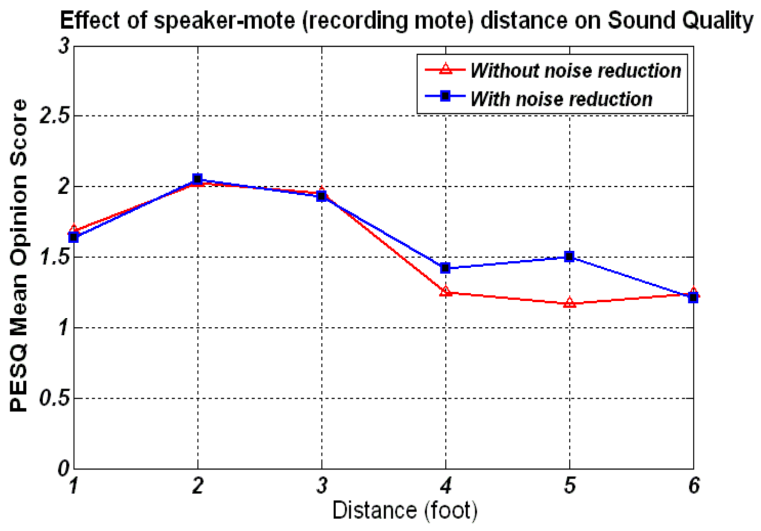


Figure 6.3 Effect of Distance on PESQ

6.3 Multi-hop and Channel Switching

Since we use multi-channel-multi-hop scheme, the packet loss rate increase correspondingly. Besides collision, channel switching brings in more possibility in packet loss. If the mote fails to work in the specified channel steadily, it will fail in sending or receiving. For each mote, the failure rate is less than 0.5 %. There are 10 motes in the system. So the whole packet loss rate of the 3-hop system should be less than 5%, which approximately matches our experimental result. Actually, sometimes the packet loss rate can be as big as 30%, when the lines of sight between motes are blocked. It's because the transmission power is too weak. In Figure 6.4, we can see multi-hop brings a little degradation on PESQ as expectation.

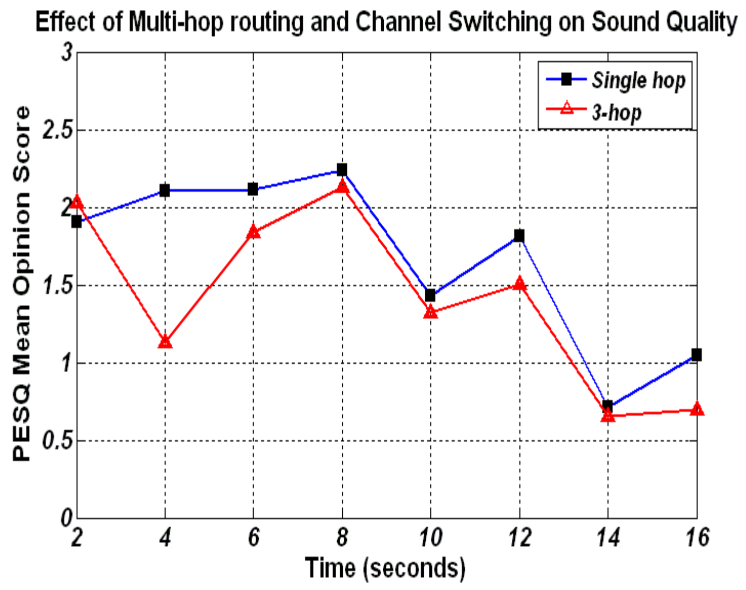


Figure 6.4 Effects of Multi-hop and Channel Switching on PESQ

Chapter 7

Discussion

7.1 Bottlenecks

In the previous chapters, we have shown the feasibility of building a sensor network based eavesdropping system. But the quality of the recorded sound is still limited by several aspects.

The first factor that limits the sound quality is the insensibility of the microphone. The microphone is very weak. If the original sound is not loud enough, the recording quality will be pretty poor. In other words, the recording quality is sensitive to distance, as it is shown in Figure 6.3.

The second reason that can greatly cause quality degradation is packet loss. Collision is one common reason that results in packet loss. Because of the relatively high data rate, we are not able to leave enough margins in design. In other words, the motes are working on the edge of their limitation. And there's no room for reliable transmission protocols. So after all the low transmission rate of Mica2 is the biggest bottleneck of this experiment.

On the other hand, failure to detect a message is another reason that causes packet loss. Like most of the sensor network applications, all the experiments of our project are proceeded with all the motes in line of sight of each other. If anything that block between the sender and the receiver, it will greatly weaken the strength of signal detected by the receiver and consequently result in packet loss. When we deployed these motes in two rooms separated with a wood wall, the packet-losing rate increase dramatically from 4% to more than 30%. That’s another inherent weakness of Mica2.

7.2 Possible Solutions

After study the feasibility of sensor network based eavesdropping, the next step is to figure out how we can improve the quality or what other fancy things we can do with better sensors.

Comparing to Mica2, MicaZ’s 250kbps transmission rate is a large boost. It’s 12 times faster than Mica2. With such a high transmission rate, we get much more room in improving. First, we can provide better quality data by adjusting the sampling function and the compression algorithm. Second, we can add reliable transmission protocols. Third, we can try video recording.

Chapter 8

Conclusion

Through this experiment, we prove the feasibility of using multi sensors to build a continuous real-time eavesdropping network. Self designed multi-rate compression algorithm and multi channel network model enable us to get a data rate higher than the limitation of Mica2. These algorithm and prototype designs can also be used in other large data volume applications.

In the end, we also discuss the bottleneck of the current project and possible future works. With the development of sensor technology, sensors will become smaller, cheaper and more powerful, which makes personal sensor network more common. And it will enable implementation of more fancy applications.

Reference

[1] Gyula Simon et al. “Sensor Network-Based Countersniper System”, *SenSys’04, November 3-5, 2004*, Baltimore, Maryland, USA

[2] Alan Mainwaring. et. al. “Wireless Sensor Networks for Habitat Monitoring”, *WSNA’02, September 28, 2002*, Atalanta, Georgia, USA

[3] Victor Shnayder et al. “Sensor Networks for Medical Care”, *Technical Report TR-08-05*, Division of Engineering and Applied Sciences, Harvard University, 2005

[4] Kris Pister. et al. “Smart dust: Autonomous sensing and communication in a cubic millimeter”, *http://robotics.eecs.berkeley.edu/~pister/SmartDust/*

[5] Jun Han, et. al., “Don’t Sweat Your Privacy”, *Proceedings of 5th International Workshop on Privacy in UbiComp (UbiPriv'07)*, September, 2007

[6] Liqian Luo,et. al., “EnviroMic: Towards Cooperative Storage and Retrieval in Audio Sensor Networks”, *27th International Conference on Distributed Computing Systems (ICDCS '07)* [7] Jacob Ziv, Abraham Lempel, “A Universal Algorithm for Sequential Data Compression”, *IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. IT-23, NO. 3,* MAY 1977

[8] D.A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes", *Proceedings of the I.R.E.*, September 1952, pp 1098-1102

[9] Mayer, Alfred Marshall (1894). "Researches in Acoustics". *London, Edinburgh and Dublin Philosophical Magazine* **37**: 259–288

[10] J. P. Princen and A. W. Johnson and A. B. Bradley, "Subband/transform coding using filter bank designs based on time domain aliasing cancellation," *IEEE Proc. Intl. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)* **12**, 2161-2164 (1987) (Initial description of what is now called the MDCT.)

[11] Steven W. Smith, Chapter 27 “Data Compression” of Book “The Scientist and Engineer's Guide to Digital Signal Processing”

[12] Zhigang Cao, et al, Chapter 5 “Pulse Coding Modulation” of Book “Modern Communication Principals”

[13] Crossbow Inc, Data sheet of Mica2, http://www.xbow.com/products/Product_pdf_files/Wireless_pdf/MICA2_Datasheet.pdf

[14] ATMEL Co., Data sheet of Atmel128L, http://www.atmel.com/atmel/acrobat/doc2467.pdf

[15] K. Whitehouse, "The Design of Calamari: an Ad-hoc Localization System for Sensor Networks," University of California at Berkeley, 2002.

[16] J. Jeong, S. Kim, and A. Broad, “Network Reprogramming”, 2003. http://webs.cs.berkeley.edu/tos/tinyos-1.x/doc/NetworkReprogramming.pdf,

[17] TinyOS Forum, http://www.tinyos.net/

[18] “Sourceforge.net: Project Info - TinyOS,” http://sourceforge.net/projects/tinyos/

[19] Houghton Mifflin Company, “The American Heritage Dictionary of the English Language”, Fourth Edition,2006

[20] Wikipedia, Audio Freqency, http://en.wikipedia.org/wiki/Audio_frequency

[21] ITU, PESQ, http://www.itu.int/rec/T-REC-P.862/en

[22] Ephraim, Malah, "Speech Enhancement Using MMSE Short-Time Spectral Amplitude Estimator", IEEETrans.on ASSP, vol.32, N6, Dec.1984.