Wireless ECG System with Bluetooth Low Energy and Compressed Sensing

by

Wanbo Li
B.Sc., Beijing University of Posts and Telecommunications, 2012

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF APPLIED SCIENCE

in the Department of Electrical and Computer Engineering

Wireless ECG System with Bluetooth Low Energy and Compressed Sensing

by

Wanbo Li
B.Sc., Beijing University of Posts and Telecommunications, 2012

Supervisory Committee

Dr. Xiaodai Dong, Supervisor
(Department of Electrical and Computer Engineering)

Dr. T. Aaron Gulliver, Department Member
(Department of Electrical and Computer Engineering)

Supervisory Committee

Dr. Xiaodai Dong, Supervisor
(Department of Electrical and Computer Engineering)

Dr. T. Aaron Gulliver, Department Member
(Department of Electrical and Computer Engineering)

ABSTRACT

Electrocardiogram (ECG) is a noninvasive technology widely used in health care systems for diagnosis of heart diseases, and a wearable ECG sensor with long-term monitoring is necessary for real-time heart disease detection. However, the conventional ECG is restricted considering the physical size and power consumption of the system. In this thesis, we propose a Wireless ECG System with Bluetooth Low Energy (BLE) and Compressed Sensing (CS).

The proposed Wireless ECG System includes an ECG sensor board based on a BLE chip, an Android application and a web service with a database. The ECG signal is first collected by the ECG Sensor Board and then transmitted to the Android application through BLE protocol. At last, the ECG signal is uploaded to the cloud database from the Android app. We also introduce Compressed Sensing into our system with a novel sparse sensing matrix, data compression and a modified Compressive Sampling Matching Pursuit (CoSaMP) reconstruction algorithm. Experiment results show that the amount of data transmitted is reduced by about 57% compared to not using Compressed Sensing, and reconstruction time is 64% less than using Orthogonal Matching Pursuit (OMP) or Iterative Re-weighted Least Squares (IRLS) algorithm.

Contents

List of Tables

List of Figures

GLOSSARY

BLE Bluetooth Low Energy

CoSaMP Compressive Sampling Matching Pursuit

CR Compression Ratio

CS Compressed Sensing

ECG Electrocardiogram

GATT Generic Attribute Profile

PRD Percentage Root-mean-square Difference

TI Texas Instruments

WiFi Wireless Fidelity

ACKNOWLEDGEMENTS

I would like to thank:

My supervisor Prof. Dong, for your mentoring, encouragement and patience, and also funding me with Research Assistantship.

My families, for your support and love.

My colleagues and friends, for your inspiration and help.

DEDICATION

To my family and friends.

Chapter 1

Introduction

Cardiovascular disease is a class of diseases involving the heart, the blood vessels or both [1]. Since the 1970s, cardiovascular disease has become the major cause of death among the whole world. According to American Heart Association [2], more than 787,000 people in the U.S. died from heart disease, stroke and other cardiovascular diseases in 2010, which is about one of every three deaths in America. Despite that the group of older adults is the group of people who are usually affected by cardiovascular disease, the antecedents of cardiovascular disease, especially atherosclerosis, start in early life, which makes primary prevention efforts essential from young people [3]. In order to assess and discover cardiovascular diseases early, electrocardiogram (ECG) has been used widely in the medical field.

ECG is used to measure and record the electrical activity and conduction of the heart [4]. In general, ECGs are performed for diagnostic or research purposes on human hearts. It picks up electrical impulses which are generated by the polarization and depolarization of cardiac tissue, and then translates them into an ECG waveform. By using ECG, some general symptoms can be indicated, such as symptoms of myocardial infarction and pulmonary embolism. In addition to this, ECG has also been used to assess patients who have systemic disease, and to monitor patients during anaesthesia [5].

Some heart diseases, like acute coronary and cerebrovascular, often happen incidentally and suddenly. This will bring a big chance of endangering patients' lives, and also a lot of stress and mental pain to them. Therefore, it is particularly important that the cardiovascular anomaly is diagnosed early and monitored. ECG monitoring is a good way to forecast heart disease and keep cardiac patients under tight watching of their heart conditions [6]. Moreover, as many heart problems are noticeable

only during activity, such like eating, stress, exercise, or even sleeping [7], a wearable, long-term ECG monitor is a practical path to achieve this goal.

1.1 Motivation and Related Work

The traditional ECG test machines used in hospitals are powerful and professional in detecting various heart diseases. However, those machines are almost unmovable and not suitable for personal daily monitoring. Besides conventional ECG devices, ambulatory electrocardiography has also been introduced as a diagnostic tool [8] to deal with those health crises.

Ambulatory electrocardiography is also named Holter monitoring, ambulatory ECG or ambulatory EKG. The main purpose of Ambulatory ECG is to describe and document abnormal electrical heart activity, which can be spontaneous, random, sleep-related or caused by emotion or stress [9]. As the capture and association of symptoms with destabilization in rhythm during daily life need a whole day's electrical activity record of heart [10,11], the monitoring must be continuous as the patient goes about normal daily activities [12].

For Holter monitoring, the patient wears a Holter monitor, i.e. a small recorder, in normal daily life. There are two basic types of recording devices: intermittent and continuous. Intermittent recorders [13] provide brief, intermittent recordings and can work for weeks or even months. While continuous recorders [14] are able to monitor about 100,000 heartbeats in 24 hours, and are more likely to find any heart issues that happen with activity.

However, both types of Holter monitors save the ECG data locally in the device which is usually only accessible to the doctor. Moreover, the Holter monitor cannot analyse the ECG signal, meaning that it cannot show the current heart situation or respond to sudden accident. In addition, the working length of the continuous type is only 24 hours. Also, the size of Holter ECG is still large with backpack pattern, and its convenience is still limited.

In recent years, wireless communication has been introduced into the ECG monitor system. It has brought plenty of convenience to the patients. Due to the usage of wireless communication technology, the size of ECG monitors can be designed very small, and the weight can be very light which is more comfortable for long-term wearing. The ECG data may also be sent wirelessly to and displayed in a separated device, for example a recording apparatus or a cellphone [15]. A variety of

wireless technologies have been applied to ECG monitoring, such as Zigbee [16, 17] and Bluetooth [15, 18]. However, considering the need of long-term and portable monitoring, low power and miniaturization are big challenges.

1.1.1 Bluetooth Low Energy

Bluetooth is a short-range wireless technology for exchanging data between fixed and mobile devices. It was originally invented to replace cables by telecommunication company Ericsson in 1994. It is able to get several devices connected overcoming synchronization problems. Bluetooth is managed by the Bluetooth Special Interest Group (SIG) whose member companies are in the areas of telecommunication, networking, etc. The IEEE (Institute of Electrical and Electronics Engineers) standardized Bluetooth as IEEE 802.15.1, but this standard is no longer maintained now.

Bluetooth operates in the 2.4GHz ISM (Industrial Scientific Medical) band. Each Bluetooth device has a uniquely defined address which can be used for identification between different devices. Asynchronous data channel is supported by Bluetooth, and frequency hopping scheme is also used to increase reliability. Compared to other wireless technologies, Bluetooth has a lower power consumption and is supported by more mobile devices. The huge number of devices using Bluetooth also lower down the Bluetooth chip price.

There are multiple versions of Bluetooth, and all the newer Bluetooth versions are backward-compatible. In 2002, Bluetooth V1.0 was adopted by the Bluetooth SIG. Afterwards, Bluetooth V2.0 with EDR (Enhanced Data Rate) was released in 2004, and the transmission rate was increased from 721 kbits/s to 2.1 Mbits/s. Bluetooth V3.0 + HS (High Speed) [19] was adopted in 2009, and it provides theoretical data transfer speed of up to 24 Mbits/s. In June 2010, Bluetooth V4.0 (Bluetooth Smart) was completed and adopted. It includes Classic Bluetooth, Bluetooth High Speed and Bluetooth Low Energy (BLE) protocols.

Bluetooth Low Energy was originally introduced by Nokia under the name Wibree in 2006. It is an alternative to the existing Bluetooth standard protocols, i.e., Bluetooth V1.0 to V3.0, and it is aimed at low-power applications running off small batteries such as coin battery. Two types of implementation of chip designs are allowed: dual mode and single mode. For the single-mode implementation, only the BLE protocol is implemented, while for the dual-mode, Bluetooth Smart functionality is integrated into the existing Classic Bluetooth controller.

Different than Classic Bluetooth capabilities, Bluetooth Low Energy is a connectionless protocol [20]. This significantly reduces the amount of time the radio must be on, and BLE devices are only awake when sending or receiving data. Therefore, Bluetooth Low Energy is designed for sending small chunks of data, and it does not support streaming [21]. Internet of Things (IoT) applications can benefit from this technology in situations when sensors need to report their current state. Bluetooth Low Energy is also suitable for health care and sports applications.

The topology of Bluetooth Low Energy is a star which is similar to Classic Bluetooth. BLE devices can operate as two separate roles [22]:

- **Central** (**master**): Repeatedly scans the preset frequencies for connectable advertising packets and, when suitable, initiates a connection. Once the connection is established, the central manages the timing and initiates the periodical data exchanges.
- **Peripheral** (**slave**): Sends connectable advertising packets periodically and accepts incoming connections. Once in an active connection, the peripheral follows the master's timing and exchanges data regularly with it.

To initiate a connection, a central device takes the connectable advertising packets sent from a peripheral. Then it sends a request back to the peripheral to establish an exclusive connection between the two devices. Once the connection is established, the peripheral stops advertising and the two devices can exchange data in both directions, as shown in Figure 1.1.

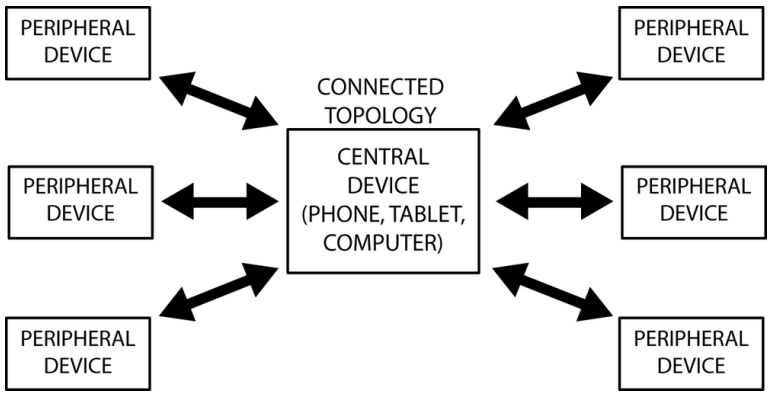

Figure 1.1: Bluetooth Low Energy connected topology

During the process of establishing the connection between master and slave, there are in total five states, including one 'Standby' state and four active states [21], as

given in Figure 1.2. Both the master and slave devices can go into deep sleep between transactions, and the power consumption is kept at a very low level until the next connection. The connection states for the peripheral (slave) and the central (master) are shown in Figure 1.3 and Figure 1.4, respectively.

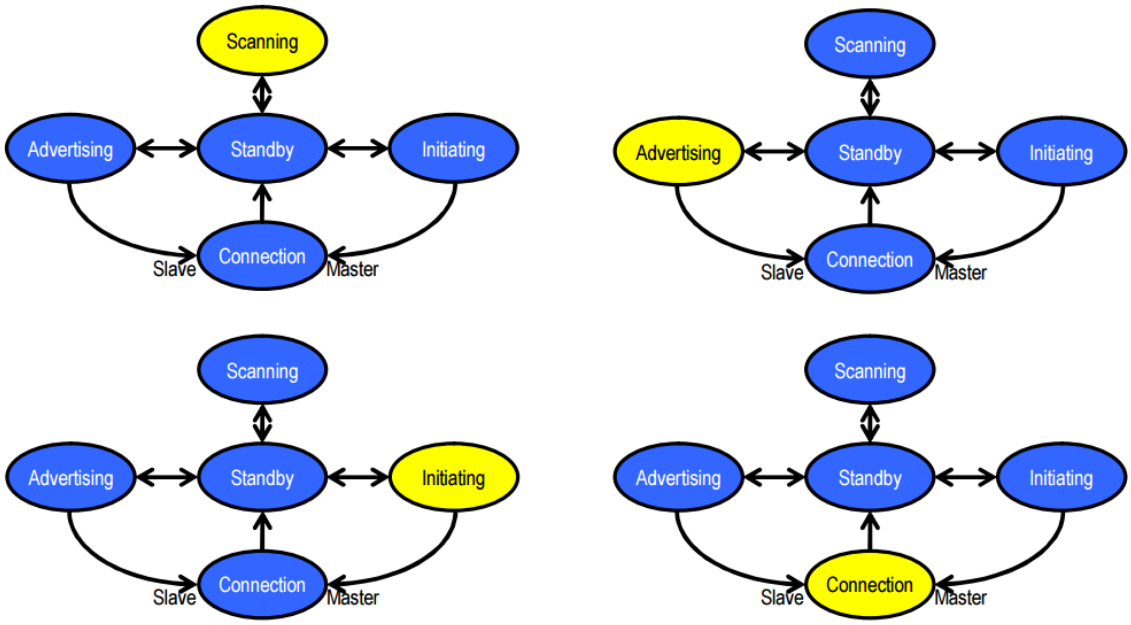

Figure 1.2: Bluetooth Low Energy connection states

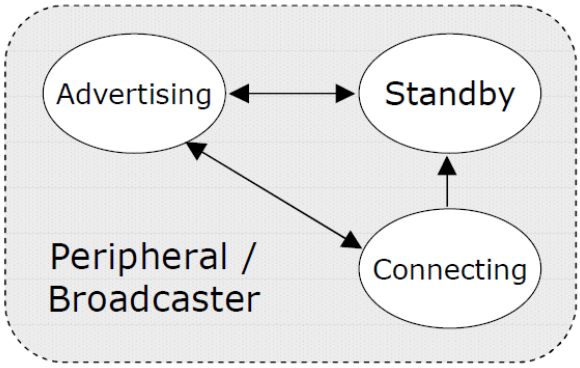

Figure 1.3: Peripheral states

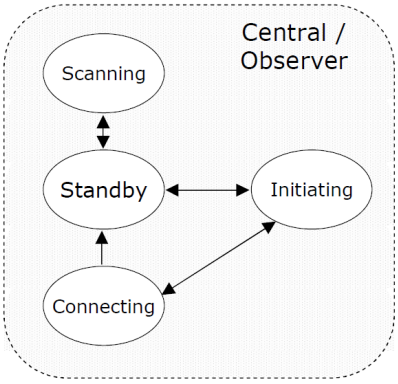

Figure 1.4: Central states

Because of the optimized design of the protocol, Bluetooth Low Energy is even able to allow applications to operate on a coin cell battery for over one year [20]. The energy consumption of BLE is also analysed in [23, 24], showing that BLE achieves lower power consumption compared to other wireless protocols such as ZigBee. The

results in [24] indicate that Bluetooth Low Energy consumes extremely little energy, and has a very attractive ratio of energy per bit transmitted.

Bluetooth Low Energy has been applied in various fields in recent years. For example in [25], BLE is integrated into an energy management system for smart homes, and is used for communication among home appliances. The simulation results in [25] illustrate that the system is more efficient in terms of reducing electricity consumption charges and increasing the comfort level of consumers with the usage of Bluetooth Low Energy. Another example is location fingerprinting with Bluetooth Low Energy beacons proposed in [26]. The results show advantages to the use of BLE beacons for positioning, achieving <2.6 m error for BLE network compared to <8.5 m for WiFi (Wireless Fidelity) network. Also in [27], the physical parameters, such as soil moisture and temperature, are transmitted from the sensors using Bluetooth Low Energy.

1.1.2 Compressed Sensing

Compressed sensing (CS), also referred to as compressive sensing or compressive sampling, has recently attracted a lot of attention in the areas of signal processing and information theory [28–30]. The motivation behind Compressed Sensing is to do "sampling" and "compressing" at the same time. According to traditional wisdom, in order to fully recover a signal, the sampling rate must be equal to or greater than the Nyquist sampling rate, which is twice the maximum frequency of the signal. This scheme uses tremendous resources to acquire often very large signals, and throws away information during compression. This process of "sampling at full rate" and then "throwing away in compression" sometimes is a waste of sensing and sampling resources in application scenarios, where such resources like sensors, energy, observation time, etc. are limited.

The key objective of compressed sensing is to reconstruct a high-dimensional signal accurately and exactly from a set of few non-adaptive linear measurements or samplings. Generally, signals in this context are represented by linear-space vectors, and many of them will represent images or other objects in the application. The fundamental theorem of linear algebra tells us that in general it is not possible to reconstruct an arbitrary signal from an incomplete set of linear measurements. However, it is now known that many signals such as real-world images or audio signals are sparse or compressible over some basis, which makes recovering these signals

accurately from incomplete linear measurements possible.

Now we know that the signal itself is sparse, but it is still not easy to recover the signal from the compressed measurements because the locations of non-zero or significant components are unknown. It is clear that the recovery requires more sophisticated techniques than using a simple linear operator. There are a lot of recovery algorithms used in the Compressed Sensing field with provable results, for example Basis Pursuit algorithm (namely the ℓ_1-minimization algorithm) [31], Orthogonal Matching Pursuit (OMP) algorithm [32] and iterative re-weighted least squares (IRLS) algorithm [33].

According to Compressed Sensing theory, $\mathbf{x} \in \mathbb{R}^N$ is a finite-length, one-dimensional, discrete-time signal, which can be considered as an $N \times 1$ column vector (called input frame of size N). Any signal in R^N can be expressed in terms of a basis of N-dimensional vectors $\{\boldsymbol{\psi}_i\}_{i=1}^N$. Using the $N \times N$ basis matrix $\boldsymbol{\Psi} = [\boldsymbol{\psi}_1|\boldsymbol{\psi}_2|...|\boldsymbol{\psi}_N]$ with columns $\{\boldsymbol{\psi}_i\}$, $\mathbf{x}$ can be stated as

$$\mathbf{x} = \sum_{i=1}^{N} s_i \boldsymbol{\psi}_i \text{ or } \mathbf{x} = \boldsymbol{\Psi}\mathbf{s} \tag{1.1}$$

where $\mathbf{s}$ is the $N \times 1$ column vector of weighting coefficients.

If the N-dimensional vector $\mathbf{s}$ has $N - K$ zero or near zero values, then signal $\mathbf{x}$ is called K-sparse in $\boldsymbol{\Psi}$ domain. And $\mathbf{y}$ is a vector of M linear projections of $\mathbf{x}$ onto another basis $\boldsymbol{\Phi}$ ($M < N$)

$$\mathbf{y} = \boldsymbol{\Phi}\mathbf{x} = \boldsymbol{\Phi}\boldsymbol{\Psi}\mathbf{s} = \boldsymbol{\Theta}\mathbf{s} \tag{1.2}$$

where $\boldsymbol{\Phi}$ is an $M \times N$ matrix, named the sensing matrix [29, 34].

The optimization based on the l_1 norm is expressed as

$$\begin{aligned} \hat{\mathbf{s}} &= \text{argmin}\|\mathbf{s}'\|_1 \\ \text{s.t. } &\boldsymbol{\Theta}\mathbf{s}' = \mathbf{y} \end{aligned} \tag{1.3}$$

It can exactly recover K-sparse signals with high probability when the two conditions below are satisfied:

1. $M > K$
2. Restricted Isometry Property (RIP) [35]

$$(1 - \delta_S)\|\mathbf{s}\|_{l_2}^2 \leq \|\boldsymbol{\Phi}\boldsymbol{\Psi}\mathbf{s}\|_{l_2}^2 \leq (1 + \delta_S)\|\mathbf{s}\|_{l_2}^2 \tag{1.4}$$

where $\delta_S \in (0, 1)$ and is not too close to one.

The distinctive advantages of Compressed Sensing lead to many applications in different fields. One example is a mobile phone camera sensor in [36]. The CS-based approach reduces the acquisition energy per image by as much as a factor of 15 at the cost of complex decompression algorithm. Compressed sensing also showed outstanding results in the application of network tomography to network management [37]. Network delay estimation and network congestion detection can both be modelled as underdetermined systems of linear equations where the coefficient matrix is the network routing matrix. Another important application is on magnetic resonance imaging (MRI) [38] and X-ray computed tomography (CT) [39]. Using CS can help lower the scan time and exposures while producing high-quality image. In addition to the above, Compressed Sensing is also applied in areas such as facial recognition [40], holography [41] and radio astronomy [42].

Moreover, an ECG signal can be well approximated by a compact representation in the wavelet domain, Compressed Sensing has been applied to ECG by some researchers. Matrix from the normal distribution is often used as the sensing matrix. In addition, different algorithms have been used, such as the weighted ℓ_1 minimization algorithm [43] and Simultaneous Orthogonal Matching Pursuit (SOMP) [44]. The MIT-BIH Arrhythmia Database is commonly chosen to evaluate the reconstructed signal quality, and the compression ratio (CR) and the percentage root-mean-square difference (PRD) are often used as performance measure. Results show that the ECG signal can be recovered with high accuracy using Compressed Sensing.

1.2 Contributions

The main contributions of this thesis are summarized as follows.

First of all, a low-power Wireless ECG System is proposed. The conventional ECG monitors and Holter ECG devices are large-sized, high-cost, and not suitable for long-term measurement. Also, the ECG data cannot be accessed by the user easily. The proposed Wireless ECG System includes an ECG sensor, an Android application and a cloud server including a cloud database. The ECG sensor is small-sized, light-weight and conveniently portable. Besides, the cost of the proposed ECG system is low, compared to the conventional ones. Because of the application of Bluetooth Low Energy protocol, the proposed Wireless ECG System is power-efficient and able to work continuously for a long time. Also, both real-time cardiograph and heart

rate are able to be viewed on the Android smartphone, and the user, his/her family members or doctors will be notified when the ECG signal is abnormal, for example due to low heart rate. The patient's ECG data can also be uploaded to and retrieved from the cloud server.

In addition, a newly designed Compressed Sensing matrix and a modified CoSaMP reconstruction algorithm are proposed. This new Compressed Sensing algorithm is able to reduce the energy cost of the ECG sensor in the proposed Wireless ECG System, and also reduce the reconstruction time cost. Moreover, the proposed algorithm is implemented on the Wireless ECG System.

1.3 Thesis Outline

The rest of this thesis is organized as follows:

Chapter 2 proposes a low-power Wireless ECG System to collect, transmit and store the patient's electrocardiogram information with Bluetooth Low Energy. The ECG sensor, Android smartphone application and cloud server are designed and implemented.

Chapter 3 introduces a power-efficient method to collect ECG signal using Compressed Sensing with a newly designed sensing matrix. A modified CoSaMP reconstruction algorithm is also proposed to reduce the reconstruction time. Moreover, the proposed algorithm is applied to the Wireless ECG System.

Chapter 4 concludes the thesis and suggests possible future work.

Chapter 2

Wireless ECG System with BLE

Electrocardiogram (ECG) is widely used for monitoring heart activity and discovering cardiovascular diseases, and many heart diseases can be diagnosed early from ECG plots. However, traditional ECG monitors, such as Holter monitors, are often not easy-carry and power-efficient. Moreover, the collected ECG data is not easy to be obtained by either patient or doctor when the patient is not using the monitor at hospital. In addition, the incapability of long-term monitoring makes it hard to capture the abnormal heart activity which occurs occasionally, and also impossible to send an alarm when the patient is in a dangerous situation, for example a heart attack.

Bluetooth Low Energy (BLE) was developed specially for low-power applications like wireless sensor network (WSN). Many hardware companies have introduced chip solutions for BLE, making the implementation of BLE based applications possible. In 2012, TI (Texas Instruments) launched the CC2540 BLE chip which is a one-chip integrated solution [20]. The CC2540 system-on-chip device has ultra-low power consumption, and it supports both single mode and dual mode BLE solutions. More importantly, the 6mm×6mm×0.85mm dimension is suitable for a portable wireless ECG sensor.

According to the result in [45], 64% of American adults now own a smartphone of some kind, and they use it as a handheld computer to accomplish tasks anywhere anytime. The latest smartphones nowadays come with powerful on-board computing capability, large screens, capacious memories and advanced operating systems. Platforms available today include Android, Apple iOS and Windows Mobile, among which Android has the largest market share of 82.8% worldwide [46]. Android supports Bluetooth Low Energy protocol since version 4.3. An Android phone can behave

as a central device, and communicate with other peripheral devices such as a BLE ECG sensor. Moreover, with the WiFi and data connectivity, the Android phone is able to exchange data with a cloud server remotely.

In this chapter, we propose a wireless ECG system based on Bluetooth Low Energy. The whole system includes an ECG sensor based on TI CC2540 chip, an Android application and a web server with a back-end database. ECG data is collected by the ECG sensor, and then sent to an Android smartphone wirelessly through the BLE protocol. Furthermore, the ECG data can be uploaded to or downloaded from the web server by WiFi or mobile data connection. In addition, the Android app is able to send an alarm in low-heartbeat situations.

The remainder of this chapter is organized as follows. Section 2.1 introduces the model of the whole system. In Section 2.2 the BLE based ECG sensor is proposed. The 'Heart Carer' Android application is implemented in Section 2.3. The design of web service and cloud database is detailed in Section 2.4, and Section 2.5 summarizes this chapter.

2.1 System Model

The proposed Bluetooth Low Energy based wireless ECG system is composed of an ECG sensor board, an Android application and a web service with database. The block diagram of the system overview is presented in Figure 2.1.

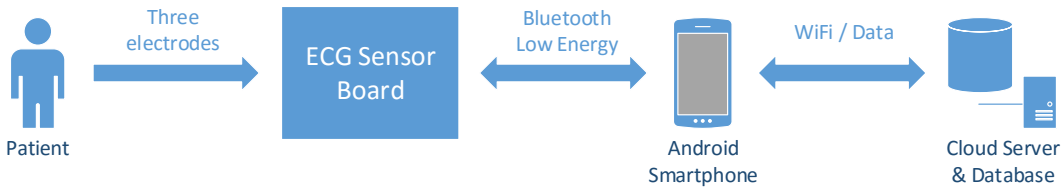

Figure 2.1: Block diagram of the proposed BLE ECG system

The ECG signal is collected from a patient's body through three electrodes connected to the ECG sensor board. After necessary processing, the ECG data is transmitted to an Android application using the Bluetooth Low Energy protocol. Other

information such as control instructions can also be sent or received over BLE. The real-time ECG signal is displayed on the smartphone screen, together with the current heart beat rate, and it can also be saved in the smartphone for further usage. When Internet connection is available, the user is able to upload saved ECG data to the cloud server, or download history data from it. User information, such as username and password, and preferred settings are also stored in the cloud database.

2.2 BLE Based ECG Sensor

The three-lead ECG sensor board in the proposed wireless ECG system, as shown in Figure 2.2, is designed based on the TI CC2540 Bluetooth Low Energy chip. It acquires the original one channel ECG signal from a patient's body through three electrodes. Then, the original signal is sent to the Android smartphone after proper processing, including signal amplification, low pass filtering, etc.

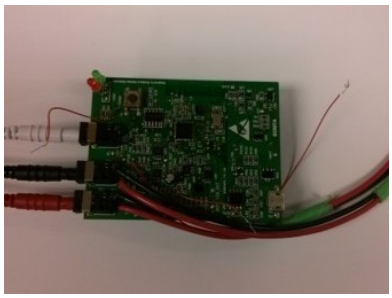

Figure 2.2: BLE based ECG sensor

Figure 2.3: TI CC2540 chip

2.2.1 TI CC2540 Chip

Since the announcement of Bluetooth Low Energy standard, many large hardware companies have released their BLE solutions. TI (Texas Instruments) is one of the earliest ones. Its BLE solutions contain the industry's largest selection of devices, and are the easiest to use and lowest power.

TI CC2540 Bluetooth Low Energy solution, as shown in Figure 2.3, is an ultra-low power, one-chip integrated solution with controller, host and application on one device [20]. The device has an enhanced 8051 MCU, in-system reprogrammable flash memory, integrated AES-128 encryption engine, and also includes peripherals to interface with analog and digital sensors. With low power sleep modes and short transition

times between operating modes, the CC2540 is suitable for systems where ultra-low power consumption is required.

For the purpose of developing and debugging, the CC2540 Mini Development Kit has been used during the development process. The kit includes a Keyfob, a USB Dongle and a CC Debugger [47], as presented in Figure 2.4 [47] from left to right.

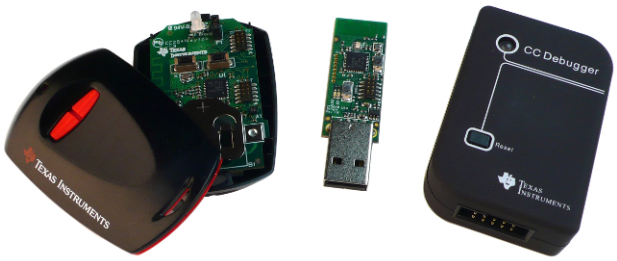

Figure 2.4: CC2540 Mini Development Kit

The Keyfob acts as the BLE Slave. It can be mounted inside the included plastic enclosure. It operates on a single CR2032 coin cell battery, and contains a two-colored LED, a buzzer, an accelerometer, and two buttons. The USB Dongle behaves as the BLE Master. It connects to a Windows PC USB port, and is pre-loaded with the master demo application software. The CC Debugger is used to flash the software onto both the USB dongle as well as the Keyfob.

2.2.2 ECG Sensor Board

The ECG sensor board uses TI CC2540 chip as the microcontroller and BLE transceiver, and it collects the ECG signal from a patient's body through three ECG electrodes. An ECG electrode is a kind of transducer needed to record ECG data. It converts the ionic potentials generated within the body into electronic potentials, which can be measured by conventional electronic instrumentation. We use 3M® Red Dot electrodes in our system, which are inexpensive and disposable.

The three lead ECG has a long history, and has been used in emergency department, telemetry monitoring and other fields [48]. Three coloured wires connect to three electrodes to form a triangle - Einthoven's triangle, as given in Figure 2.5 [48]. The three electrodes are coloured white, black and red. The white electrode is placed just below the clavicle (collarbone) on the right shoulder, labelled "RA" for right arm.

And the black electrode is connected below the left clavicle near the shoulder, labelled "LA" for left arm. The red electrode is connected below the left pectoral muscle near the apex of the heart. The end of the red electrode cable is usually labelled "LL" for left leg.

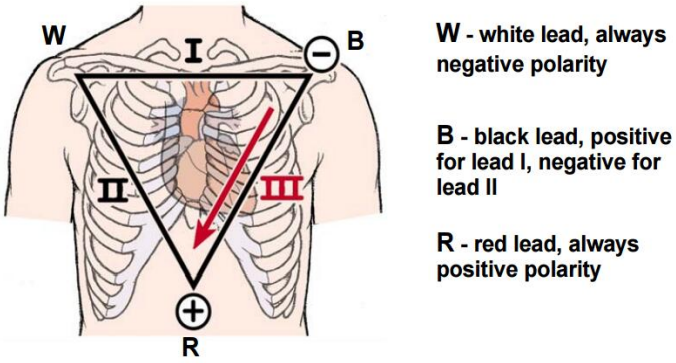

Figure 2.5: The 3-lead cardiac monitoring system

The sampling rate of the ECG sensor is 1 kHz which is adequate for an ECG signal, and the resolution is 8 bits. There is a physical button on the sensor board, controlling the sending of connectable advertisement. When not in connection, the sensor board will begin advertising if the button is pressed. Meanwhile, the Android phone, i.e. central device, will scan and find the ECG sensor, then establish the connection between the two devices.

2.2.3 BLE Basic Concepts

As the firmware is developed and works based on the Bluetooth Low Energy protocol, here we first elaborate the protocol layers and their detailed responsibilities. The whole BLE protocol stack architecture is presented in Figure 2.6 [21], consisting of three main sections: Controller, Host and Application (Apps) [21].

Controller includes the lower layers of the BLE protocol stack, including Physical Layer (PHY) and Link Layer (LL) [22]. **Host** contains the upper layers of the protocol stack, including Logical Link Control and Adaptation Protocol (L2CAP), Security Manager Protocol (SMP), Attribute Protocol (ATT), Generic Attribute Profile (GATT) and Generic Access Profile (GAP). **Application** is the user application interfacing with the BLE protocol stack to cover a particular use case. Additionally, the Host Controller Interface (HCI) is provided to allow interoperability between hosts and controllers produced by different companies.

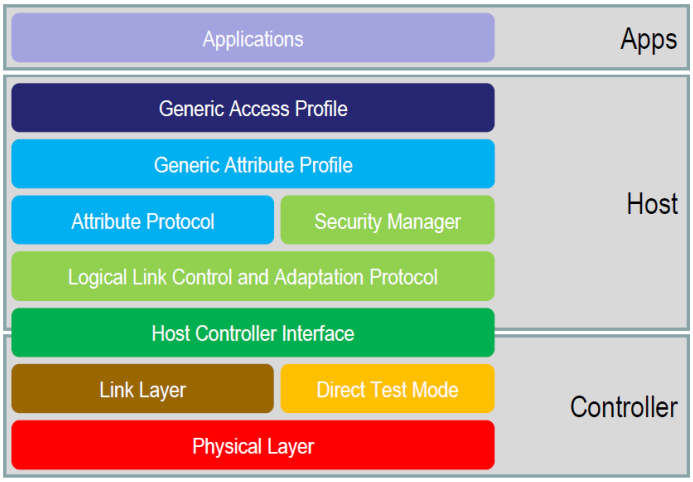

Figure 2.6: BLE protocol stack

Among the layers listed above, Attribute Protocol (ATT) and Generic Attribute Profile (GATT) is the key of BLE protocol and low energy cost. In Bluetooth Low Energy, servers expose data to clients using attributes as shown in Figure 2.7 [50]. Note that the Client/Server role of a device is independent of the Central/Peripheral or the Master/Slave role, i.e. it is possible for a device to act as both a client and server simultaneously [49].

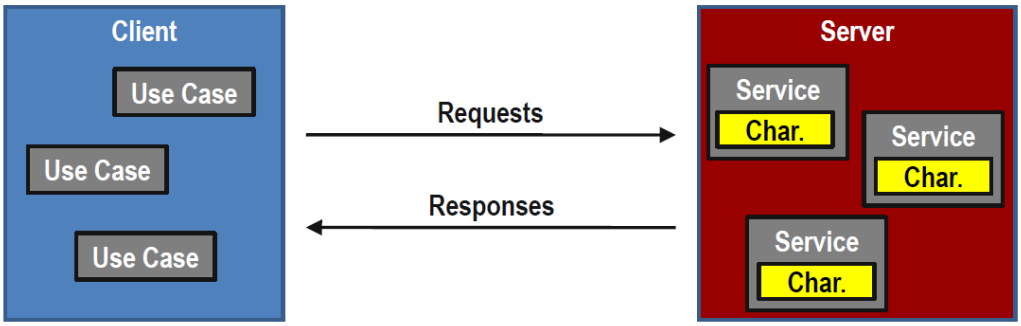

Figure 2.7: Client server architecture

The concept of an attribute is a discrete value containing four properties: a value, a handle (address), a type and a set of permissions. **Handle** is used to address an individual attribute. **Type** tells what the data represents, and it can be a UUID (universal unique identifier) assigned by the Bluetooth SIG, or a custom type. **Permissions** enforces if and how the attribute client can access the attribute's value. An example of an attribute table is given in Table 2.1.

GATT is designed for use by the application or a profile, so that an attribute client

Handle	Type	Permissions	Value
39	0x2800 (GATT Service UUID)	Read	E0:FF (2 bytes)
40	0x2803 (GATT Characteristic UUID)	Read	10:29:00:E1:FF (5 bytes)
41	0xFFE1 (Simple Keys state)	(none)	00 (1 byte)
42	0x2902 (GATT Client Characteristic Configuration UUID)	Read and Write	00:00 (2 bytes)

Table 2.1: Attribute table example

can communicate with attribute server. The top level of the hierarchy is a profile. A **profile** is composed of one or more services necessary to fulfill a use case. A **service** is composed of characteristics or references to other services. Each **characteristic** contains a value and may contain optional information about the value, such as a descriptor.

When two BLE devices are in the connected state, the GATT client device can perform actions as follows: discover characteristic by UUID, read/write characteristic value or confirm an Indication. The GATT server can respond to client actions, send Notification or Indication which both don't need a read request. The **Notification** does not need confirmation from the client, while the **Indication** must be acknowledged before any further data can be sent.

2.2.4 Firmware and ECG Data Transmission

In the proposed wireless ECG system, the Heart Rate Profile (HRP) [51] is implemented. The profile defines two roles: Heart Rate Sensor and Collector. The **Heart Rate Sensor** is the device measuring heart rate and other information, which is the ECG sensor board in our system, and it shall be a GATT Server. The **Collector** is the device that receives the measurement and other data from a Heart Rate Sensor, corresponding to the Android smartphone in the proposed system, and it shall be a GATT Client. The Heart Rate Sensor instantiates the Heart Rate Service (HRS) which is defined in [52].

In the Heart Rate Service, the following characteristics are exposed: Heart Rate Measurement, Body Sensor Location and Heart Rate Control Point. Only the Heart

Rate Measurement characteristic is used in our system, and the measurement value is sent out in form of Notification. There is also a descriptor included in the Heart Rate Measurement characteristic, named 'Heart Rate Measurement Client Characteristic Configuration descriptor', and its permission is Read/Write. The descriptor is used to configure the measurement notification.

The UUID of Heart Rate Service is 0x180D, and 0x2A37 for Heart Rate Measurement characteristic. For the configuration descriptor of Heart Rate Measurement characteristic, the UUID is 0x2902, and it can be written "01:00" to enable or "00:00" to disable the measurement Notification.

The firmware on the CC2540 microcontroller is developed based on TI 'SimpleBLEPeripheral' sample project, using IAR Embedded Workbench for 8051. The sample project is also integrated with Heart Rate Profile so that the device can behave as an ECG sensor. As the measured data is sent through the form of Notification, the actual working state of the ECG sensor depends on whether the Notification is turned on or off. The Notification state can be switched by two ways: the physical button on the sensor board or the 'Sense' button in the Android application, as shown in Figure 2.8. And the state is indicated by a boolean variable 'ifNotiOn' in our firmware program.

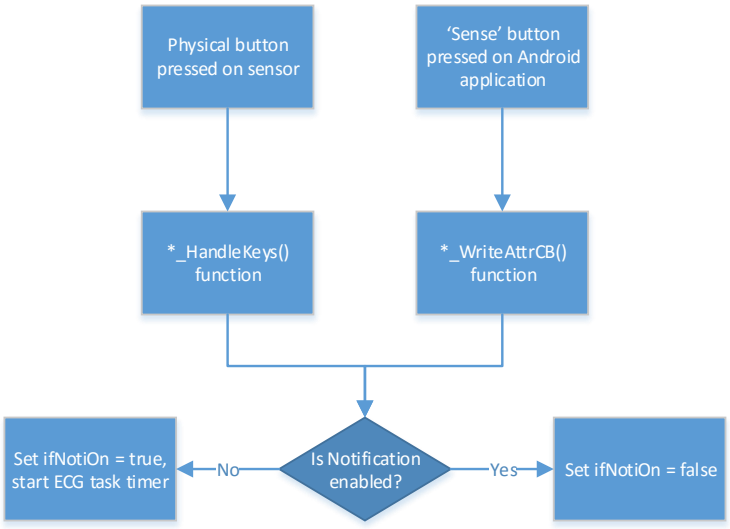

Figure 2.8: Switch Notification state

The ECG task timer in Figure 2.8 will trigger the 'ECG Event' defined in the program after predefined 'ECG Event Period' time, which is 1ms as the ECG sampling

rate is 1kHz. The whole 'ECG Event' flow is presented in Figure 2.9. As the incoming ECG signal is an analog signal, the ECG measurement values to be sent are achieved from ADC (Analog-to-Digital Converter) with 8-bit resolution, i.e. 1 byte, and the value range is from 0 to 255.

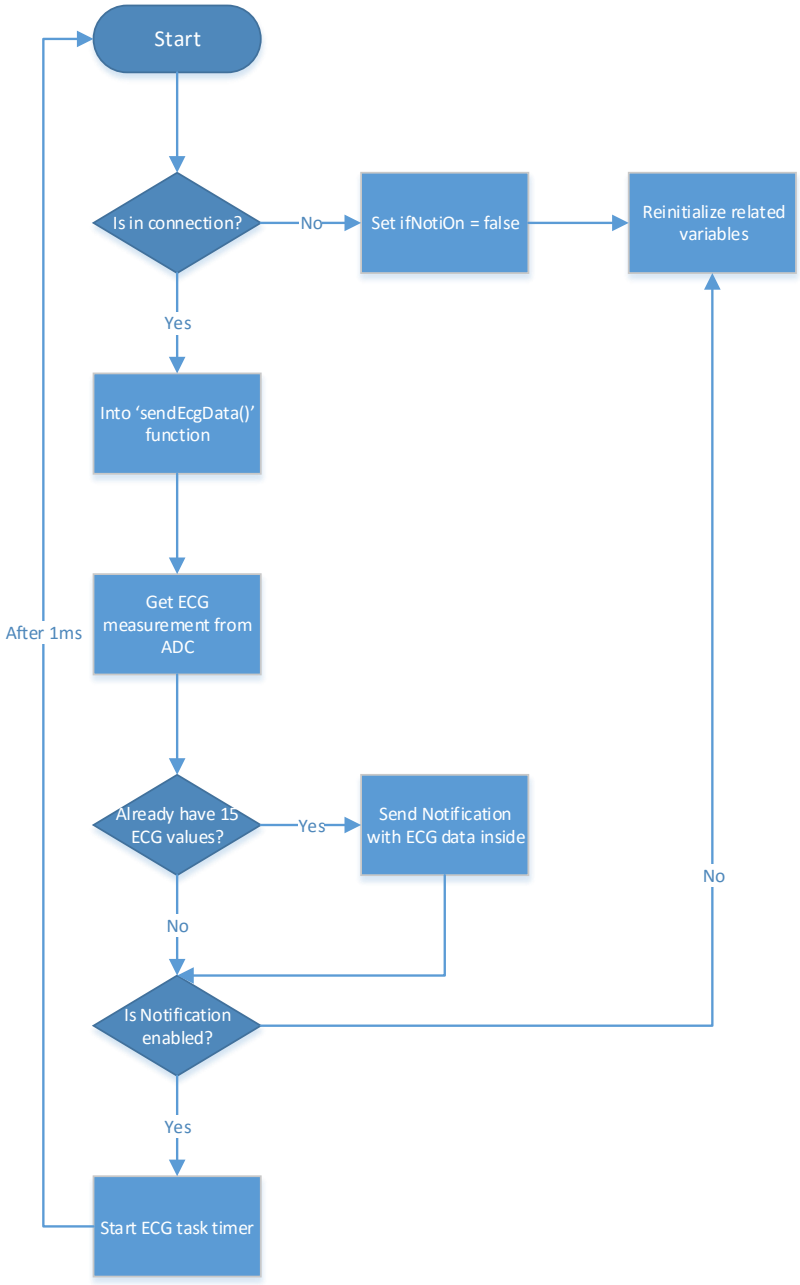

Figure 2.9: 'ECG Event' flow

For TI CC2540, the provided 'GATT_Notification()' function is used to send Notification to the GATT Client. However, the Notification is not actually sent immediately after the function is called. Notifications are first saved to a buffer during the connection interval, and then sent out the next time connection is established. The TI firmware also restricts the payload of one Notification to be 20 bytes, and the minimum and maximum connection interval should be multiple of 1.25ms. According to our own experiment results and other researchers' comments on the Internet, we discover that sending more than three Notifications during one connection will cause packet loss, which is unacceptable. Therefore in the end, we choose to set the minimum and maximum connection interval both to be 45ms, and 3 Notifications will be sent during each connection while one Notification contains 15 ECG measurement values. There is also one byte used as counter in each Notification, which can be used to order the Notifications that are sent together.

2.3 Android Application

The Android application in the proposed wireless ECG system is developed based on Android version 4.3. It receives the patient's ECG data from the sensor board via Bluetooth Low Energy. The ECG data can be viewed in real time and stored on the Android smartphone. The user can also upload the history data to the cloud database, or download from it. Moreover, alarm can be set to send when the heartbeat is low.

2.3.1 Android Fundamentals

Android is a mobile operating system (OS) currently developed by Google [53]. It is based on the Linux kernel and designed primarily for touchscreen mobile devices such as smartphones and tablets. In addition to touchscreen devices, Google has further developed Android TV for televisions, Android Auto for cars, and Android Wear for wrist watches, each with a specialized user interface. Variants of Android are also used on notebooks, game consoles, digital cameras, and other electronics.

Android apps are written in the Java programming language. XML (Extensible Markup Language) is also used in Android for layouts, menus, styles, etc. App source code, along with any data and resource files, is complied into an APK by the Android SDK (Software Development Kit) tools. An APK is an Android package, which is an archive file with an '.apk' suffix. One APK file contains all the contents of an Android

app and is the file that Android-powered devices use to install the app.

2.3.1.1 Android History and Versions

Android was initially developed by Android Inc., which was bought by Google in 2005 [53]. In 2007, Android was unveiled along with the Open Handset Alliance (OHA), a consortium of hardware, software and telecommunication companies devoted to advancing open standards for mobile devices. As this announcement indicated, Android would be much different from the iPhone OS in that it would not be limited to simply one device from one manufacturer. In October 2008, the first commercially available Android smartphone 'HTC Dream' was released. According to the latest statistics, Android dominated the smartphone market with a share of 82.8%, while the Apple iOS is in the second place with a market share of 22.3% [46].

The fist version of Android is Android alpha, which was released in November 2007. And in September 2008, Google released the first commercial version, Android 1.0. Android has been continually developed by Google and the OHA. The most recent major update is Android 6.0 "Marshmallow", released in October 2015. Updates of Android version also come with API (Application Programming Interface) updates. The latest Android 6.0 corresponds to API Level 23.

2.3.1.2 App Components

App components are the essential building blocks of an Android app [54]. Each component is a different point through which the system can enter your app. There are four different types of app components: Activity, Service, Content Provider and Broadcast Receiver. Each type serves a distinct purpose and has a distinct lifecycle that defines how the component is created and destroyed.

2.3.1.2.1 Activity

An Activity is an application component that provides a screen with which users can interact in order to do something. Each Activity is given a window in which to draw its user interface, and the window usually fills the screen. An application usually consists of multiple Activities that are loosely bound to each other. Typically, one Activity in an application is specified as the "main" Activity, which is presented to the user when launching the application for the first time. Each Activity can then start another Activity in order to perform different actions.

An Activity can exist in essentially three states:

- Resumed: The Activity is in the foreground of the screen and has user focus.
- Paused: Another Activity is in the foreground and has focus, but this one is still visible. A paused Activity is completely alive, but can be killed by the system in extremely low memory situations.
- Stopped: The Activity is in the background and not visible to the user. It's still alive, but can be killed by the system when memory is needed elsewhere.

Figure 2.10 shows the important state paths of an Activity. The square rectangles represent callback methods, which can be implemented to perform operations when the Activity moves between states. The coloured ovals are the major states the Activity can be in.

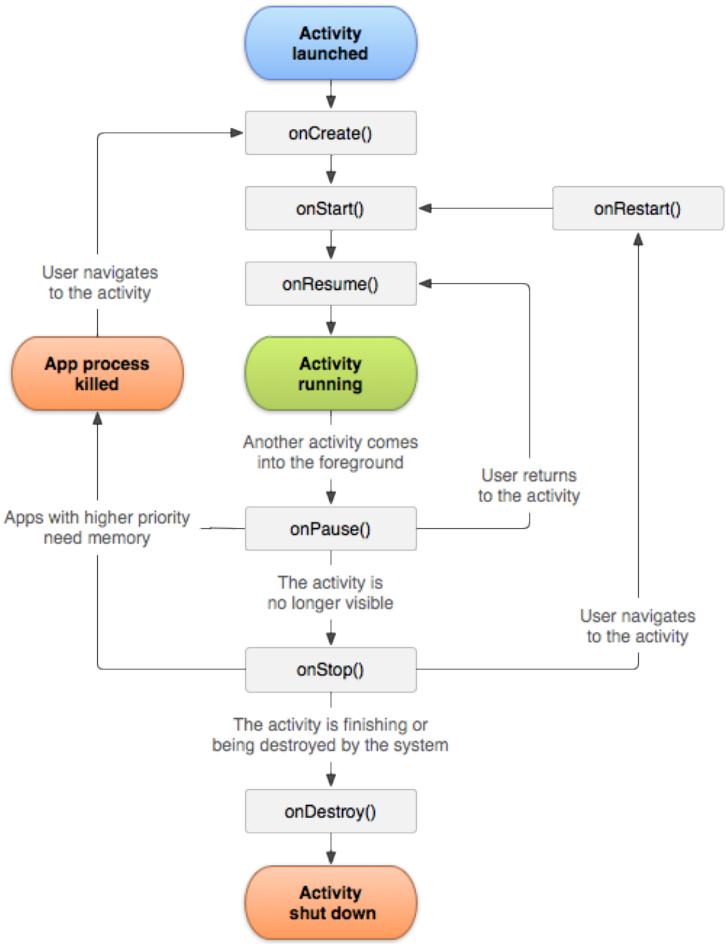

Figure 2.10: Activity lifecycle

A Fragment represents a behaviour or a portion of user interface in an Activity. Multiple Fragments can be combined in a single Activity to build a multi-pane UI, and one Fragment can also be reused in multiple Activities. An example of how Fragment is used in UI design is shown in Figure 2.11 [55]. Moreover, a Fragment must always be embedded in an Activity and the Fragment's lifecycle is directly affected by the host activity's lifecycle.

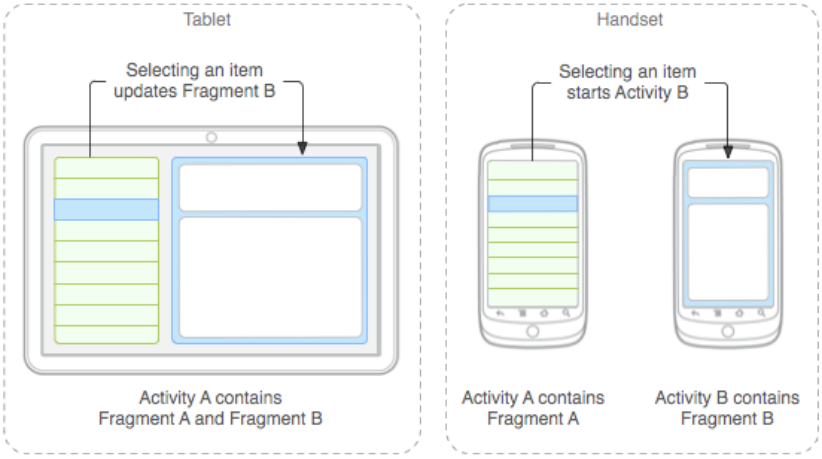

Figure 2.11: Example of UI design for tablet and handset with Fragments

2.3.1.2.2 Service

A Service is a component that runs in the background to perform long-running operations or to perform work for remote processes, and it does not provide a user interface. An application component can start a Service and it will continue to run in the background even if the user switches to another application. Additionally, a component, such as an Activity, can bind to a Service to interact with it and even perform interprocess communication (IPC).

A Service can essentially take two forms:

- Started: A Service is "started" when an application component starts it by calling 'startService()' function. Usually, a started Service performs a single operation and does not return a result to the caller.
- Bound: A Service is "bound" when an application component binds to it by calling 'bindService()' function. A bound Service offers a client-server interface that allows components to interact with the Service. It runs only as long as another application component is bound to it.

The Service lifecycle is presented in Figure 2.12.

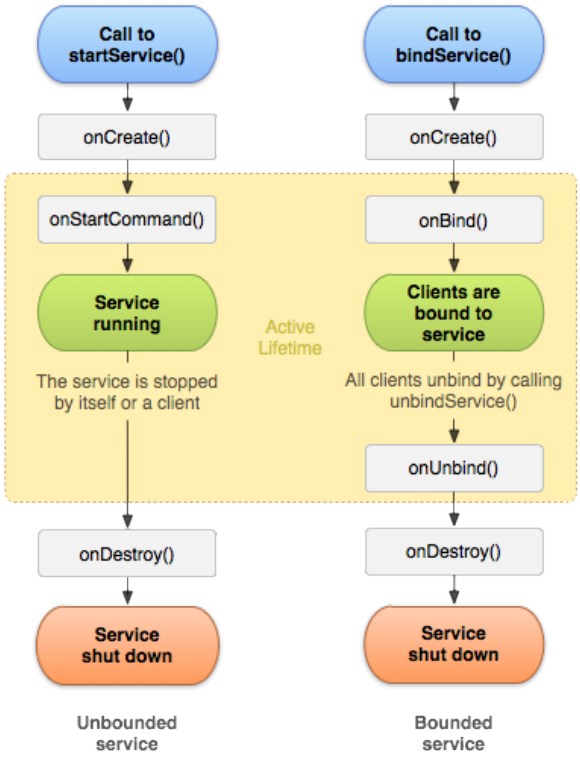

Figure 2.12: Service lifecycle

2.3.1.2.3 Content Provider

A content provider manages a shared set of app data. The data can be stored in the file system, an SQLite database, on the web, or any other persistent storage location your app can access. Through the content provider, other apps can query or even modify the data (if the content provider allows it). For example, the Android system provides a content provider that manages the user's contact information. As such, any app with proper permissions can query part of the content provider to read and write information about a particular person.

2.3.1.2.4 Broadcast Receiver

A broadcast receiver is a component that responds to system-wide broadcast announcements. Many broadcasts originate from the system, for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured. Apps can also initiate broadcasts, for instance, to let other apps know that some data has been downloaded to the device and is available for them to use.

2.3.1.3 The Manifest File

Before the Android system can start an app component, the system must know that the component exists by reading the app's 'AndroidManifest.xml' file (the "manifest" file). The manifest file must be at the root of the app project directory, and all the app's components must be declared in it. Its responsibilities also include:

- Identify any user permissions the app requires, such as Internet access or read-access to the user's contacts.
- Declare the minimum API Level required by the app, based on which APIs the app uses.
- Declare hardware and software features used or required by the app, such as a camera, Bluetooth services, or a multi-touch screen.
- API libraries which the app needs to be linked against (other than the Android framework APIs), such as the Google Maps library.
- And more

2.3.1.4 App Resources

An Android app is composed of more than just code. It requires resources that are separate from the source code, such as images, audio files, and anything related to the visual presentation of the app. For example, developers should define animations, menus, styles, colors, and the layout of activity user interfaces with XML files. Using app resources makes it easy to update various characteristics of the app without modifying code, and developers can optimize the app for a variety of device configurations (such as different languages and screen sizes) by providing sets of alternative resources.

For every resource included in the Android project, the SDK build tools define a unique integer ID, which can be used to reference the resource from the app code or from other resources defined in XML. For example, if the app contains an image file named 'logo.png' (saved in the 'res/drawable/' directory), the SDK tools will generate a resource ID named 'R.drawable.logo', which can be used to reference the image and then insert it in the user interface.

2.3.2 Wireless ECG Application Architecture

The Android application in the proposed wireless ECG system is named 'Heart Carer'. It is developed based on Android version 4.3, i.e. API Level 18, using Eclipse IDE (Integrated Development Environment) with ADT (Android Development Tools) plugin. The app is composed of Activities, Fragments, Services, and other resources. The whole architecture and workflow of the Android application is shown in Figure 2.13.

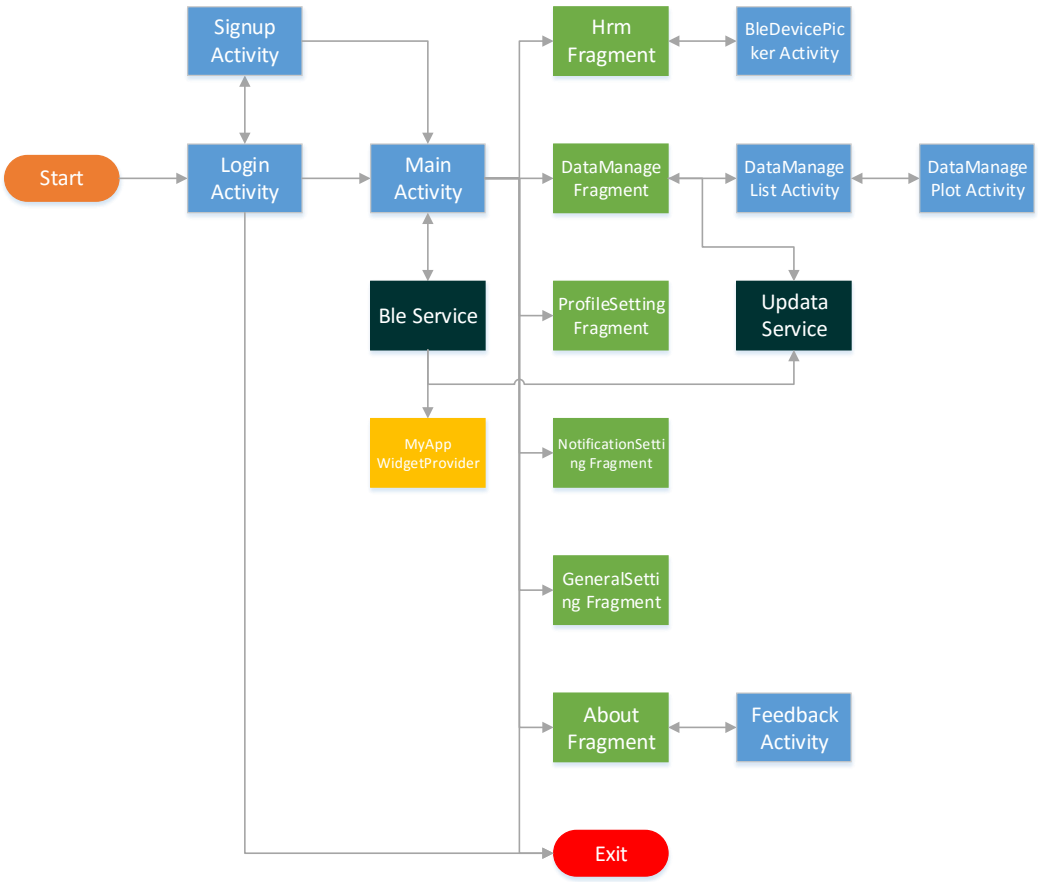

Figure 2.13: Application architecture

When a user launches the app for the first time, they will get into **Login Activity**, i.e., the "main" Activity. A user can choose to log in or continue without account, then they will reach Main Activity. If a user does not have an account and wants to register one, they can go to **Signup Activity** before getting to Main Activity.

There are six Fragments contained in the **Main Activity** and only one of them is shown at one time. The **Hrm Fragment** shows the real-time ECG signal, BLE

connection state, and the current heart rate. It also has three control buttons for sensing, plotting and saving files. The user can get to **BleDevicePicker Activity** to pick a BLE ECG sensor to connect. In the **DataManage Fragment**, user can select a date range to search their history ECG data within the range. The search results will be shown in **DataManageList Activity**, grouped by date. The user can choose one history record to plot, and this will open the **DataManagePlot Activity**. The **ProfileSetting Fragment** allows the user to update their email address, phone number, name, and password. And in the **NotificationSetting Fragment**, the user can change the emergency phone number and the emergency SMS (Short Message Service) message. The **GeneralSetting Fragment** provides the controls of some basic system functions, such as Bluetooth and WiFi. The user can also set the duration of saving files and the low heart beat value here. In the **About Fragment**, information about the app is presented, and there is also a button for checking updates. User can go to the **Feedback Activity** to give feedback about the app from the About Fragment.

There are also two Services in the program. The **Ble Service** is started by the Main Activity, and it will keep running until the user exits the app. It is in charge of the BLE connection and the data through it. It will also update the heart rate value in the Widget, which is created by **MyAppWidgetProvider**. The **Updata Service** is used to upload certain local ECG records to the cloud server. It can be started by Ble Service or DataManage Fragment, and it will kill itself when the uploading is done.

2.3.3 Activities and Fragments

In the proposed Android application, there are totally seven Activities and six Fragments. They all have associated user interfaces for the user to interact with the app. The function and UI design of them will be detailed in the following sections.

2.3.3.1 Login Activity

The Login Activity is the “main” Activity. It is the first screen a user will see after launching the application. The user interface is shown in Figure 2.14. There are two ‘EditText’ fields for user to fill in the username and password information. The password information is concealed because the ‘inputType’ of the ‘EditText’ is set to ‘textPassword’. Moreover, a ‘CheckBox’ for remembering password is located below

them. If it is checked, the password will be filled in automatically the next time the user opens the app. The username is always remembered, and it is saved in the Android system together with the password, using 'SharedPreferences' API. The information is saved securely using the 'private mode', so it is not accessible by other applications.

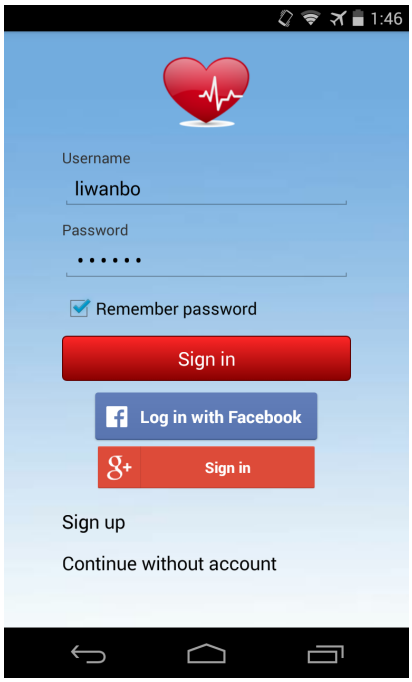

Figure 2.14: Login Activity

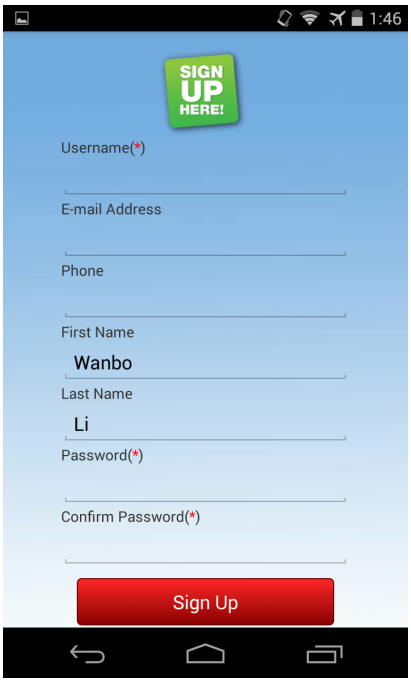

Figure 2.15: Signup Activity

Below the 'CheckBox' are three buttons for logging in to the application. If the credential is correct, the user will be taken to the Main Activity. The first one is for logging in with the typed in username and password. If there is no Internet connection or the username/password pair is wrong, the app will warn the user in form of a 'Toast' [56], i.e., a short message shown on the screen for seconds. The other two buttons are for logging in using a Facebook or Google+ account. They are added here for the convenience that many people already have those two accounts, and it will be convenient for them to just use them. As the supporting APIs for them are both from external libraries, changes are made in the 'AndroidManifest.xml' file accordingly to include the two libraries.

There are also two 'TextView' fields at the bottom of the view. One is for the user to create a new account, the other one is for the situation that the user wants to continue using the app without logging in. In the first case, the user will be taken

to the Signup Activity.

2.3.3.2 Signup Activity

The Signup Activity is designed for the user to register a new account for using the application. The UI layout is given in Figure 2.15, which mainly includes several 'EditText' fields and a button. The user needs to provide at least a username and a password to sign up, which can be told from the red star sign in brackets. Other information, such as email address and phone number, can also be added to the newly created account. User can also choose to auto-fill the phone number, name and email address using the default information saved in the phone when entering the Signup Activity.

The user can submit the filled-in information by clicking the 'Sign Up' button, and if the account is created successfully, the user will get into the Main Activity and begin using the main function of the app. In addition, after the 'Sign Up' button is pressed, some checks will first be done by the app and the cloud server. For example, whether the username has already been used, or whether the texts in 'Password' and 'Confirm Password' fields match. If something goes wrong, a 'Toast' will be generated to warn the user.

2.3.3.3 Main Activity

The Main Activity includes six different Fragments. They can be toggled between each other through the 'Navigation Drawer', as shown in Figure 2.16. The Navigation Drawer is a panel that displays the app's main navigation options on the left edge of the screen. It is hidden most of the time, but is revealed when the user swipes a finger from the left edge of the screen, or touches the app icon in the action bar. The action bar, also known as the app bar, locates at the very top of the whole app. In the proposed application, when the Navigation Drawer is hidden, the action bar will show the particular icon and name of the current Fragment; when the Navigation Drawer is unfolded, the app icon and name will be presented together with the currently logged in username.

There are seven items in the presented Navigation Drawer list. The first six items are for Fragment navigation, and the last one is for exiting the app. The layout for each list item is defined in 'drawer_list_item.xml' file, consisting of one 'ImageView' field and one 'TextView' field which correspond to the item icon and description.

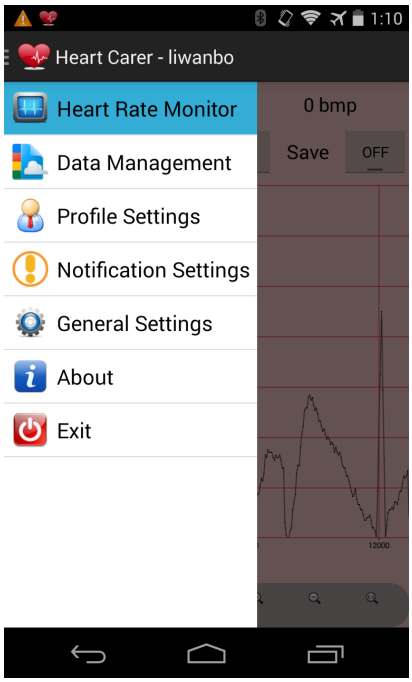

Figure 2.16: Navigation Drawer

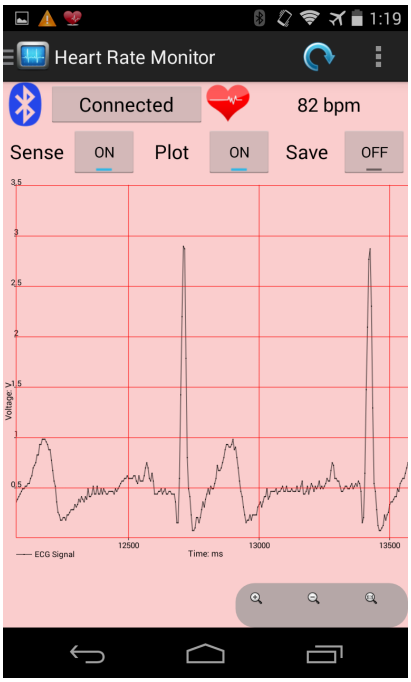

Figure 2.17: Hrm Fragment

Specially for the last item, if 'Exit' is chosen, an alert dialog will be shown with "Yes or No" buttons to ask the user if they really want to exit the app.

During initialization, the Main Activity binds to the Ble Service and then starts it. As the Hrm Fragment needs to communicate with Ble Service but is not able to do it directly, the Main Activity plays the role of a bridge between them. It registers a 'ServiceMessenger' to forward the messages received from Ble Service to Hrm Fragment, and also implements a Java interface which is defined in Hrm Fragment to forward commands from Hrm Fragment to Ble Service. Accordingly, Ble Service also registers an 'ActivityMessenger' for sending data to Main Activity, and in Hrm Fragment, a 'handleMainActivityMes' function is written to handle the incoming messages from Main Activity. The 'Messenger' transmits data in the form of 'Message', which contains a description and arbitrary data object. A user-defined message code is used to identify what this message is about. The flow chart of this inter-object communication is given in Figure 2.18.

2.3.3.4 Hrm Fragment

The Hrm (Heart Rate Monitor) Fragment, as presented in Figure 2.17, is the most important Fragment in the app. It displays the BLE connection status, current

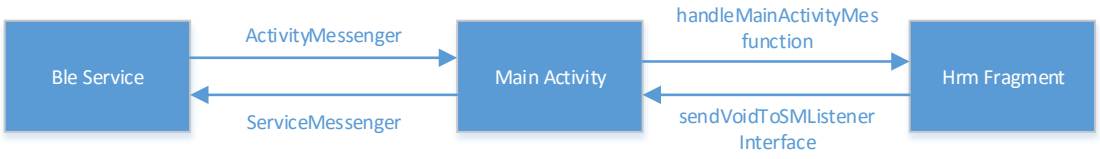

Figure 2.18: Inter-object communication

heart rate, and the real-time ECG signal. The BLE connection status can be 'Not Connected', 'Connecting' or 'Connected', which is shown as 'text' on a 'Button', and the user can change the connection status by clicking the Button. The current heart rate value is set to be 0 bpm (beats per minute) by default.

The ECG signal is plotted using 'AChartEngine' third-party library. The chart has one x-axis and one y-axis, representing time in millisecond and voltage in volt, respectively. The default range for the x-axis is set to be 0 to 1500 ms as this usually covers around two heart beat cycles. And for the y-axis, 0 to 3.5 V is set by default, because the reference voltage of the incoming ECG signal from the sensor board is about 3.3 V. If the length of the plotted ECG signal exceeds the x-axis range, the range will change accordingly. The library also provides three zoom buttons for zooming in, zooming out and restoring to the original scale.

There are also three 'ToggleButton' fields for function controls: 'Sense' button for sensor board's Notification control, 'Plot' button for ECG signal plotting, and 'Save' button for saving real-time ECG data to local files on the Android phone. On the top right corner are four 'OptionsMenu' items. Only one item, 'Clear Graph' button, is revealed, and the others are hidden under the "three dots" button. The functions of the four menu items are as below:

- 'Clear Graph': Clear the current ECG plot and draw ECG signal from the beginning of x-axis.
- 'Select sensor': Switch to BleDevicePicker Activity to pick a BLE ECG sensor to connect with. User will first be asked to enable the system Bluetooth function if it is off.
- 'Sensor info': Pop up an 'AlertDialog' showing the name and MAC address (media access control address) of the sensor chosen in BleDevicePicker Activity.
- 'My device info': Pop up an 'AlertDialog' showing the name and MAC address of the Bluetooth chip in the Android phone.

The app works in 'Normal' mode by default, and it can also work in 'Compressed Sensing' mode which will be elaborated in Chapter 3. When in 'Compressed Sensing' mode, the 'Plot' button and plotting function are disabled.

2.3.3.5 BleDevicePicker Activity

The BleDevicePicker Activity allows the user to scan nearby connectable BLE devices and choose one to establish connection, as shown in Figure 2.19. The top-right-corner menu button can be toggled between 'Scan' and 'Stop', used for starting and stopping the scanning respectively. The scanning will last for 10 seconds, and meanwhile the user need to press the physical button on the ECG sensor to make it visible to the Android phone. If one BLE device is found by the app, the device name and MAC address will be shown in the list below the action bar, and if the name matches 'Heart Carer' (this name is set for the ECG sensor in the firmware), the app icon is displayed, otherwise a question mark is shown. When the user selects one BLE device in the list, the app will return to Hrm Fragment and try connecting to it. The work of scanning is done by a public static 'BluetoothAdapter' defined in Main Activity.

Figure 2.19: BleDevicePicker Activity

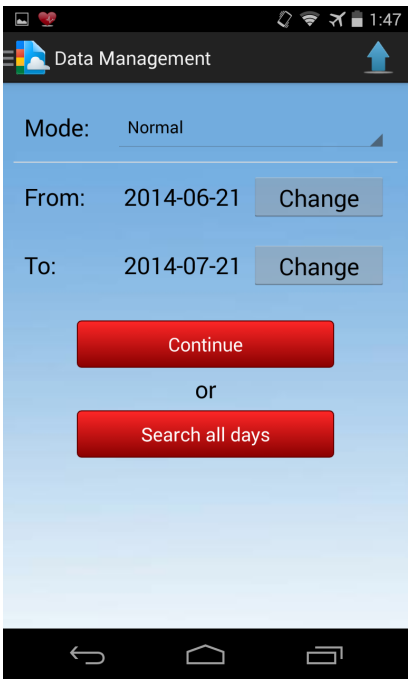

Figure 2.20: DataManage Fragment

2.3.3.6 DataManage Fragment

The DataManage Fragment, as illustrated in Figure 2.20, is designed for searching patient's history ECG records saved in cloud database by mode type and date range. The mode can be chosen from 'Normal', 'Compressed Sensing' and 'Both'. The start date and end date are both set to be the current date by default, and the user is able to change them by clicking the 'Change' button and using the popped up 'DatePickerDialog'. The user can start the search with the selected date range by pressing the 'Continue' button, or use the 'Search all days' button to seek all the history ECG data. If there are history ECG records existing in the selected date range, the DataManageList Activity will be opened and the records will be listed there, otherwise a warning will be shown to the user.

The information of the search results is stored in two arrays and then they are passed to DataManageList Activity. The type of the first one is 'ArrayList' of 'HashMap', saving the dates of the records. The other one's type is 'ArrayList' of 'ArrayList' of 'HashMap', containing the record time and length information.

Moreover, there is a menu button provided to upload all the locally saved ECG files to cloud database, which is done by starting the Updata Service.

2.3.3.7 DataManageList Activity

The search results of ECG history records are listed in DataManageList Activity grouped by date, as shown in Figure 2.21. Results are displayed in form of 'ExpandableList', which is composed of 'GroupView' and 'ChildView'. The 'GroupView' here shows the date and number of items in the group, and it will be expanded to show its 'ChildView' after clicked. Each 'ChildView' contains one image and two texts, showing the time and length information of the record. The image is for distinguishing whether the record has been downloaded.

Two 'ContextMenu' options are added to each 'ChildView' item, and the 'ContextMenu' is popped up when user long presses the 'ChildView' item. The first option is 'Show graph (first 60 seconds)'. This will open DataManagePlot Activity and plot the selected ECG history record in it, and if the record has not been downloaded, the app will first download it to the local from the cloud database, saved as '.bin' files. However, the length of a record can be very long, and it may take the phone too many resources to draw it out. Therefore, we decided to just download and draw the first 60 seconds of the record, and if the length is less than that, we do the all.

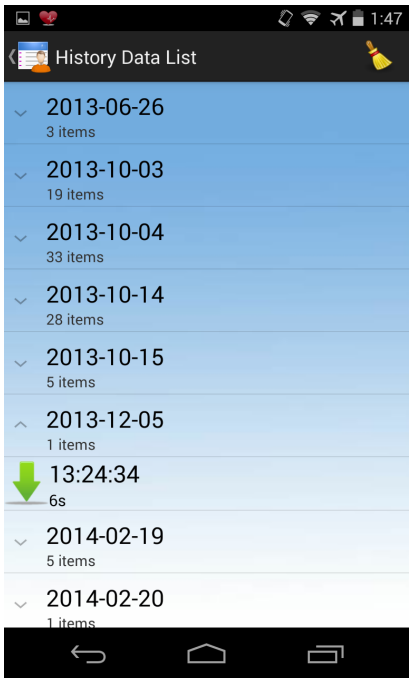

Figure 2.21: DataManageList Activity

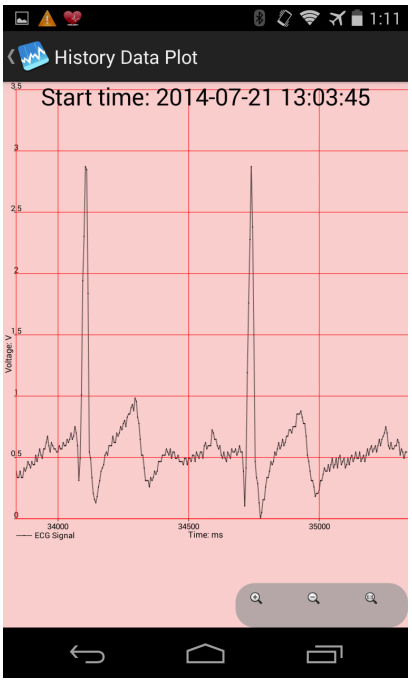

Figure 2.22: DataManagePlot Activity

The second option is 'Delete', which will delete the record from the cloud database and also the downloaded file if exists.

In addition, one menu button is added to clear all the downloaded record files, in case user needs space on their phone.

2.3.3.8 DataManagePlot Activity

As presented in Figure 2.22, the selected history ECG record is plotted in DataManagePlot Activity by parsing the local downloaded record file. The start date and time is printed on the top of the ECG signal. If the user logs in the app through Facebook or Google+ account, then the current ECG signal shown on the screen can be shared on the social networks through the menu button. However, the button is not visible to normal users.

2.3.3.9 ProfileSetting Fragment

The user profile information, such as email address and phone number, is stored in the cloud database, can be updated in ProfileSetting Fragment, as shown in Figure 2.23. During the initialization of this Fragment, current profile settings are downloaded from the cloud. EditText fields are used here for email address, phone number,

name and password settings. Specially for the 'E-mail Address' field, the 'inputType' attribute is set to 'textEmailAddress'; for the 'Phone' field, it is set to 'phone'; and for the three 'Password' fields, the attributes are 'textPassword'. User can update their profile settings by clicking the 'Update' button, and the action result will be shown in form of 'Toast'. The 'Refresh' menu button is for getting the latest profile settings from the cloud database. There is also another hidden menu button named 'User ID' for getting the user ID, corresponding to the 'userid' column in the database.

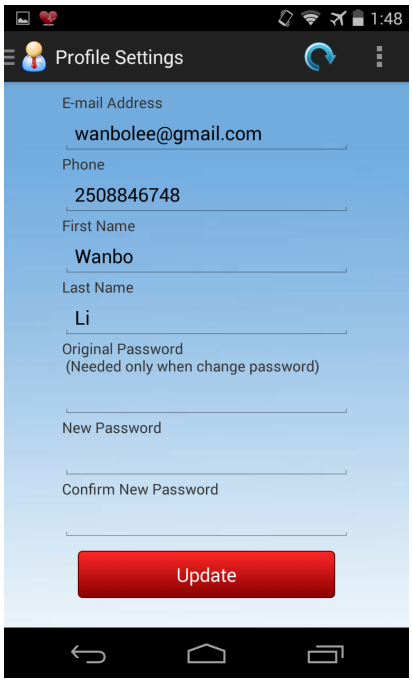

Figure 2.23: ProfileSetting Fragment

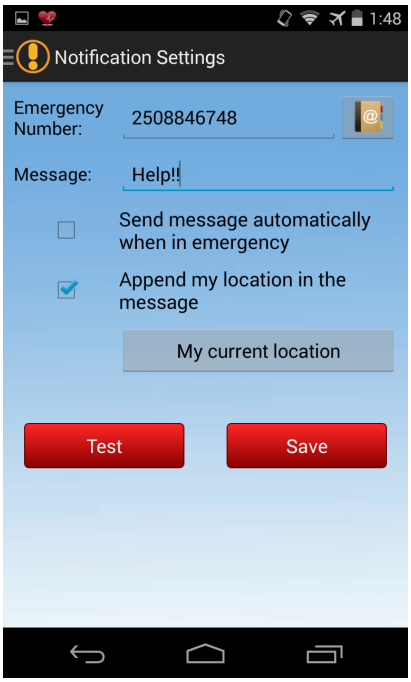

Figure 2.24: Notification settings

2.3.3.10 NotificationSetting Fragment

The UI layout of NotificationSetting Fragment is given in Figure 2.24. The notification function is designed to send a SMS message to a preset phone number when the user heart rate is lower than the preset value. 'Emergency Number' can be typed in or chosen from the contacts in the phone by clicking the small contact book button. The message body can also be edited here, and user can choose whether to 'Send message automatically when in emergency' and 'Append my location in the message' using the 'CheckBox'. If user wants to see their current location, they can press the 'My current location' button which will open a map app to show it. Two buttons

are placed at the very bottom, one is for sending test SMS message with the current settings, the other is to save the updated settings in cloud database.

2.3.3.11 GeneralSetting Fragment

The GeneralSetting Fragment mainly includes two parts, the upper part is for some system function settings, and the lower part is for app related settings, as shown in Figure 2.25. In the upper part, 'Switch' views are used to turn on/off the Bluetooth, WiFi or auto-rotate screen functions. And 'Setting' buttons are available for WiFi and location services functions to change settings in the setting app provided by the system. The app mode can be switched in the lower part, between normal and Compressed Sensing mode. Bluetooth is turned off when exiting the app by default, which is also changeable. The length of the ECG data saving period can be set in unit of minute, and the low heart rate threshold bpm can be also set here. The 'Save in cloud' button is for saving the changes user made in the lower part in the cloud database. However, the user can still save the settings locally by using the two 'Set' buttons.

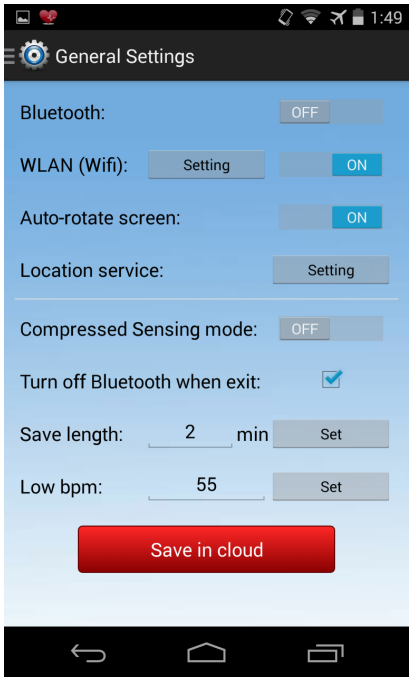

Figure 2.25: GeneralSetting Fragment

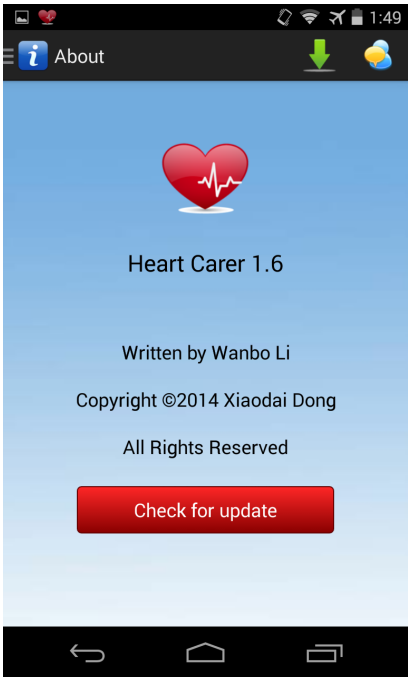

Figure 2.26: About Fragment

2.3.3.12 About Fragment

The About Fragment, as presented in Figure 2.26, is used for displaying the basic information of the app, for example the app name and version. The author and copyright information are also mentioned here. The button at the bottom is for checking new versions of the app, and if a new version exists, users will be asked if they want to download it. The first menu button is used to download the APK file of the app, and the other one is for entering Feedback Activity to give feedback to the app developers.

2.3.3.13 Feedback Activity

The UI of Feedback Activity is quite brief, just having a 'EditText' field and a button, and it's designed for the user to give feedback about the app to help developers improve the application. The UI design is given in Figure 2.27.

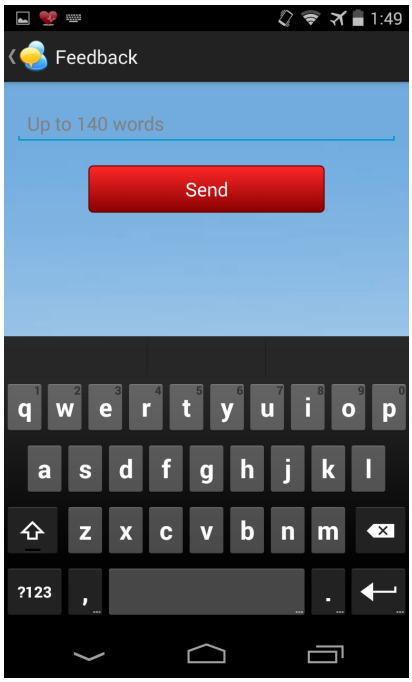

Figure 2.27: Feedback Activity

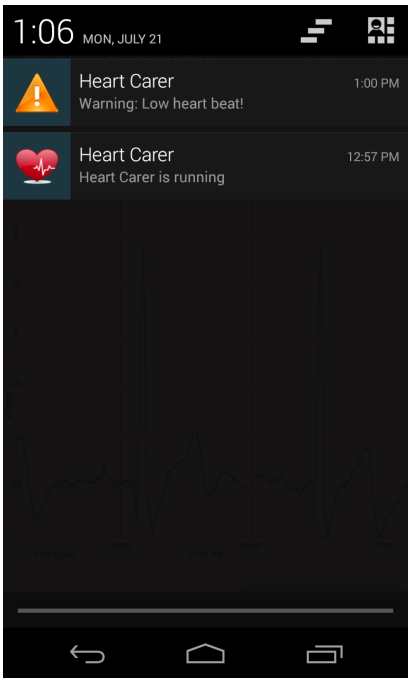

Figure 2.28: Notifications in status bar

2.3.4 Services

Although a Service does not have user interface, it still plays a significant role in the Android application, especially in the proposed app. There are two Services in the

proposed application, Ble Service and Updata Service, and the details of them are introduced in below sections.

2.3.4.1 Ble Service

The Ble Service is the most important component in the whole application. It is in charge of the BLE connection, including connecting to and disconnecting from the ECG sensor, receiving ECG data and controlling the Notification function of the sensor. Its responsibilities also contain parsing and saving the incoming ECG data, sending alarm when heart rate is low, and updating the heart rate value in the Widget. The reason of not doing these jobs in an Activity but in a Service is stated as below.

2.3.4.1.1 Reason for Using Service

When the ECG system is in working state, the connection between Android phone and ECG sensor needs to be always on. However, user may run the app in the background, and then the Android system will kill it if memory is needed by other apps. A Service bound to an Activity is less likely to be killed, and if the Service is declared to run in the foreground, it will almost never be killed. In the proposed application, the Ble Service is bound to Main Activity, and it is made to run in the foreground with an ongoing notification on the status bar, as shown in Figure 2.28 with 'Heart Carer is running' text.

2.3.4.1.2 Inter-object Communication

There are several public static variables used to indicate the status of functions, such as Notification function on ECG sensor or BLE connection status, and they can also be read or changed by other components like Hrm Fragment. However, the Ble Service mainly communicates with other components (mostly Hrm Fragment through Main Activity) through 'Messenger', as introduced in the Main Activity section. The 'Messenger' sends and receives 'Message' which includes a 'what' field to identify itself. For the incoming 'Message' from Hrm Fragment to Ble Service, the relationship between the value in 'what' field and the corresponding command is designed as shown in Table 2.2.

Value in 'what' field	Command
0	Register ActivityMessenger
1	Connect to selected BLE device
2	Disconnect from BLE sensor
3	Toggle the ECG sensor's Notification function
4	Toggle the 'saving ECG data to file' function

Table 2.2: 'Message' from Hrm Fragment to Ble Service

Same for the 'Message' received by Hrm Fragment from Ble Service, the relationship is given in Table 2.3.

Value in 'what' field	Command or status update
0	Plot real-time ECG signal with given value
1	ECG sensor is connected
2	ECG sensor is disconnected
3	Saving ECG data to file has started
4	Saving ECG data to file has stopped
5	Update heart rate value
6	Reinitialize all views

Table 2.3: 'Message' from Ble Service to Hrm Fragment

2.3.4.1.3 BLE Connection

The Ble Service is developed based on the Android BLE API which was introduced in Android 4.3 (API Level 18). Several major classes provided by the BLE API are used here for BLE connection, including BluetoothDevice, BluetoothGatt, BluetoothGattCharacteristic and BluetoothGattDescriptor. The functions of these classes can be deduced from the class name. For example, the BluetoothDevice class represents a remote Bluetooth device. And the BluetoothGatt API is for the Bluetooth GATT Profile, which provides Bluetooth GATT functionality to enable communication with BLE devices.

BluetoothGattCallback class is also used in Ble Service for receiving information from the BLE peripheral device. 'onConnectionStateChange' callback method gets called when BLE connection state changes. 'onServicesDiscovered' function is invoked when BLE services are discovered by the phone. And 'onCharacteristicChanged' is triggered as a result of a remote characteristic notification from the ECG sensor. The

ECG data is contained in the incoming parameter of 'BluetoothGattCharacteristic' type, and the data is enclosed in a byte array.

2.3.4.1.4 ECG Data Processing

For the convenience of saving ECG data to file and calculating heart rate, two buffers are created to temporarily store the received ECG data. As the ECG sampling frequency is 1 kHz and the resolution is 8-bit, one byte contains one measurement value, and both buffers are designed to be byte array of size 5000, i.e. 5 seconds of ECG data which is sufficient for the heart rate calculation. ECG data is processed byte by byte, and the flow chart of processing one byte of ECG data with the two buffers is illustrated in Figure 2.29. The 'pointer' in the chart represents the position in the current buffer where the incoming ECG data byte will be saved.

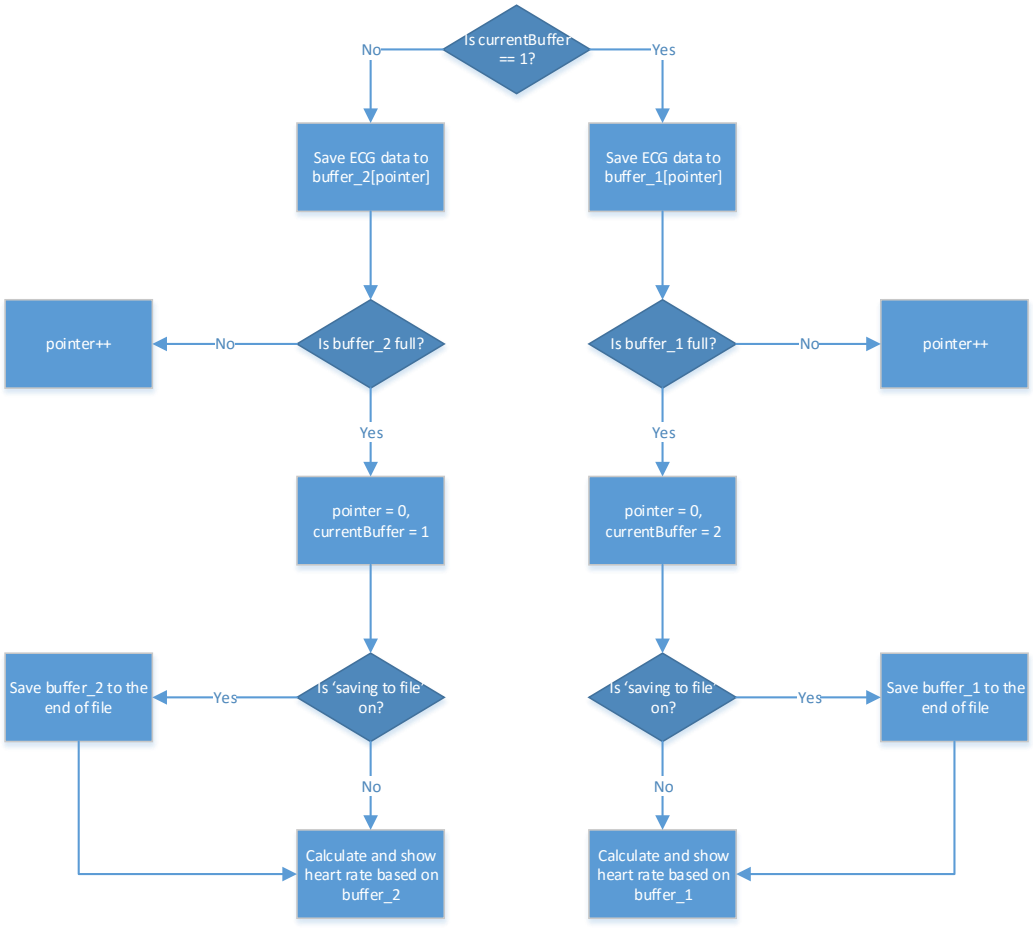

Figure 2.29: Process ECG data with two buffers

The ECG data file is first saved under '/sdcard/Heart Carer Data/userid/Cache' directory named by the start time, and the file type is '.bin'. As the saving length for each ECG data file can be changed by the user, a timer is used to control the saving process. When the user-defined time is up, the saved file is moved to '*/Saved for upload' folder, and a new cache file is created under '*/Cache' directory for the next saving cycle. The timer is also reset. If the connection is down within a saving period, the file is still saved automatically, and the timer is disabled.

Not every byte of ECG data is plotted in the app. Considering the resources it takes to draw ECG signal on the screen, we decide to just plot every 5 ECG measurement values in real time. The actual results on the testing Android phone turn to be acceptable and actually very good.

If the calculated heart rate is lower than the user-defined threshold, a notification will be generated and shown on the status bar which locates at the very top of the screen, together with a 3-second vibration. The appearance of the generated notification with a warning text is given in Figure 2.28. Moreover, if a user turns on the 'Send message automatically when in emergency' function, an SMS message will be sent to the pre-saved phone number. And if the 'append location' feature is open, the user's current location will be added to the SMS message.

2.3.4.2 Updata Service

The responsibility of Updata Service is uploading all the locally saved ECG data files under '*/Saved for upload' folder to the cloud database. The Java class 'UpdataService' extends the 'IntentService' class, while 'BleService' extends the 'Service' class. 'IntentService' is a base class for Services that handles asynchronous requests (expressed as Intents) on demand. 'IntentService' will receive the Intents, launch a worker thread, and stop the service as appropriate.

The job of uploading files is done inside 'onHandleIntent' function. Firstly, all the files of '.bin' type are selected using 'FilenameFilter'. Then all the nonempty files are uploaded to the cloud database through the web service. The local files will be deleted if they are uploaded successfully. In the end, Updata Service will end itself after all the local ECG data files are uploaded to the cloud database.

2.3.5 Widget and more

App Widgets are miniature application views that can be embedded in other applications (such as the Home screen) and receive periodic updates [57]. A Widget is created in the proposed application to show the current heart rate value on the Home screen, as shown in Figure 2.30. The UI layout includes an 'ImageView' showing the app icon, a 'TextView' for the heart rate value, and another 'TextView' giving bpm as the heart rate unit. The codes of the Widget is in 'MyAppWidgetProvider', which extends 'AppWidgetProvider' class provided by Android API.

Figure 2.30: Heart rate widget

Figure 2.31: Application icon

Google Nexus 4 phone is used as the development platform for this application, and AVD (Android Virtual Device) is another option for Android development if a real device is not available, but AVD is much slower than an actual Android phone.

As Android requires all apps to be digitally signed with a certificate before they can be installed, a 'keystore' is created and used to sign the proposed application. The Android Keystore system can store cryptographic keys in a container to make it more difficult to extract from the device. Moreover, the ProGuard is enabled in 'project.properties' file to shrink and obfuscate codes.

The icon of the proposed Android app is given in Figure 2.31.

2.4 Web Service and Cloud Database

The cloud database in the proposed system is for storing the ECG records, user profiles and preference settings. And the web service serves as a bridge between Android app and cloud database by providing some APIs, for example 'login' API for checking the login credential. The web service is running on the front end of a server located in our lab, and the cloud database is on the back end of the same server.

2.4.1 Web Service

The proposed web service is developed under Jersey Framework using Java language. It is running on a Nginx-Tomcat web server and it exposes APIs to help Android app communicate with the cloud database.

2.4.1.1 Web Service Introduction

A web service is a software system designed to support interoperable machine-to-machine interaction over the World Wide Web [58]. In practice, the Web service typically provides an object-oriented Web-based interface to a database server, utilized for example by a web page, or by a mobile application, that provides a user interface to the end user. Web services can be published, found, and used on the Web, and they are self-contained and self-describing.

On a technical level, web services can be implemented in various ways. Generally, web services can be classified as "big" web services and RESTful web services [59]. **"Big" web services** use XML messages that follow the SOAP (Simple Object Access Protocol) standard, which is a format based on XML for sending and receiving messages. Such systems often contain a machine-readable description of the operations offered by the service, written in WSDL (Web Services Description Language), an XML language for defining interfaces syntactically. The SOAP message format and the WSDL interface definition language have gained widespread adoption in traditional enterprises.

RESTful web services are based on the way how the web works [60]. The World Wide Web is based on an architectural style called REST - Representational State Transfer. Web services following this architectural style are said to be RESTful Web services. REST-style architecture follows this concept and consists of clients and servers. Clients initiate requests to servers; servers process requests and return appropriate responses. Requests and responses are built around the transfer of representations of resources which are identified by URI (Uniform Resource Identifier).

RESTful web services are better integrated with HTTP (Hypertext Transfer Protocol) than SOAP-based services, and do not require XML SOAP messages or WSDL service definitions. Unlike SOAP-based web services, RESTful is not dependent on any protocol and it makes use of existing well-known standards, such as HTTP, URI, JSON (JavaScript Object Notation) and XML. RESTful web service is also more lightweight and easier to build. Based on the above advantages, RESTful web service

is chosen to be implemented in the system.

2.4.1.2 HTTP Methods

RESTful web services typically communicate over HTTP with the same HTTP verbs (GET, POST, PUT, DELETE, etc.) that web browsers use to retrieve web pages from and send data to remote servers. The definitions of the four most commonly used HTTP methods are given as below [61]:

- **GET**: Retrieve whatever information is identified by the Request-URI.
- **POST**: Request that the server accept the entity enclosed in the request as a new subordinate of the web resource identified by the URI.
- **PUT**: Request that the enclosed entity be stored under the supplied Request-URI.
- **DELETE**: Request that the origin server delete the resource identified by the Request-URI.

Among these four methods, GET and POST are applied in the proposed web service.

2.4.1.3 Jersey Framework

In Java EE 6, JAX-RS provides the functionality for RESTful web services. Jersey framework is the production-ready reference implementation for the JAX-RS specification. Jersey also provides it's own API that extend the JAX-RS toolkit with additional features and utilities to further simplify RESTful service and client development [62]. Jersey framework version 1.17.1 is used in the proposed web service.

The Jersey API can significantly streamline the development of a RESTful web service, and much of its ease comes from the use of annotations. Below are some annotations used in the proposed web service:

- @Context: Generally used to obtain contextual Java types related to the request or response [63].
- @Path: Relative URI path for the current Java method.
- @Get: The method will be called by a HTTP GET request.
- @POST: The method will be called by a HTTP POST request.

- @Consumes: Specify the MIME (Multipurpose Internet Mail Extensions) media types of representations that can be consumed by a resource.
- @Produces: Specify the MIME media types of representations a resource can produce and send back to the client.

In the proposed web service, a global url-pattern is created in the 'web.xml' file to make any request that goes to '/rest/' attempt to be mapped to the appropriate methods.

2.4.1.4 URIs and APIs

All the proposed URIs and corresponding APIs in the proposed web service are presented in Table 2.4, and they are designed to meet all the current requirements of the system. The '@Consumes' column shows the type of data that the API consumes, and the '@Produces' column means the data type that the API returns to the client. The HTTP method of '/', 'getcurversion' and 'getapk' URIs is GET, and for the other URIs is POST. As a HTTP GET method does not consume any entity, the value of '@Consumes' is marked 'N/A' in the table for GET methods. The 'form-data' type may contain files, therefore it is used for uploading ECG record files and app APK file. The 'octet-stream' type indicates that a body contains arbitrary binary data, and it is used for downloading app APK file in the proposed web service. JDBC (Java Database Connectivity) API is utilized as the connection bridge between the web service and database.

URI	@Consumes	@Produces	Usage Description
/	N/A	String	Display the web service status
/login	JSON	JSON	Log in with existing account
/signup	JSON	JSON	Sign up a new account
/updata	form-data	String	Upload ECG record file
/downdata	JSON	JSON	Download certain ECG record
/downuserinfo	String	JSON	Get user profile settings
/upuserinfo	JSON	String	Update user profile settings
/getdatalist	JSON	JSON	Get ECG record list
/deletedata	JSON	String	Delete certain ECG record
/upsms	JSON	String	Update notification settings
/fblogin	JSON	JSON	Log in with Facebook account
/gplogin	JSON	JSON	Log in with Google+ account
/upgeneralsetting	JSON	String	Update general settings
/uploadapk	form-data	String	Upload app APK file
/getcurversion	N/A	String	Get the current app version
/getapk	N/A	octet-stream	Download APK file of the app
/sendfeedback	JSON	String	Send feedback about the app

Table 2.4: Proposed URIs and APIs

2.4.1.5 Web Server

The proposed web service is running on the Apache Tomcat server, an open-source web server developed by the Apache Software Foundation (ASF). Nginx is also used here as a reverse proxy server. It receives requests from clients and forwards them to the Tomcat server, and same reversely. The Nginx server also behaves as a load balancer and a HTTP cache. The default ports of Tomcat and Nginx are 8080 and 80, respectively.

2.4.2 Cloud Database

The proposed cloud database is deployed on the same server with the web service. MySQL database is used, and four tables are created to store ECG records and other user information.

2.4.2.1 MySQL Database

MySQL is a fast, easy-to-use RDBMS (relational database management system) being used for many small and big businesses. MySQL is one of the most widely used RDBMS in the world, and it is now owned by Oracle Corporation. MySQL works on many operating systems and with many languages including C, C++, Java, etc. It also works very quickly and works well even with large data sets. MySQL supports command line, and also GUI tool on Windows named 'MySQL Workbench'.

2.4.2.2 Tables

SQL (Structured Query Language) is used to create tables in MySQL database. All the tables in the proposed database are listed as below, together with the name and type of the contained columns.

- 'userinfo' table: userid(int unsigned, primary key), username(varchar), facebookid(varchar), googleplusid(varchar), userpassword(varchar), email(varchar), firstname(varchar), lastname(varchar), phone(varchar), emergencynum(varchar), emergencymes(varchar), ifsendsms(boolean), ifappendloc(boolean), ifcsmode(boolean), ifturnoffbt(boolean), savelength(int unsigned), lowbpm(int unsigned), lastdevice(varchar), lastip(varchar), lasttime(datetime), version(varchar), imei(varchar)
- 'ecgdatainfo' table: dataid(int unsigned, primary key), userid(int), starttime(datetime), length(int unsigned), ifdone(boolean), ifcs(boolean)
- 'ecgdata' table: dataid(int unsigned, foreign key), datacontent(mediumblob)
- 'userfeedback' table: feedbackid(int unsigned, primary key), userid(int), feedback(varchar)

The 'userinfo' table stores the user profile and preference settings. The information of the ECG records is saved in 'ecgdatainfo' table, and the data content of the records is saved in a separated 'ecgdata' table to increase the query speed of ECG records. And the 'dataid' column in 'ecgdata' table is a 'foreign key' referencing the 'dataid' in 'ecgdatainfo' table, and the items in both table are one-to-one matched. The ECG data content is stored in 'datacontent' column with a 'mediumblob' data type. The storage limit of a 'mediumblob' is 2^{24} bytes, which is enough for ECG data of about 4.66 hours. If monitoring length is larger than this value, the ECG data will be saved

in separated records. The 'userfeedback' table stores the feedbacks about the app from the user.

2.5 Summary

In this chapter, we have proposed a power-efficient wireless ECG personal health system. The ECG sensor, which is based on TI CC2540 chip, uses Bluetooth Low Energy protocol to send the collected ECG data to an Android smartphone. ECG data is able to be viewed and stored on the smartphone, and also to be uploaded to or downloaded from the cloud server by the user. Moreover, the system can send an alarm when a low heart beat is detected.

Chapter 3

Compressed Sensing on ECG Signal

Conventional approaches to sample signals follow Nyquist-Shannon sampling theorem [64]: If a signal with a bandwidth of Ω is sampled at the rate of 2Ω samples per second (the Nyquist rate), the signal can be reconstructed reliably and without errors. However, the theory of Compressed Sensing (CS), also known as compressive sampling, asserts that one can recover certain signals and images from a smaller number of samples than required by the Nyquist rate. CS relies on two principles: sparsity, which relates to the signals of interest, and incoherence, which pertains to the sensing modality.

In recent years, Compressed Sensing has been applied to ECG signals in some research works [44, 65]. In [44], it was shown that the redundancy between consecutive heartbeats implies a high fraction of common support between adjacent heartbeats. They also used distributed compressed sensing to this common support between heartbeat samples. And in [65], Mishra *et al.* implemented Compressed Sensing on ECG signals with different wavelet families, and they found that the reverse biorthogonal wavelet family has the best performance. In comparison to the conventional ECG compression algorithms, Compressed Sensing transfers the computational burden from the encoder to the decoder, and thus allows simpler implementation and longer lifetime for the ECG sensor.

Compressed Sensing has also been applied to Wireless Body Area Networks (WBANs). Dixon *et al.* [66] studied several design considerations for CS acquisition systems for ECG biosignals. Using 1-bit Bernoulli measurement matrix result in high compres-

sion ratios in their experiments. And in [67], Zhang *et al.* applied CS to fetal ECG telemonitoring via WBANs. Block sparse Bayesian learning framework was used to compress/reconstruct the ECG signals. In the proposed Wireless ECG System, the battery lifetime of both the ECG sensor and smartphone is a big concern. The introduction of Compressed Sensing can help reduce the amount of data to be transmitted from the sensor to the smartphone and then to the cloud server. And as the radio frequency (RF) transmission and reception takes a large proportion of the whole power consumption of both the ECG sensor and smartphone, the CS technology can save the energy and extend the battery life. This has a distinct advantage to conventional ECG compression methods which are too complicated for an ECG sensor but can only be implemented on an smartphone. In those cases, the samples transmitted from the ECG sensor are not compressed and hence consuming more power for both the ECG sensor and the smartphone. Furthermore, much less storage space is required on the smartphone due to the reduced number of samples in CS.

In this chapter, a novel sparse sensing matrix is proposed to sample real-time ECG signal with Compressed Sensing, together with a modified CoSaMP reconstruction algorithm. Moreover, a new variable-length algorithm is proposed to compress the difference values of measurement results. In addition, the CS-based algorithm is implemented on the proposed Wireless ECG System to reduce the power consumption of the ECG sensor and the smartphone by reducing the data to be transmitted.

The rest of the chapter is organized as follows. Section 3.1 introduces the system model of the proposed CS based algorithm. Section 3.2 proposes the algorithmic approach to apply Compressed Sensing to ECG signals. The implementation of the proposed algorithm on the Wireless ECG System is presented in Section 3.3. Finally, Section 3.4 concludes this chapter.

3.1 System Model

The original ECG signal $\mathbf{x}$ is sampled by the proposed sparse sensing matrix $\mathbf{\Phi}$, and then the projection vector $\mathbf{y}$ is obtained. Afterwards, the ECG signal is reconstructed from $\mathbf{y}$ using a modified CoSaMP algorithm. The whole process is briefly illustrated in Figure 3.1.

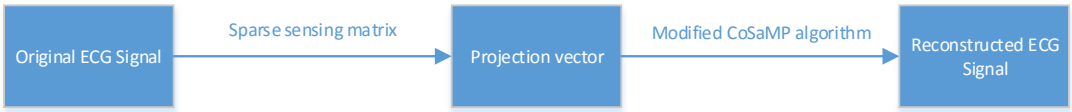

Figure 3.1: Compressed Sensing on ECG Signal

3.2 Algorithmic Approach

A new sparse sensing matrix is proposed to apply Compressed Sensing to the ECG signal. The matrix is designed for real-time sampling to be implemented by an embedded system. A variable-length compression algorithm is also designed to further reduce the transmitted data. The ECG signal is reconstructed using a modified CoSaMP algorithm. The analog-to-digital (AD) sampling rate is set to be 500 Hz with 8-bit resolution, and experiments are carried out over a period of ECG signal collected by the proposed Wireless ECG System. The sampling rate is reduced to 500 Hz due to the space limitation of TI CC2540 for storing the sensing matrix. This AD rate is more than sufficient for an ECG signal which is sampled at 250 Hz in many commercial ECG devices. Compressed sensing is then operated on these digital samples and after CS, the data are transmitted to the smartphone.

In order to quantify the compression performance and assess the quality of reconstructed ECG signals, two most widely used performance indicators are applied: the compression ratio (CR) and percentage root-mean-square difference (PRD) [68]. The CR is computed as follows:

$$\mathrm{CR} = \frac{b_{orig} - b_{comp}}{b_{orig}} \times 100\% \tag{3.1}$$

where b_{orig} and b_{comp} represent the number of bits required for the original and compressed signals, respectively. The PRD quantifies the percent error between the original signal vector $\mathbf{x}$ and the reconstructed $\tilde{\mathbf{x}}$:

$$\mathrm{PRD} = \frac{\|\mathbf{x} - \tilde{\mathbf{x}}\|_2}{\|\mathbf{x}\|_2} \times 100\% \tag{3.2}$$

where the operator $\|\cdot\|_2$ denotes the Euclidean 2-norm.

Table 3.1 gives the resulting different quality classes and corresponding PRD, based on the work of Zigel *et al.* on the weighted diagnostic measure for ECG signal compression [69].

PRD	**Reconstructed Signal Quality**
$0 \sim 2\%$	"Very good" quality
$2 \sim 9\%$	"Very good" or "good" quality
$\geq 9\%$	Not possible to determine the quality group

Table 3.1: Quality class and corresponding PRD

3.2.1 Sensing Matrix

As the ECG signal $\mathbf{x}$ is sparse in wavelet domain, the matrix $\mathbf{\Psi}$ is set to be a Daubechies wavelet matrix, db4 particularly. Figure 3.2 shows the sample ECG signal $\mathbf{x}$ in wavelet domain after Inverse Discrete Wavelet Transform (IDWT). The IDWT is performed with a threshold leaving only 226 of the total 2048 samples non-zero, and the PRD of the reconstructed ECG signal after Discrete Wavelet Transform (DWT) is 2.13%. Considering real-time transmission and the heartbeat cycle, the window length is fixed to be $2^{11} = 2048$ samples, i.e., 4.096 seconds as the sampling rate is 500 Hz, where 11 is the wavelet decomposition level.

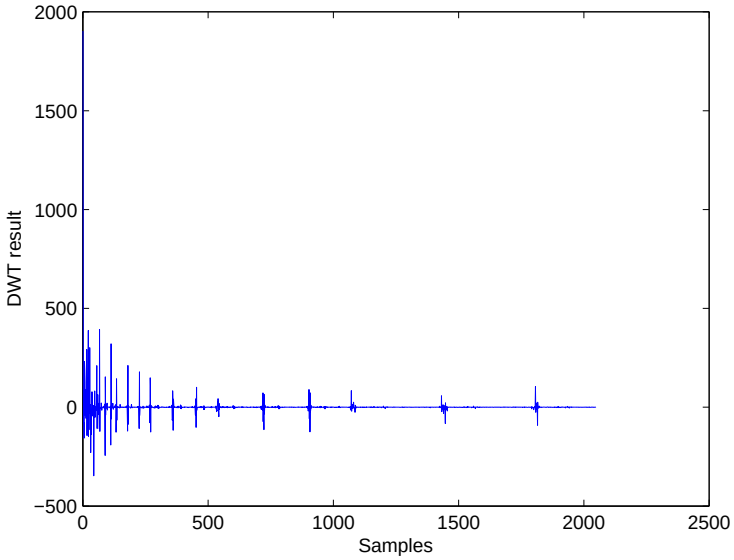

Figure 3.2: ECG signal in wavelet domain

There are many kinds of well known matrices already qualified for sensing matrix $\mathbf{\Phi}$ in Compressed Sensing, such as Gaussian random matrix which satisfies RIP property and leads to a good reconstruction quality. However, these kinds of matrices require too much computation and are not suitable for real-time and power-efficient sampling. Here we propose an innovative matrix to implement CS on the BLE sensor in Wireless ECG System. To decrease the execution time and space needed to store sensing matrix on sensor, we explore sparse binary sensing matrix with each column having exactly k nonzero entries equal to 1. The RIP property is not valid for this kind of matrix. However, it satisfies a different form of RIP, called RIP-p [70] as follows. If for any k-sparse vector $\mathbf{s}$ and $p \approx 1$, we have

$$(1-\delta)\|\mathbf{s}\|_p \leq \|\mathbf{\Phi\Psi s}\|_p \leq (1+\delta)\|\mathbf{s}\|_p \tag{3.3}$$

where $\mathbf{\Phi}$ is an $M \times N$ matrix, then the RIP-p property is satisfied, and perfect reconstruction can be achieved by a linear program.

As Compressed Sensing is performed on every 2048 samples of the incoming ECG signal, N is 2048 here. M is set to be 1024, which is a rule-of-thumb value decided by experiment results as illustrated in Figure 3.3, so the compression ratio is 1/2. This number 1024 is slightly more than 4 times the sparsity (226), which is reasonable. It can be seen that the PRD decreases while M is increasing, and it decreases more

quickly when M is small. Therefore, 1024 is a tradeoff between low PRD and small M. To further reduce the chip memory consumption for sensing matrix and computing results, a novel sparse binary sensing matrix is proposed here for sampling real-time ECG signals. The sensing matrix $\mathbf{\Phi}$ is constructed by repeating the template matrix $\mathbf{\Lambda}$ four times:

$$\mathbf{\Phi} = \begin{bmatrix} \mathbf{\Lambda} & 0 & 0 & 0 \\ 0 & \mathbf{\Lambda} & 0 & 0 \\ 0 & 0 & \mathbf{\Lambda} & 0 \\ 0 & 0 & 0 & \mathbf{\Lambda} \end{bmatrix}. \tag{3.4}$$

The size of $\mathbf{\Lambda}$ is 256×512, and it is designed to have only ones randomly with exactly 12 ones on each row and 6 ones on each column. Name the number of ones on each row in sensing matrix $\mathbf{\Phi}$ to be d. The value 12 is also a tradeoff, between low PRD and small value of d, as shown in Figure 3.4.

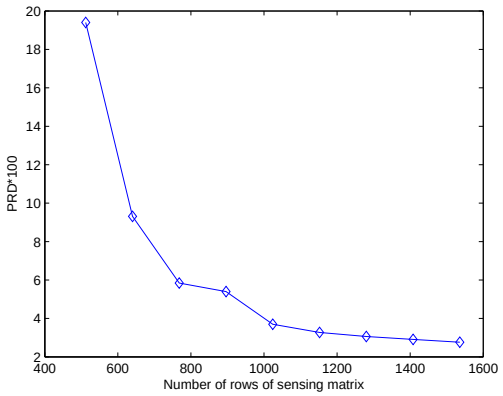

Figure 3.3: M vs. PRD

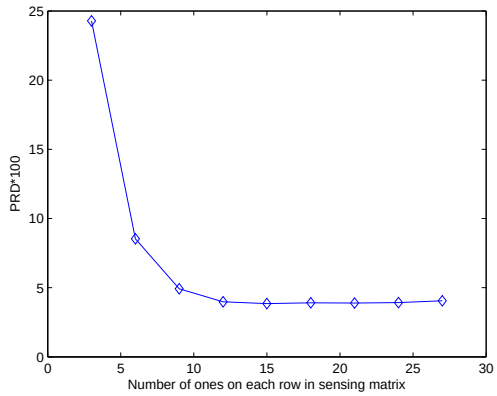

Figure 3.4: d vs. PRD

With the diagonal matrix structure design of sensing matrix $\mathbf{\Phi}$, the calculation can be performed every 512 samples, i.e. 1.024 seconds. And 256 values are obtained as one quarter of the final $\mathbf{y}$ for every sampling period.

3.2.2 Data Compression

To further increase the compression ratio of the compressed ECG signal, the difference values of the measurement results are transmitted instead of the original values. Fig. 3.5 shows that the distribution of the difference values concentrates around zero, resulting in potentials of further compression. Compared to using the sensing matrix proposed in [71] which only has exact number of ones on each column, the difference

values of the measurement results are more concentrated with our proposed sensing matrix, meaning that higher CR can be achieved.

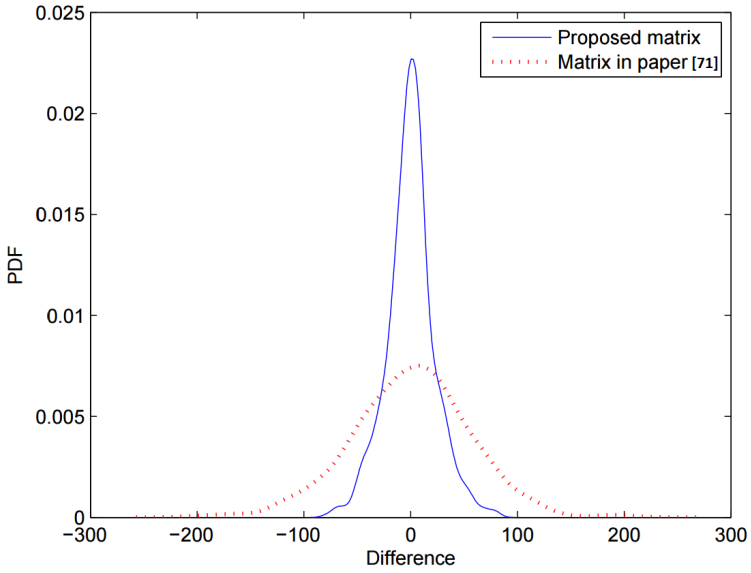

Figure 3.5: PDF of the difference values

Therefore, a new variable-length compression algorithm with 2-bit flag is proposed, as shown in Table 3.2. The main idea of this algorithm is using less bits to represent the difference values which are closer to zero. The whole value range of this algorithm, $-156 \sim 155$, can cover all the possible difference values and provide a lossless compression.

2-bit flag	**Number of following bits**	**Value range**
00	3	-4 $\sim$ -1, 0 $\sim$ 3
01	4	-12 $\sim$ -5, 4 $\sim$ 11
10	5	-28 $\sim$ -13, 12 $\sim$ 27
11	8	-156 $\sim$ -29, 28 $\sim$ 155

Table 3.2: Proposed variable-length compression algorithm

3.2.3 Reconstruction Algorithm

There are some algorithms existing for performing the signal reconstruction problem. Some of these include:

- Convex Optimization:
  - Basis Pursuit (BP)
  - Basis Pursuit De-Noising (BPDN)
  - Second-Order Cone Programming

- Iterative Greedy Algorithms:
  - Matching Pursuit (MP)
  - Orthogonal Matching Pursuit (OMP)
  - Iterative Re-weighted Least Squares (IRLS)
  - Compressive Sampling Matching Pursuit (CoSaMP)

CoSaMP algorithm is utilized to recover compressed ECG signal in the proposed system. The reconstruction time and PRD results are given in the following section, together with comparison with using OMP and IRLS algorithms.

Let $\mathbf{y} = \mathbf{\Phi x} + \mathbf{e}$ be a vector of samples of an arbitrary signal $\mathbf{x}$ contaminated with arbitrary noise $\mathbf{e}$, where $\mathbf{\Phi}$ is the sensing matrix with restricted isometry constant $\delta_{2k} \leq c$. It has been proved [72] that the CoSaMP algorithm produces a k-sparse approximation $\mathbf{a}$ of $\mathbf{x}$ which satisfies

$$\|\mathbf{x} - \mathbf{a}\|_2 \leq \mathrm{C} \cdot \max \left\{ \eta, \frac{1}{\sqrt{k}} \|\mathbf{x} - \mathbf{x}_{k/2}\|_1 + \|\mathbf{e}\|_2 \right\} \tag{3.5}$$

where η denotes a given precision parameter, $\mathbf{x}_{k/2}$ represents a best $(k/2)$-sparse approximation to $\mathbf{x}$. The running time is $\mathrm{O}(\mathfrak{L} \cdot \log(\|\mathbf{x}\|_2/\eta))$, where $\mathfrak{L}$ bounds the cost of a matrix-vector multiply with $\mathbf{\Phi}$ or $\mathbf{\Phi}^*$, and the working storage is $\mathrm{O}(N)$.

Due to the particularity of the proposed sparse binary sensing matrix, the original CoSaMP algorithm is not able to recover the original ECG signal perfectly. Let $\mathbf{r}$ be the signal proxy and k be the sparsity level of the signal. In the proposed CoSaMP algorithm, the number of large components extracted from $\mathbf{r}$ is changed from $2k$ to $\frac{k}{8}$ because of the sparsity of the proposed sensing matrix. The entire algorithm is specified in Algorithm 1. For the sensing matrix $\mathbf{\Theta}$ of the 'Input', $\mathbf{\Theta} = \mathbf{\Phi\Psi}$.

Algorithm 1 Modified CoSaMP Algorithm

Input: Sensing matrix $\mathbf{\Theta}$ $(M \times N)$, noisy sample vector $\mathbf{y}$

Output: An k-sparse approximation $\mathbf{a}$ of the target signal

$k = \frac{N}{8}$ {Sparsity level}
$\mathbf{a}^0 = 0$ {Trivial initial approximation}
$\mathbf{v} = \mathbf{y}$ {Current samples equals to input samples}
$j = 0$
repeat
  $j = j + 1$
  $\mathbf{r} = \mathbf{\Theta}^* \mathbf{v}$ {Form signal proxy}
  $\Omega = \text{supp}(\mathbf{r}_{\frac{k}{8}})$ {Identify large components}
  $T = \Omega \cup \text{supp}(\mathbf{a}_k^{j-1})$ {Merge supports}
  $\mathbf{b}|_T = \mathbf{\Theta}_T^{\dagger} \mathbf{y}$ {Signal estimation by least-squares}
  $\mathbf{b}|_{T^C} = 0$
  $\mathbf{a}^k = \mathbf{b}_k$ {Prune to obtain next approximation}
  $\mathbf{v} = \mathbf{y} - \mathbf{\Theta}\mathbf{a}^j$ {Update current samples}
until $j = 15$
return $\mathbf{a}$

Notation: $\mathbf{\Phi}^*$ indicates the Hermitian transpose of $\mathbf{\Phi}$. $\mathbf{\Phi}^{\dagger}$ indicates the pseudo-inverse of $\mathbf{\Phi}$. T^C indicates the compliment of set T. $\mathbf{x}|_T$ indicates the vector $\mathbf{x}$ is restricted only the elements given in T. $\mathbf{x}_N$ indicates the best N-point support set of the vector $\mathbf{x}$, $\text{supp}(\mathbf{x})$.

3.2.4 Experiment Results

The first 2048 values of the sample ECG signal are plotted in Figure 3.6. The reconstructed ECG signal using the proposed Compressed Sensing method is given in Figure 3.7 with a PRD of 3.70%.

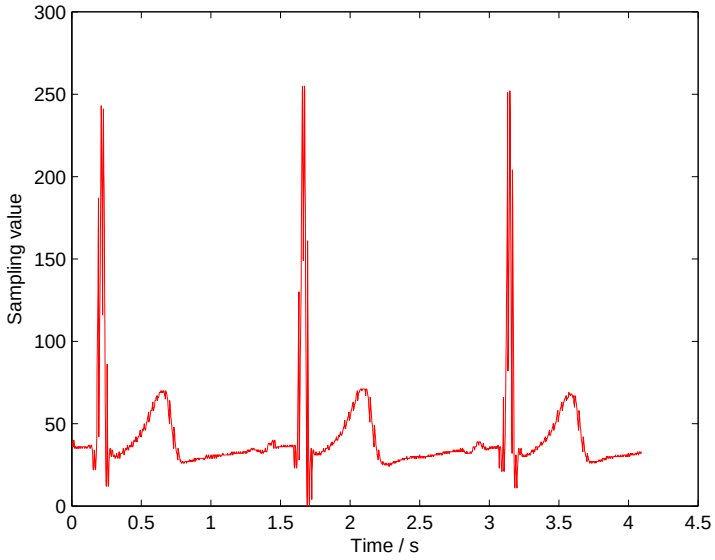

Figure 3.6: Original ECG signal

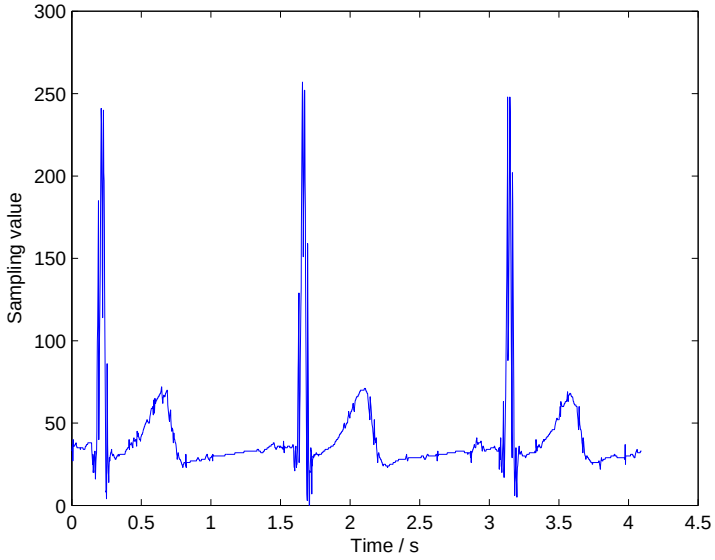

Figure 3.7: Reconstructed ECG signal

Considering the fluctuation of heart rate, ECG signals with half and twice heart rate of the above sample are constructed from the existing signals. The half heart rate ECG signal is constructed by stretching the first half (2.048 seconds) of the above sample ECG signal to a period of 4.096 seconds. And the sample ECG signal with

the following 4.096 seconds signal is squeezed to form the twice heart rate signal. The same Compressed Sensing algorithm is applied to the new signal. The PRD of the reconstructed ECG signal is 3.97% and 5.17%, respectively. The original and recovered ECG signals for twice heart rate are presented in Figure 3.8 and Figure 3.9.

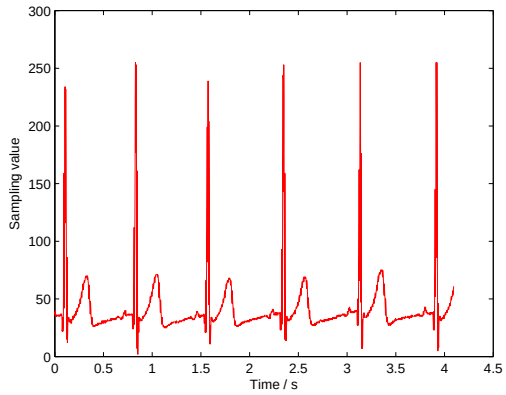

Figure 3.8: Original ECG signal (twice heart rate)

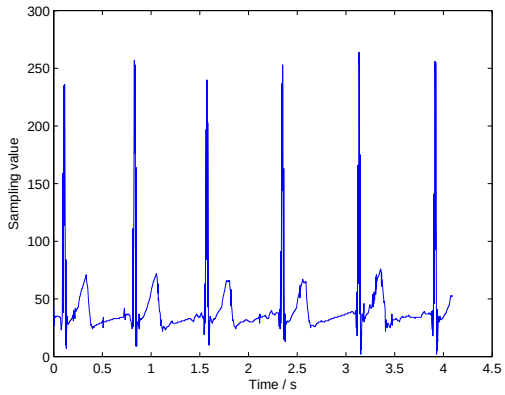

Figure 3.9: Reconstructed ECG signal (twice heart rate)

The time needed for the reconstruction of compressed sensed ECG signals using different algorithms is evaluated in Table 3.3. Two other reconstruction algorithms, OMP and IRLS, are introduced here as comparisons to the proposed modified CoSaMP algorithm. According to the results, with similar PRD, both OMP and IRLS need more than one second. But for the proposed CoSaMP algorithm, only 0.3711 seconds is needed, which is 71.40% less than OMP and 64.36% less than IRLS. The reconstruction is performed on the same computer using MATLAB (MATrix LABoratory) software.

Algorithm	**Reconstruction time in seconds**	**PRD×100**
Proposed CoSaMP	0.3711	3.70
OMP	1.2975	3.67
IRLS	1.0412	3.68

Table 3.3: Reconstruction time with different algorithms

Each ECG cycle consists of 5 waves: P, Q, R, S, T, corresponding to different phases of the heart activities, as shown in Figure 3.10. The P wave represents the normal atrium (upper heart chambers) depolarization. The QRS complex (one single

heart beat) corresponds to the depolarization of the right and left ventricles (lower heart chambers). The T wave represents the re-polarization (or recovery) of the ventricles.

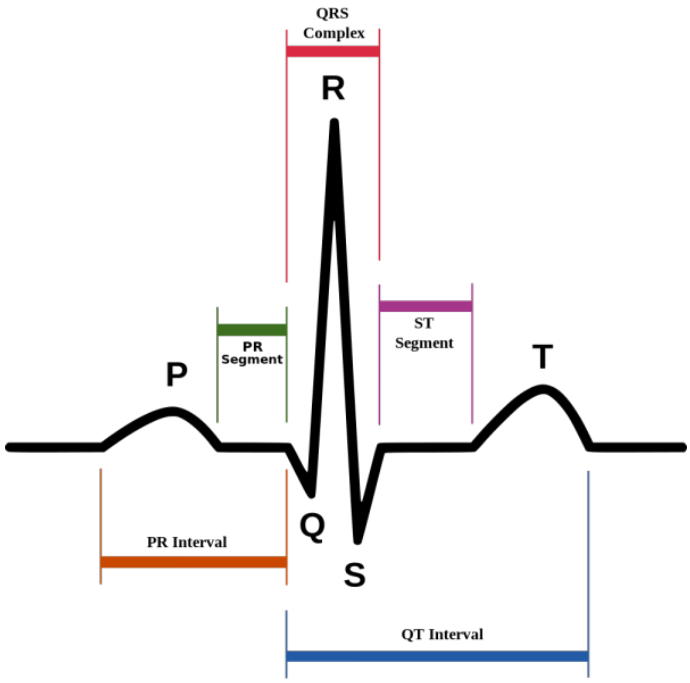

Figure 3.10: ECG Wave Components

The average PRD for P wave, QRS complex and T wave is illustrated in Table 3.4. The P wave, QRS complex and T wave are separated from the sample ECG signal using cross correlation algorithm [73]. As shown in the table, the PRD of QRS complex is less than P and T wave, i.e., the proposed Compressed Sensing algorithm performs better on QRS complex than the other two components.

	P Wave	**QRS Complex**	**T Wave**
PRD×100	4.79	2.80	4.08

Table 3.4: PRD of different ECG components

3.3 Implementation on Wireless ECG System

The support of the proposed Compressed Sensing algorithm is added to the Wireless ECG System, and the implementation is done on all the three system components as detailed in the following sections.

3.3.1 On ECG Sensor

The projection vector $\mathbf{y}$ of ECG signal $\mathbf{x}$ is calculated on the ECG sensor, and then transmitted to the Android application after processing and compressing. The 256×512 template matrix $\mathbf{\Lambda}$ of sensing matrix $\mathbf{\Phi}$ is stored in the memory of CC2540 chip, in the form of a 256×12 matrix containing the positions of ones. As previously mentioned, for every 1.024 seconds, the 512 ECG signal values are first saved in the memory, and then they are used to calculate a quarter of the projection vector $\mathbf{y}$ which has $\frac{1024}{4} = 256$ values. Because four groups of data are needed for signal reconstruction, the group number information is added to the transmitted packet.

3.3.2 On Android Application

The proposed Android application supports two work modes, normal mode and Compress Sensing mode. The two modes can be switched from the General Settings page. We tried to perform reconstruction on the Android phone, but due to the restrictions of computation power, the reconstruction process for one sampling cycle, i.e. 4.096 seconds (2048 samples), takes more than 10 seconds, and this is not acceptable for real-time ECG signal plotting. Therefore, the received data is only saved on the phone and uploaded to the web server without any processing. This also reduces the amount of data transmission from the smartphone to the server, which further saves the smartphone power consumption.

3.3.3 On Web Server

The ECG data sampled by Compressed Sensing is saved in database together with normal ECG data, and there is a 'ifcs' column in 'ecgdatainfo' table to distinguish the two types. As the reconstruction algorithm is written in MATLAB language, MATLAB Compiler SDK is introduced to convert MATLAB codes to Java codes.

3.3.4 Experiment Results

For the sample ECG signal in Figure 3.6 which has 2048 samples, as the sampling resolution is 8-bit, $2048 \times 8 = 16384$ bits will be transmitted in normal mode. However, when applying the proposed compressed sensing and data compression algorithm, only 7078 bits are sent to the Android phone, i.e. a CR of 56.8%.

3.4 Summary

In this chapter, we have proposed a new sparse sensing matrix and a modified CoSaMP reconstruction algorithm to apply Compressed Sensing to the Wireless ECG System. According to the experiment results, about 57% of the data to be transmitted by the ECG sensor is reduced comparing with when not using CS. And the reconstruction time of the sample ECG signal is saved 64% compared to using OMP or IRLS algorithm.

Chapter 4

Conclusions and Future Work

4.1 Conclusions

In this thesis, we have introduced a wireless ECG system with Bluetooth Low Energy and Compressed Sensing, including a portable ECG sensor, an Android application and a cloud server.

In Chapter 2, we have proposed a low-energy-cost wireless ECG monitor system. The ECG sensor is designed based on TI CC2540 chip, and the sampling rate is 1kHz with 8-bit resolution. The ECG signal is collected through three electrodes attached to human body, and then transmitted to Android phone using Bluetooth Low Energy. The Android application can plot the real-time ECG signal on the screen and save it on the phone at the same time. The saved ECG records can be uploaded to the cloud database through the web service. The current heart rate is also presented, and an alarm message can be sent out when heart rate is lower than the preset threshold.

In Chapter 3, we have proposed a sparse binary sensing matrix to sample the ECG signal with Compressed Sensing. The sensing matrix is specially designed for real-time signal and memory restricted platforms. A variable-length compression algorithm is also raised to reduce the transmitted data. A modified CoSaMP algorithm is used to reconstruct the ECG signal. Experiment results show that a compression ratio of 57% is achieved with the proposed algorithmic approach, and it takes 64% less time to reconstruct the sampled ECG signal compared to using OMP or IRLS algorithm.

4.2 Future Work

The ECG records are currently only available to the patient users in the proposed system. For the future work, doctors can be introduced into the system to help analyse users' ECG records and also to communicate with users about their health situation. Moreover, as the ECG signal collected by Compressed Sensing cannot be viewed in real-time, a new component can be added to the Android application for plotting the CS history data. In addition, more experiments can be carried out on different abnormal ECG signals.

Appendix A

Source Codes of Wireless ECG System and Compressed Sensing Reconstruction Algorithm

Below is part of the ECG sensor firmware source codes.

Part of 'heartrateservice.c' file:

```
// Minimum connection interval (units of 1.25ms, 80=100ms) if automatic parameter update request is enabled
#define DEFAULT_DESIRED_MIN_CONN_INTERVAL     36 //45ms

// Maximum connection interval (units of 1.25ms, 800=1000ms) if automatic parameter update request is enabled
#define DEFAULT_DESIRED_MAX_CONN_INTERVAL     36 //45ms
```

```
static uint8 byteCount = 0;
static uint8 notiCount = 0;
static uint8 hrData[16];
static uint16 adcResult;
```

```
  if ( keys & HAL_KEY_SW_1 )
  {
    SK_Keys |= SK_KEY_LEFT;
    if(!ifNotiOn){
      ifNotiOn = true;
      osal_start_timerEx(simpleBLEPeripheral_TaskID, SBP_ECG_EVT, SBP_ECG_EVT_PERIOD);
    }else{
      ifNotiOn = false;
    }
  }
```

```
static void sendEcgData(void)
{
  HalAdcSetReference( HAL_ADC_REF_AVDD );
```

```
  adcResult = HalAdcRead( HAL_ADC_CHN_AIN6, HAL_ADC_RESOLUTION_10 );
  byteCount++;

  hrData[byteCount] = ((adcResult) >> 1) & 0xFF;

  if(byteCount==15){
    hrData[0] = notiCount;
    attHandleValueNoti_t nData;
    nData.len = 16;
    nData.handle = 54; //0x0036
    osal_memcpy( &nData.value, &hrData, 16 );
    GATT_Notification( 0, &nData, FALSE );
    byteCount = 0;
    notiCount++;
  }

  if(ifNotiOn)
  {
    osal_start_timerEx( simpleBLEPeripheral_TaskID, SBP_ECG_EVT, SBP_ECG_EVT_PERIOD );
  }else{
    byteCount = 0;
    notiCount = 0;
  }
}
```

Part of ‘heartrateservice.c’ file:

```
/*********************************************************************
 * @fn      heartRate_WriteAttrCB
 *
 * @brief   Validate attribute data prior to a write operation
 *
 * @param   connHandle - connection message was received on
 * @param   pAttr - pointer to attribute
 * @param   pValue - pointer to data to be written
 * @param   len - length of data
 * @param   offset - offset of the first octet to be written
 * @param   complete - whether this is the last packet
 * @param   oper - whether to validate and/or write attribute value
 *
 * @return  Success or Failure
 */
static bStatus_t heartRate_WriteAttrCB( uint16 connHandle, gattAttribute_t *pAttr,
                                        uint8 *pValue, uint8 len, uint16 offset )
{
  bStatus_t status = SUCCESS;

  uint16 uuid = BUILD_UINT16( pAttr->type.uuid[0], pAttr->type.uuid[1]);
  switch ( uuid )
  {
    case HEARTRATE_COMMAND_UUID:
      break;

    case GATT_CLIENT_CHAR_CFG_UUID:
```

```
      if(!ifNotiOn){
        ifNotiOn = true;
        osal_start_timerEx(simpleBLEPeripheral_TaskID, SBP_ECG_EVT, SBP_ECG_EVT_PERIOD);
      }else{
        ifNotiOn = false;
        //osal_stop_timerEx(simpleBLEPeripheral_TaskID, SBP_ECG_EVT);
      }
      break;

  default:
    status = ATT_ERR_ATTR_NOT_FOUND;
    break;
  }

  return ( status );
}
```

Below is part of the Android application source codes.

'BleService.java' file:

```
package com.wanboli.heartcarer;

import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.OutputStream;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.Locale;
import java.util.Timer;
import java.util.TimerTask;
import java.util.UUID;

import android.annotation.SuppressLint;
import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.app.Service;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothGatt;
import android.bluetooth.BluetoothGattCallback;
import android.bluetooth.BluetoothGattCharacteristic;
import android.bluetooth.BluetoothGattDescriptor;
import android.bluetooth.BluetoothProfile;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.net.ConnectivityManager;
import android.net.NetworkInfo;
import android.os.Bundle;
import android.os.Handler;
```

```
34 import android.os.IBinder;
35 import android.os.Message;
36 import android.os.Messenger;
37 import android.os.Vibrator;
38 import android.util.Log;
39
40 /**
41  * This Service runs after logging in and ends when exiting app.
42  *
43  * @author Wanbo Li
44  *
45  */
46 @SuppressLint("HandlerLeak")
47 public class BleService extends Service {
48   public static int ConState;
49   public static final int ConState_NotConnected = 0;
50   public static final int ConState_Connected = 1;
51   public static final int ConState_Connecting = 2;
52   public static boolean enableNoti;
53   public static boolean ifDraw;
54   public static boolean ifSaving;
55   public static BluetoothDevice mDevice;
56
57   private final String TAG = "BleService";
58   private Messenger ActivityMessenger;
59   private BluetoothGatt mBluetoothGatt;
60   private BluetoothGattCharacteristic characteristic;
61   private BluetoothGattDescriptor descriptor;
62   private boolean ifSavingToFile;
63   private boolean flagBuffer_1 = true;
64   private boolean flagBuffer_2 = false;
65   private final int buf_length = 5000;
66   private byte[] buffer_1 = new byte[buf_length];
67   private byte[] buffer_2 = new byte[buf_length];
68   private int pointerBuf = 0;
69   private boolean flagSaveBufInit = false;
70   private int pointerBufInit;
71   private int pointerBufEnd;
72   private OutputStream outputStream;
73   private SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd_HH-mm-ss", Locale.
         getDefault());
74   private String saveFileName;
75   private Timer timerSavingFile = null;
76   private TimerTask taskTimerSavingFile = null;
77   private boolean flagTimerSavingFileFirst = true;
78   private Signal signal = new Signal();
79   private boolean ifHbNormal = true;
80   private Handler mHandler = new Handler();
81   private Vibrator vibrator;
82   private final int VibrateLength = 300; //unit: ms
83   private NotificationManager mNotiManager;
84   //Compressed Sensing
85   private int csCounter;
```

```
86  private ArrayList<byte[]> csByteArray = new ArrayList<byte[]>();
87
88  public void onCreate() {
89    Log.i(TAG, "onCreate()");
90
91    initStatic();
92    registerReceiver(wifiReceiver, new IntentFilter(ConnectivityManager.
          CONNECTIVITY_ACTION));
93    vibrator = (Vibrator) getSystemService(Context.VIBRATOR_SERVICE);
94    mNotiManager = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
95  }
96  //Initiate static variables
97  private void initStatic() {
98    ConState = ConState_NotConnected;
99    enableNoti = true;
100   ifDraw = Global.ifCsMode ? false : true;
101   ifSaving = false;
102   mDevice = null;
103 }
104 //Called by the system every time a client starts the service by calling "startService"
105 @Override
106 public int onStartCommand (Intent intent, int flags, int startId){
107   Log.i(TAG, "onStartCommand()");
108   Log.v("startId", String.valueOf(startId));
109
110   if(startId==1) StartForeground();
111     return super.onStartCommand(intent, flags, startId);
112 }
113 //Show foreground notification
114 private void StartForeground() {
115   Notification noti = new Notification.Builder(BleService.this)
116       .setContentIntent(PendingIntent.getActivity(BleService.this, 0,
117           Global.defaultIntent(BleService.this), 0))
118           .setContentTitle(getResources().getString(R.string.app_name))
119           .setContentText(getResources().getString(R.string.app_name) + getResources().
                getString(R.string.ble_run))
120           .setSmallIcon(R.drawable.main_heart_beat_64)
121           .build();
122   startForeground(1, noti);
123 }
124 //Return the communication channel to the service
125 @Override
126 public IBinder onBind(Intent intent) {
127   Log.i(TAG, "onBind()");
128   return mMessenger.getBinder();
129 }
130 //When wifi is connected, upload saved files
131 private BroadcastReceiver wifiReceiver = new BroadcastReceiver(){
132   public void onReceive(Context arg0, Intent intent) {
133     NetworkInfo networkInfo = ((ConnectivityManager)getSystemService
          (Context.CONNECTIVITY_SERVICE)).getActiveNetworkInfo();
134     if(networkInfo != null && networkInfo.getType() == ConnectivityManager.TYPE_WIFI
          && Global.ifRegUser && !Global.ifUploading){
135
136
```

```
137         startService(new Intent(BleService.this, UpdataService.class));
138       }
139     }
140   };
141   //Disconnect BLE connection
142   private void disconnect() {
143       mBluetoothGatt.disconnect();
144   }
145   //Handler which handles incoming Message
146   private class IncomingHandler extends Handler {
147     public void handleMessage(Message msg) {
148       int i = msg.what;
149       if(i==0){
150         //Register
151         ActivityMessenger = msg.replyTo;
152         sendVoidToAM(6);
153       }else if(i==1){
154         //Connect
155         mBluetoothGatt = mDevice.connectGatt(BleService.this, false, mGattCallback);
156       }else if(i==2){
157         //Disconnect
158         disconnect();
159       }else if(i==3){
160         //SetNoti
161         setNoti();
162       }else if(i==4){
163         //Button - SaveToFile
164         if(ConState!=ConState_Connected || enableNoti){
165           toastMakeText("Please start " + getResources().getString(R.string.global_sensor)
                + "!");
166           sendVoidToAM(4);
167             return;
168           }else{
169             if(ifSaving){
170               stopSavingFinal();
171             }else{
172               startSaving();
173               ifSaving = true;
174               toastMakeText("Start saving!");
175             sendVoidToAM(3);
176             if(!Global.ifCsMode) startTimerSavingFile();
177             }
178           }
179       }
180     }
181   }
182   //Local Messenger used to talk to ActivityMessenger, Message received by IncomingHandler
183   final Messenger mMessenger = new Messenger(new IncomingHandler());
184   private void sendVoidToAM(int i){
185     if(Global.ifHrmFragmentAlive){
186       try {
187         ActivityMessenger.send(Message.obtain(null, i));
188       } catch (Exception e) {
```

```
189         e.printStackTrace();
190       }
191     }
192   }
193   //Start saving ECG data to file
194   private void startSaving(){
195     Log.v(TAG, "startSaving()");
196
197     saveFileName = sdf.format(new Date(System.currentTimeMillis())) +
198         (Global.ifCsMode ? "_cs_" : "") + ".bin";
199     try{
200       outputStream = new BufferedOutputStream(new FileOutputStream(Global.cachePath +
201           "/" + saveFileName));
202     }catch (Exception e){
203       e.printStackTrace();
204       toastMakeText("Error: Start saving!");
205       stopSavingFinal();
206       return;
207     }
208     if(Global.ifCsMode){
209       csCounter = 0;
210     }else{
211       flagSaveBufInit = true;
212       pointerBufInit = pointerBuf;
213     }
214   }
215   //Stop saving ECG data to file
216   private void stopSaving(){
217     Log.v(TAG, "stopSaving()");
218
219     if(Global.ifCsMode){
220       if(ifSavingToFile) return;
221     }else{
222       pointerBufEnd = pointerBuf;
223       byte[] tempByte = new byte[flagSaveBufInit ?
224           pointerBufEnd - pointerBufInit : pointerBufEnd];
225       System.arraycopy(flagBuffer_1 ? buffer_1 : buffer_2,
226           flagSaveBufInit ? pointerBufInit : 0, tempByte, 0,
227           flagSaveBufInit ? pointerBufEnd-pointerBufInit : pointerBufEnd);
228       saveToFile(tempByte);
229     }
230     try{
231       outputStream.close();
232       String oldPath = Global.cachePath + "/" + saveFileName;
233       if(Global.ifCsMode){
234         csByteArray.clear();
235         if(csCounter==0){
236           toastMakeText("Less than 4 seconds, no data saved!");
237           new File(oldPath).delete();
238           return;
239         }
240         String oldPathCs = oldPath;
241         saveFileName = saveFileName.replace(".", csCounter + ".");
```

```
242         csCounter = 0;
243         oldPath = Global.cachePath + "/" + saveFileName;
244         new File(oldPathCs).renameTo(new File(oldPath));
245       }
246       String newPath = Global.savedPath + "/" + saveFileName;
247       new File(oldPath).renameTo(new File(newPath));
248     }catch (Exception e){
249         e.printStackTrace();
250         toastMakeText("Error: Stop saving!");
251     }
252     toastMakeText(saveFileName + " saved!");
253     if((!Global.ifUploading) && Global.isWifiConn(BleService.this)){
254       startService(new Intent(BleService.this, UpdataService.class));
255     }
256     System.gc();
257   }
258   //Stop saving to file thoroughly
259   private final void stopSavingFinal(){
260     if(ifSaving){
261       if(!Global.ifCsMode) cancelTimerSavingFile();
262       ifSaving = false;
263       stopSaving();
264     }
265     sendVoidToAM(4);
266   }
267   //BLE Callback
268   private final BluetoothGattCallback mGattCallback = new BluetoothGattCallback() {
269     //Called when connection state changes
270     public void onConnectionStateChange (BluetoothGatt gatt, int status, int newState){
271       if (newState == BluetoothProfile.STATE_CONNECTED) {
272         ConState = ConState_Connected;
273         sendVoidToAM(1);
274         mBluetoothGatt.discoverServices();
275         vibrator.vibrate(VibrateLength);
276       }else if (newState == BluetoothProfile.STATE_DISCONNECTED) {
277         ConState = ConState_NotConnected;
278         enableNoti = true;
279         sendVoidToAM(2);
280         //Re-initiate mBluetoothGatt
281         if(mBluetoothGatt != null){
282               mBluetoothGatt.close();
283               mBluetoothGatt = null;
284             }
285         stopSavingFinal();
286         vibrator.vibrate(VibrateLength);
287       }
288     }
289     //Called when BLE services are discovered
290     public void onServicesDiscovered(BluetoothGatt gatt, int status) {
291       Log.v(TAG, "onServicesDiscovered()");
292       if (status==BluetoothGatt.GATT_SUCCESS) {
293         initNoti();
294         setNoti();
```

```
295       }else{
296         toastMakeText("Error: Discover Services!");
297       }
298     }
299     //Called when notification received
300     @SuppressWarnings("unchecked")
301     public void onCharacteristicChanged (BluetoothGatt gatt,
302         BluetoothGattCharacteristic characteristic){
303       //Log.v(TAG, "onCharacteristicChanged()");
304       byte[] notiValue = characteristic.getValue();
305       if(Global.ifCsMode){
306         //Log.v(TAG, Integer.toBinaryString(notiValue[0] & 0xFF));
307         if(!ifSaving) return;
308         if((notiValue[0] & 0x80) == 0x80){
309           Log.v(TAG, "New Cycle");
310           if(csByteArray.size() > 0){
311             saveToFileCs((ArrayList<byte[]>)csByteArray.clone());
312             csByteArray.clear();
313           }
314           csByteArray.add(notiValue);
315         }else{
316           if(csByteArray.size() > 0) csByteArray.add(notiValue);
317         }
318       }else{
319         //For each byte, put into buffer and plot if requested
320         byte tempValue;
321         for(int i=1; i<notiValue.length; i++){
322           tempValue = notiValue[i];
323           if(flagBuffer_1){
324             buffer_1[pointerBuf] = tempValue;
325             if(pointerBuf == buf_length-1){
326               pointerBuf = 0;
327               flagBuffer_1 = false;
328               flagBuffer_2 = true;
329               if(ifSaving) saveBufData(1);
330               showHeartBeat(1);
331             }else pointerBuf++;
332           }else if(flagBuffer_2){
333             buffer_2[pointerBuf] = tempValue;
334             if(pointerBuf==(buf_length-1)){
335               pointerBuf = 0;
336               flagBuffer_2 = false;
337               flagBuffer_1 = true;
338               if(ifSaving) saveBufData(2);
339               showHeartBeat(2);
340             }else pointerBuf++;
341           }
342           //Draw graph every 5 data when ifDraw==true
343           if(ifDraw && Global.ifHrmFragmentAlive && (i==1||i==6||i==11)){
344             try {
345               Message msg = Message.obtain(null, 0);
346               Bundle b = new Bundle();
347               b.putByte("tempValue", tempValue);
```

```
348             msg.setData(b);
349             ActivityMessenger.send(msg);
350           }catch (Exception e){
351             e.printStackTrace();
352           }
353         }
354       }
355     }
356   }
357 };
358 /**
359  * Save ECG data to file (Compressed Sensing)
360  *
361  * @param byteArrayList: ECG data to save
362  */
363 private void saveToFileCs(ArrayList<byte[]> byteArrayList) {
364   Log.v(TAG, "saveToFileCs - Start");
365
366   ifSavingToFile = true;
367   csCounter++;
368   for(byte[] data : byteArrayList){
369     try {
370         outputStream.write(data);
371     } catch (Exception e) {
372       e.printStackTrace();
373       toastMakeText("Error: saveToFileCs!");
374       stopSavingFinal();
375     }
376   }
377   ifSavingToFile = false;
378   if(!ifSaving) stopSaving();
379   if(csCounter==Global.savingLength/1000/4){
380     stopSaving();
381     startSaving();
382   }
383
384   Log.v(TAG, "saveToFileCs - End");
385 }
386 /**
387  * Calculate Heart Beat
388  *
389  * @param i: Which buffer to calculate Heart Beat, i = 1 or 2
390  */
391 private void showHeartBeat(final int i) {
392   int bpm = (i==1) ? signal.getFeatures(buffer_1, 5).getBpm() :
393     signal.getFeatures(buffer_2, 5).getBpm();
394   updateBeat(bpm);
395   if(bpm < Global.lowBpm && ifHbNormal){
396     showNotification();
397     ifHbNormal = false;
398   }
399   if(bpm >= 60) ifHbNormal = true;
400 }
```

```
401   /**
402    * Show Heart Beat on screen and widget
403    *
404    * @param bpm: Heart Beat to show in integer
405    */
406   private void updateBeat(int bpm) {
407     try {
408       Message msg = Message.obtain(null, 5);
409       Bundle b = new Bundle();
410       b.putInt("bpm", bpm);
411       msg.setData(b);
412       ActivityMessenger.send(msg);
413     } catch (Exception e) {
414       e.printStackTrace();
415     }
416     //Update widget
417       sendBroadcast(new Intent(Global.WidgetAction).putExtra("bpm", bpm));
418   }
419   //Show notification when low Heart Beat, and send SMS
420   private void showNotification() {
421     Notification noti = new Notification.Builder(BleService.this)
422           .setContentIntent(PendingIntent.getActivity
423               (BleService.this, 0, Global.defaultIntent(BleService.this), 0))
424             .setContentTitle(getResources().getString(R.string.app_name))
425             .setContentText("Warning: Low heart beat!")
426             .setSmallIcon(R.drawable.warning_64)
427             .setAutoCancel(true)
428             .setLights(Global.color_Red, 2000, 1000)
429             .build();
430     mNotiManager.notify(0, noti);
431     vibrator.vibrate(3000);
432     if(Global.ifRegUser && Global.ifSendSms){
433       if(Global.ifAppendLoc && MainActivity.ifLCConnected){
434         new Thread(){
435           public void run(){
436             Global.sendSMS(Global.emergencynum, Global.emergencymes +
437                 " My current location: " +
438                 MainActivity.getCurAddress(BleService.this));
439           }
440         }.start();
441       }else{
442         Global.sendSMS(Global.emergencynum, Global.emergencymes);
443       }
444     }
445   }
446   /**
447    * Save ECG data from buffer to file
448    *
449    * @param buf: Which buffer to save, buf = 1 or 2
450    */
451   private void saveBufData(final int buf) {
452     if(flagSaveBufInit){
453       flagSaveBufInit = false;
```

```
454         byte[] tempByte = new byte[buf_length - pointerBufInit];
455         System.arraycopy(buf==1 ? buffer_1 : buffer_2,
456             pointerBufInit, tempByte, 0, buf_length - pointerBufInit);
457         saveToFile(tempByte);
458       }else{
459         saveToFile(buf==1 ? buffer_1 : buffer_2);
460       }
461     }
462     /**
463      * Save ECG data to file
464      *
465      * @param data: ECG data to save in byte array
466      */
467     private void saveToFile(final byte[] data) {
468       try {
469           outputStream.write(data);
470       } catch (Exception e) {
471         e.printStackTrace();
472         toastMakeText("Error: saveToFile!");
473         stopSavingFinal();
474       }
475     }
476     //Initiate for Notification receiving
477     private void initNoti() {
478       characteristic = mBluetoothGatt
479           .getService(UUID.fromString("0000180d-0000-1000-8000-00805f9b34fb"))
480           .getCharacteristic(UUID.fromString("00002a37-0000-1000-8000-00805f9b34fb"));
481       descriptor = characteristic.getDescriptor
482           (UUID.fromString("00002902-0000-1000-8000-00805f9b34fb"));
483       descriptor.setValue(BluetoothGattDescriptor.ENABLE_NOTIFICATION_VALUE);
484     }
485     //Enable or disable Notification
486     private void setNoti() {
487       mBluetoothGatt.setCharacteristicNotification(characteristic, enableNoti);
488       mBluetoothGatt.writeDescriptor(descriptor);
489       if(ifSaving && !enableNoti){
490         stopSavingFinal();
491       }
492       enableNoti = !enableNoti;
493       csByteArray.clear();
494     }
495     //Start Timer for saving ECG data to file
496     private void startTimerSavingFile() {
497       if(timerSavingFile == null){
498         timerSavingFile = new Timer();
499       }
500       if(taskTimerSavingFile == null){
501         taskTimerSavingFile = new TimerTask(){
502           public void run(){
503             if(flagTimerSavingFileFirst){
504               flagTimerSavingFileFirst = false;
505               return;
506             }
```

```
507             handlerTimer.sendEmptyMessage(0);
508           }
509         };
510       }
511       if((timerSavingFile != null)&&(taskTimerSavingFile != null)){
512         timerSavingFile.scheduleAtFixedRate(taskTimerSavingFile, 0, Global.savingLength);
513       }
514     }
515     //Start Timer for saving ECG data to file
516     private void cancelTimerSavingFile() {
517       flagTimerSavingFileFirst = true;
518       if(timerSavingFile != null){
519         timerSavingFile.cancel();
520         timerSavingFile = null;
521       }
522       if(taskTimerSavingFile != null){
523         taskTimerSavingFile.cancel();
524         taskTimerSavingFile = null;
525       }
526     }
527     //Handler which stops previous saving and starts a new one
528     private Handler handlerTimer = new Handler(){
529       public void handleMessage(Message msg){
530         stopSaving();
531         startSaving();
532       }
533     };
534     //Use mHandler to make toast text asynchronously
535     private final void toastMakeText(final String string){
536       mHandler.post(new Runnable() {
537         public void run() {
538           Global.toastMakeText(BleService.this, string);
539         }
540       });
541     }
542     //Life cycle
543     public void onDestroy() {
544       Log.i(TAG, "onDestroy()");
545 
546       //Update widget
547         sendBroadcast(new Intent(Global.WidgetAction).putExtra("bpm", 0));
548           unregisterReceiver(wifiReceiver);
549           stopSavingFinal();
550           if(ConState != ConState_NotConnected) disconnect();
551           if(mBluetoothGatt != null){
552             mBluetoothGatt.close();
553             mBluetoothGatt = null;
554           }
555       mNotiManager.cancelAll();
556     }
557 }
```

Below is part of the web service source codes.

'EcgCloudDatabase.java' file:

```
1  package robertlee.restwep.resource;
2
3  import java.io.File;
4  import java.io.FileOutputStream;
5  import java.io.InputStream;
6  import java.io.OutputStream;
7  import java.nio.ByteBuffer;
8  import java.sql.Connection;
9  import java.sql.DriverManager;
10 import java.sql.PreparedStatement;
11 import java.sql.ResultSet;
12 import java.sql.Statement;
13 import java.text.SimpleDateFormat;
14 import java.util.ArrayList;
15 import java.util.Date;
16 import java.util.Locale;
17
18 import javax.naming.InitialContext;
19 import javax.servlet.http.HttpServletRequest;
20 import javax.sql.DataSource;
21 import javax.ws.rs.Consumes;
22 import javax.ws.rs.GET;
23 import javax.ws.rs.POST;
24 import javax.ws.rs.Path;
25 import javax.ws.rs.Produces;
26 import javax.ws.rs.core.Context;
27 import javax.ws.rs.core.MediaType;
28 import javax.ws.rs.core.Request;
29 import javax.ws.rs.core.Response;
30 import javax.ws.rs.core.UriInfo;
31
32 import org.apache.commons.codec.binary.Base64;
33 import org.codehaus.jettison.json.JSONArray;
34 import org.codehaus.jettison.json.JSONException;
35 import org.codehaus.jettison.json.JSONObject;
36
37 import com.sun.jersey.core.header.ContentDisposition;
38 import com.sun.jersey.core.header.FormDataContentDisposition;
39 import com.sun.jersey.multipart.FormDataMultiPart;
40 import com.sun.jersey.multipart.FormDataParam;
41
42 /**
43  * This class behaves as a Web Service providing various functions
44  */
45 @Path("/")
46 public class EcgCloudDatabase {
47     private final String apkFileLocation = "D:/Android/Heart Carer APK";
48     private InitialContext initContext = null;
49   private DataSource ds = null;
50
51     @Context
```

```
52      UriInfo uriInfo;
53      @Context
54      Request request;
55      //Running test
56      @GET
57      @Produces(MediaType.TEXT_PLAIN)
58      public String runningTest() {
59        return "EcgCloudDatabase Web Service is running!";
60      }
61      //Login
62      @POST
63      @Path("login")
64      @Consumes(MediaType.APPLICATION_JSON)
65      @Produces(MediaType.APPLICATION_JSON)
66      public JSONObject login(JSONObject loginPara, @Context HttpServletRequest request){
67        System.out.println("Login!");
68        Connection connection = null;
69        Statement statement = null;
70        ResultSet rs = null;
71        PreparedStatement ps = null;
72
73        JSONObject response = new JSONObject();
74        try {
75          connection = dbConnect();
76          String username = loginPara.getString("username");
77          String password = loginPara.getString("password");
78          statement = connection.createStatement();
79            rs = statement.executeQuery(String.format
80                ("select * from userinfo where username='%s'", username));
81            if(rs.next()){
82              if(rs.getString("userpassword").equals(password)){
83                response.put("result", "Success: Login!");
84                response = generalLogin(response, rs);
85                updateLastDeviceIP(statement, rs.getInt("userid"), loginPara,
86                    getClientIP(request));
87              }else{
88                response.put("result", "Password not match!");
89              }
90            }else{
91              response.put("result", "This user doesn't exist!");
92            }
93      } catch (Exception e) {
94        e.printStackTrace();
95        try {
96          response.put("result", "Error: Log in");
97        } catch (JSONException e1) {
98          e1.printStackTrace();
99        }
100     }finally{
101       dbDisconnect(connection, statement, rs, ps);
102     }
103       return response;
104     }
```

```
105   //Signup
106     @POST
107     @Path("signup")
108     @Consumes(MediaType.APPLICATION_JSON)
109     @Produces(MediaType.APPLICATION_JSON)
110     public JSONObject signup(JSONObject signupPara, @Context HttpServletRequest request){
111       System.out.println("Signup!");
112       Connection connection = null;
113       Statement statement = null;
114       ResultSet rs = null;
115       PreparedStatement ps = null;
116
117       JSONObject response = new JSONObject();
118       try {
119         String username = signupPara.getString("username");
120         String phone = signupPara.getString("phone");
121         String email = signupPara.getString("email");
122         String password = signupPara.getString("password");
123         String firstname = signupPara.getString("firstname");
124         String lastname = signupPara.getString("lastname");
125         connection = dbConnect();
126         statement = connection.createStatement();
127           rs = statement.executeQuery(String.format("select userid from userinfo where
                username='%s'", username));
128           if(!rs.next()){
129             String insert_userinfo = "insert into userinfo(username,userpassword,email,
                  firstname,lastname,phone)"+
130                       " values (?,?,?,?,?,?)";
131           ps = connection.prepareStatement(insert_userinfo, Statement.
                RETURN_GENERATED_KEYS);
132           ps.setString(1, username);
133         ps.setString(2, password);
134         ps.setString(3, email);
135         ps.setString(4, firstname);
136         ps.setString(5, lastname);
137         ps.setString(6, phone);
138         int count = ps.executeUpdate();
139         if(count==1){
140           ResultSet rs_userid = ps.getGeneratedKeys();
141           if(rs_userid.next()){
142             int userid=rs_userid.getInt(1);
143             response.put("result", "Success: Signup!");
144             response.put("userid", userid);
145             response = defaultUser(response);
146             updateLastDeviceIP(statement, userid, signupPara, getClientIP(request));
147           }else{
148             response.put("result", "Failure: Signup - Get userid!");
149           }
150         }else{
151           response.put("result", "Failure: Signup!");
152         }
153           }else{
154             response.put("result", "Username already existed!");
```

```
155           }
156       } catch (Exception e) {
157         e.printStackTrace();
158         try {
159           response.put("result", "Error: Sign up");
160         } catch (JSONException e1) {
161           e1.printStackTrace();
162         }
163       }finally{
164         dbDisconnect(connection, statement, rs, ps);
165       }
166         return response;
167       }
168     //Updata - Upload saved ECG data to Cloud Server
169       @POST
170       @Path("updata")
171       @Consumes(MediaType.MULTIPART_FORM_DATA)
172       @Produces(MediaType.TEXT_PLAIN)
173       public String updata(FormDataMultiPart updataPara) throws Exception{
174         System.out.println("Updata!");
175         Connection connection = null;
176         Statement statement = null;
177         ResultSet rs = null;
178         PreparedStatement ps = null;
179
180         try {
181           int userid = Integer.valueOf(updataPara.getField("userid").getValue());
182           String starttime = updataPara.getField("starttime").getValue();
183           int length = Integer.parseInt(updataPara.getField("length").getValue());
184           boolean ifcs = Boolean.parseBoolean(updataPara.getField("ifcs").getValue());
185           connection = dbConnect();
186           //Check if data already uploaded
187           statement = connection.createStatement();
188           rs = statement.executeQuery(String.format("select ifdone from ecgdatainfo where
                userid=%s and starttime='%s'",
189               String.valueOf(userid), starttime));
190           if(rs.next()){
191             if(rs.getBoolean("ifdone")){
192                 return "Data already existed in database!";
193               }else{
194                 int count_del = statement.executeUpdate(String.format("delete from
                      ecgdatainfo where userid=%s and starttime='%s'",
195                      String.valueOf(userid), starttime));
196                 if(count_del != 1) return "Error: delete from ecgdatainfo!";
197               }
198             }
199           //Data not uploaded, upload data
200           String insert_ecgdatainfo = "insert into ecgdatainfo(starttime, length, ifdone,
                userid, ifcs)"+
201                    " values (?, ?, ?, ?, ?)";
202           ps = connection.prepareStatement(insert_ecgdatainfo, Statement.
                RETURN_GENERATED_KEYS);
203         ps.setString(1, starttime);
```

```
204       ps.setInt(2, length);
205       ps.setBoolean(3, false);
206       ps.setInt(4, userid);
207       ps.setBoolean(5, ifcs);
208       int count_1 = ps.executeUpdate();
209           if(count_1 == 1){
210             ResultSet rs_dataid = ps.getGeneratedKeys();
211         int dataid = 0;
212         if(rs_dataid.next()){
213           dataid = rs_dataid.getInt(1);
214           String insert_ecgdata = "insert into ecgdata(dataid, datacontent)"+" values (?,
                  ?)";
215           ps.close();
216               ps = connection.prepareStatement(insert_ecgdata);
217               ps.setInt(1, dataid);
218               InputStream content = updataPara.getField("content").getValueAs(InputStream.
                      class);
219               ps.setBinaryStream(2, content);
220           int count_2 = ps.executeUpdate();
221               if(count_2==1){
222                 int count_3 = statement.executeUpdate(String.format
223                     ("update ecgdatainfo set ifdone=true where dataid=%s", String.valueOf(
                          dataid)));
224                 return count_3==1 ? "Success: Updata!" : "Failure: Updata!";
225               }else{
226                 return "Error: Insert ecgdata!";
227               }
228         }else{
229           return "No dataid in ecgdatainfo!";
230         }
231           }else{
232             return "Error: Insert ecgdatainfo!";
233           }
234     } catch (Exception e) {
235       e.printStackTrace();
236       return "Error: Updata";
237     }finally{
238       dbDisconnect(connection, statement, rs, ps);
239     }
240     }
241     //Downdata - Download saved ECG data from Cloud Server
242   @POST
243     @Path("downdata")
244     @Consumes(MediaType.APPLICATION_JSON)
245     @Produces(MediaType.APPLICATION_JSON)
246     public JSONObject downdata(JSONObject downdataPara){
247       System.out.println("Downdata!");
248       Connection connection = null;
249       Statement statement = null;
250       ResultSet rs = null;
251       PreparedStatement ps = null;
252
253       JSONObject response = new JSONObject();
```

```
254     try {
255       int userid = downdataPara.getInt("userid");
256       String starttime = downdataPara.getString("starttime");
257       connection = dbConnect();
258       statement = connection.createStatement();
259         rs = statement.executeQuery(String.format("select dataid, ifcs from ecgdatainfo
              where userid=%s and starttime='%s'",
260             String.valueOf(userid), starttime));
261         if(rs.next()){
262           int dataid = rs.getInt("dataid");
263           boolean ifcs = rs.getBoolean("ifcs");
264           rs.close();
265           rs = statement.executeQuery(String.format("select datacontent from ecgdata
                where dataid=%s", String.valueOf(dataid)));
266           if(rs.next()){
267             response.put("result", "Success: Downdata!");
268             InputStream inStream = rs.getBinaryStream("datacontent");
269             //Compressed Sensing
270             if(ifcs){
271               ArrayList<byte[]> tempByteArray = new ArrayList<byte[]>();
272               int flag = 1;
273               byte[] tmp = new byte[20];
274               ArrayList<byte[]> resultArray = new ArrayList<byte[]>();
275               while((inStream.read(tmp)) != -1){
276                 //New Cycle
277                 if((tmp[0] & 0x80) == 0x80){
278                   flag++;
279                   if(flag != 2){
280                     resultArray.add(CompressedSensing.csReconstruct(tempByteArray));
281                   }
282                   tempByteArray.clear();
283                 }
284                 tempByteArray.add(tmp);
285               }
286               resultArray.add(CompressedSensing.csReconstruct(tempByteArray));
287               byte[] contentByte = new byte[2048 * flag];
288               ByteBuffer target = ByteBuffer.wrap(contentByte);
289               for(byte[] temp : tempByteArray) target.put(temp);
290               response.put("content", Base64.encodeBase64String(contentByte));
291             }else{
292               //Normal
293               int flag = 0;
294               byte[] tmp = new byte[1];
295               ArrayList<Byte> byteList = new ArrayList<Byte>();
296               while((inStream.read(tmp)) != -1){
297                 byteList.add(tmp[0]);
298                 flag++;
299                 //Max 1-minute data
300                 if(flag==60*1000) break;
301               }
302               inStream.close();
303               int byteLength = byteList.size();
304               byte[] contentByte = new byte[byteLength];
```

```
305                 for(int i=0; i<byteLength; i++){
306                   contentByte[i] = byteList.get(i);
307                 }
308                 response.put("content", Base64.encodeBase64String(contentByte));
309               }
310             }else{
311               response.put("result", "No data found in ecgdata table!");
312             }
313           }else{
314             response.put("result", "Data required not found!");
315           }
316     } catch (Exception e) {
317       e.printStackTrace();
318       try {
319         response.put("result", "Error: Down data");
320       } catch (JSONException e1) {
321         e1.printStackTrace();
322       }
323     }finally{
324       dbDisconnect(connection, statement, rs, ps);
325     }
326     return response;
327     }
328   //DownUserInfo - Return user profile information
329     @POST
330     @Path("downuserinfo")
331     @Consumes(MediaType.TEXT_PLAIN)
332     @Produces(MediaType.APPLICATION_JSON)
333     public JSONObject downuserinfo(String userid_string){
334       System.out.println("DownUserInfo!");
335       Connection connection = null;
336       Statement statement = null;
337       ResultSet rs = null;
338       PreparedStatement ps = null;
339
340       int userid = Integer.valueOf(userid_string);
341       JSONObject response = new JSONObject();
342       try {
343         connection = dbConnect();
344         statement = connection.createStatement();
345           rs = statement.executeQuery(String.format
346               ("select email,firstname,lastname,phone from userinfo where userid=%s",
                      String.valueOf(userid)));
347           if(rs.next()){
348             response.put("result", "Success: DownUserInfo!");
349             response.put("email", rs.getString("email"));
350             response.put("firstname", rs.getString("firstname"));
351             response.put("lastname", rs.getString("lastname"));
352             response.put("phone", rs.getString("phone"));
353           }else{
354             response.put("result", "User not found!");
355           }
356     } catch (Exception e) {
```

```
357       e.printStackTrace();
358       try {
359         response.put("result", "Error: Down User Info!");
360       } catch (JSONException e1) {
361         e1.printStackTrace();
362       }
363     }finally{
364       dbDisconnect(connection, statement, rs, ps);
365     }
366       return response;
367     }
368     //UpUserInfo - Update user profile information
369     @POST
370     @Path("upuserinfo")
371     @Consumes(MediaType.APPLICATION_JSON)
372     @Produces(MediaType.TEXT_PLAIN)
373     public String upuserinfo(JSONObject request){
374       System.out.println("UpUserInfo!");
375       Connection connection = null;
376       Statement statement = null;
377       ResultSet rs = null;
378       PreparedStatement ps = null;
379
380       try {
381         int userid = request.getInt("userid");
382         String phone = request.getString("phone");
383         String email = request.getString("email");
384         String firstname = request.getString("firstname");
385         String lastname = request.getString("lastname");
386         String oriPassWord = request.getString("originalPW");
387         String passWord = request.getString("newPW");
388         connection = dbConnect();
389         if(oriPassWord.equals("")){
390             statement = connection.createStatement();
391             int count=statement.executeUpdate
392                 (String.format("update userinfo set phone='%s', email='%s', firstname='%s
                       ', lastname='%s' where userid=%s",
393                     phone, email, firstname, lastname, String.valueOf(userid)));
394             return count==1 ? "Success: UpUserInfo!" : "Failure: UpUserInfo!";
395         }else{
396           statement = connection.createStatement();
397               rs = statement.executeQuery(String.format("select userpassword from userinfo
                      where userid=%s", String.valueOf(userid)));
398               if(rs.next()){
399                 if(rs.getString("userpassword").equals(oriPassWord)){
400                   statement.close();
401                   statement = connection.createStatement();
402                   int count = statement.executeUpdate
403                       (String.format("update userinfo set userpassword='%s', phone='%s',
                             email='%s', firstname='%s', lastname='%s' where userid=%s",
404                             passWord, phone, email, firstname, lastname, String.valueOf(
                                 userid)));
405                   return count==1 ? "Success: UpUserInfo!" : "Failure: UpUserInfo!";
```

```
406                 }else{
407                   return "Wrong original password!";
408                 }
409               }else{
410                 return "No original password found in database!";
411               }
412         }
413       } catch (Exception e) {
414         e.printStackTrace();
415         return "Error: Up User Info!";
416       }finally{
417         dbDisconnect(connection, statement, rs, ps);
418       }
419       }
420       //GetDataList - Return history ECG data list
421       @POST
422       @Path("getdatalist")
423       @Consumes(MediaType.APPLICATION_JSON)
424       @Produces(MediaType.APPLICATION_JSON)
425       public JSONArray getdatalist(JSONObject getdatalistPara){
426         System.out.println("GetDataList!");
427         Connection connection = null;
428         Statement statement = null;
429         ResultSet rs = null;
430         PreparedStatement ps = null;
431
432         JSONArray response = new JSONArray();
433         try {
434           int userid = getdatalistPara.getInt("userid");
435           String startdate = getdatalistPara.getString("startdate");
436           String enddate = getdatalistPara.getString("enddate");
437           boolean ifSearchAll = getdatalistPara.getBoolean("ifSearchAll");
438           String datamode = getdatalistPara.getString("datamode");
439           connection = dbConnect();
440           statement = connection.createStatement();
441           String tmp;
442           if(datamode.equals("Both")){
443             tmp = "select starttime, length, ifcs from ecgdatainfo where ifdone=true";
444             rs = statement.executeQuery(ifSearchAll ?
445                 String.format(tmp + " and userid=%s", String.valueOf(userid)) :
446                 String.format(tmp + " and userid=%s and starttime>='%s' and starttime<='%s'"
                        ,
447                   String.valueOf(userid), startdate, enddate));
448             if(!rs.next()){
449                   response.put("No data found in this date range!");
450                 }else{
451                   response.put("Success: GetDataList!");
452                   rs.beforeFirst();
453                   while(rs.next()){
454                     response.put(rs.getString("starttime").substring(0, 19) +
455                         (rs.getBoolean("ifcs") ? " - CS" : ""));
456                     response.put(rs.getInt("length"));
457                   }
```

```
458             }
459         }else if(datamode.equals("Normal")){
460           tmp = "select starttime, length from ecgdatainfo where ifcs=false and ifdone=
                  true";
461           rs = statement.executeQuery(ifSearchAll ?
462               String.format(tmp + " and userid=%s", String.valueOf(userid)) :
463               String.format(tmp + " and userid=%s and starttime>='%s' and starttime<='%s'"
                      ,
464                   String.valueOf(userid), startdate, enddate));
465           if(!rs.next()){
466                 response.put("No data found in this date range!");
467               }else{
468                 response.put("Success: GetDataList!");
469                 rs.beforeFirst();
470                 while(rs.next()){
471                   response.put(rs.getString("starttime").substring(0, 19));
472                   response.put(rs.getInt("length"));
473                 }
474               }
475         }else{
476           tmp = "select starttime, length from ecgdatainfo where ifcs=true and ifdone=true
                  ";
477           rs = statement.executeQuery(ifSearchAll ?
478               String.format(tmp + " and userid=%s", String.valueOf(userid)) :
479               String.format(tmp + " and userid=%s and starttime>='%s' and starttime<='%s'"
                      ,
480                   String.valueOf(userid), startdate, enddate));
481           if(!rs.next()){
482                 response.put("No data found in this date range!");
483               }else{
484                 response.put("Success: GetDataList!");
485                 rs.beforeFirst();
486                 while(rs.next()){
487                   response.put(rs.getString("starttime").substring(0, 19) + " - CS");
488                   response.put(rs.getInt("length"));
489                 }
490               }
491         }
492           return response;
493       } catch (Exception e) {
494         e.printStackTrace();
495         try {
496           response.put(0, "Error: GetDataList!");
497         } catch (JSONException e1) {
498           e1.printStackTrace();
499         }
500         return response;
501       }finally{
502         dbDisconnect(connection, statement, rs, ps);
503       }
504       }
505       //DeleteData - Delete selected ECG data in Cloud Database
506       @POST
```

```
507     @Path("deletedata")
508     @Consumes(MediaType.APPLICATION_JSON)
509     @Produces(MediaType.TEXT_PLAIN)
510     public String deletedata(JSONObject deletedataPara){
511       System.out.println("DeleteData!");
512       Connection connection = null;
513       Statement statement = null;
514       ResultSet rs = null;
515       PreparedStatement ps = null;
516
517       try {
518         int userid = deletedataPara.getInt("userid");
519         String starttime = deletedataPara.getString("starttime");
520         connection = dbConnect();
521         statement = connection.createStatement();
522         rs = statement.executeQuery(String.format("select dataid from ecgdatainfo " +
523             "where userid=%s and starttime='%s'", String.valueOf(userid), starttime));
524         if(rs.next()){
525           int count_del = statement.executeUpdate(String.format
526               ("delete from ecgdatainfo where userid=%s and starttime='%s'",
527                   String.valueOf(userid), starttime));
528             return count_del==1 ? "Success: DeleteData!" : "Error: delete from ecgdatainfo
                    !";
529           }else{
530             return "Not found in database!";
531           }
532     } catch (Exception e) {
533       e.printStackTrace();
534       return "Error: DeleteData!";
535     }finally{
536       dbDisconnect(connection, statement, rs, ps);
537     }
538     }
539     //UpSms - Update notification settings
540     @POST
541     @Path("upsms")
542     @Consumes(MediaType.APPLICATION_JSON)
543     @Produces(MediaType.TEXT_PLAIN)
544     public String upsms(JSONObject upsmsPara){
545       System.out.println("UpSms!");
546       Connection connection = null;
547       Statement statement = null;
548       ResultSet rs = null;
549       PreparedStatement ps = null;
550
551       try {
552         int  userid = upsmsPara.getInt("userid");
553         String emergencynum = upsmsPara.getString("emergencynum");
554         String emergencymes = upsmsPara.getString("emergencymes");
555         boolean ifSendSms = upsmsPara.getBoolean("ifSendSms");
556         boolean ifAppendLoc = upsmsPara.getBoolean("ifAppendLoc");
557         connection = dbConnect();
558         statement = connection.createStatement();
```

```
559         int count = statement.executeUpdate(String.format("update userinfo set
                emergencynum='%s',emergencymes='%s',ifsendsms=%s,ifappendloc=%s where userid=%
                s",
560             emergencynum,emergencymes,String.valueOf(ifSendSms),String.valueOf(ifAppendLoc
                    ),String.valueOf(userid)));
561         return count==1 ? "Success: UpSms!" : "Error: update userinfo!";
562       } catch (Exception e) {
563         e.printStackTrace();
564         return "Error: UpSms!";
565       }finally{
566         dbDisconnect(connection, statement, rs, ps);
567       }
568       }
569       //Facebook Login
570       @POST
571       @Path("fblogin")
572       @Consumes(MediaType.APPLICATION_JSON)
573       @Produces(MediaType.APPLICATION_JSON)
574       public JSONObject fblogin(JSONObject request, @Context HttpServletRequest httpRequest)
              {
575         System.out.println("Facebook Login!");
576         Connection connection = null;
577         Statement statement = null;
578         ResultSet rs = null;
579         PreparedStatement ps = null;
580
581         JSONObject response = new JSONObject();
582         try {
583           String facebookid = request.getString("facebookid");
584           String firstname = request.getString("firstname");
585           String lastname = request.getString("lastname");
586           String email = request.getString("email");
587           String phone = request.getString("phone");
588           connection = dbConnect();
589           statement = connection.createStatement();
590             rs = statement.executeQuery(String.format
591                 ("select * from userinfo where facebookid='%s'", facebookid));
592             if(rs.next()){
593               response.put("result", "Success: Facebook Login!");
594               response = generalLogin(response, rs);
595               updateLastDeviceIP(statement, rs.getInt("userid"), request, getClientIP(
                      httpRequest));
596             }else{
597               String insert_userinfo = "insert into userinfo(facebookid,email,firstname,
                      lastname,phone)"+
598                     " values (?,?,?,?,?)";
599               ps= connection.prepareStatement(insert_userinfo, Statement.
                      RETURN_GENERATED_KEYS);
600             ps.setString(1, facebookid);
601           ps.setString(2, email);
602           ps.setString(3, firstname);
603           ps.setString(4, lastname);
604           ps.setString(5, phone);
```

```
605         int count=ps.executeUpdate();
606         if(count==1){
607           ResultSet rs_userid = ps.getGeneratedKeys();
608           if(rs_userid.next()){
609             int userid = rs_userid.getInt(1);
610             response.put("result", "Success: Facebook Login!");
611                 response.put("userid", userid);
612                 response = defaultUser(response);
613                 updateLastDeviceIP(statement, userid, request, getClientIP(httpRequest));
614           }
615         }else{
616           response.put("result", "Failure: Facebook Login - Get userid!");
617         }
618           }
619       } catch (Exception e) {
620         e.printStackTrace();
621         try {
622           response.put("result", "Error: Facebook Login!");
623         } catch (JSONException e1) {
624           e1.printStackTrace();
625         }
626       }finally{
627         dbDisconnect(connection, statement, rs, ps);
628       }
629         return response;
630       }
631       //Google+ Login
632       @POST
633       @Path("gplogin")
634       @Consumes(MediaType.APPLICATION_JSON)
635       @Produces(MediaType.APPLICATION_JSON)
636       public JSONObject gpLogin(JSONObject request, @Context HttpServletRequest httpRequest)
              {
637         System.out.println("Google+ Login!");
638         Connection connection = null;
639         Statement statement = null;
640         ResultSet rs = null;
641         PreparedStatement ps = null;
642
643         JSONObject response = new JSONObject();
644         try {
645           String googleplusid = request.getString("googleplusid");
646           String email = request.getString("email");
647           String firstname = request.getString("firstname");
648           String lastname = request.getString("lastname");
649           String phone = request.getString("phone");
650           connection = dbConnect();
651           statement = connection.createStatement();
652             rs = statement.executeQuery(String.format
653                 ("select * from userinfo where googleplusid='%s'", googleplusid));
654           if(rs.next()){
655             response.put("result", "Success: Google+ Login!");
656             response = generalLogin(response, rs);
```

```
657             updateLastDeviceIP(statement, rs.getInt("userid"), request, getClientIP(
                    httpRequest));
658           }else{
659             String insert_userinfo = "insert into userinfo(email,firstname,lastname,
                    googleplusid,phone)"+
660                     " values (?,?,?,?,?)";
661             ps= connection.prepareStatement(insert_userinfo, Statement.
                    RETURN_GENERATED_KEYS);
662           ps.setString(1, email);
663         ps.setString(2, firstname);
664         ps.setString(3, lastname);
665         ps.setString(4, googleplusid);
666         ps.setString(5, phone);
667         int count = ps.executeUpdate();
668         if(count==1){
669           ResultSet rs_userid = ps.getGeneratedKeys();
670           if(rs_userid.next()){
671             int userid = rs_userid.getInt(1);
672             response.put("result", "Success: Google+ Login!");
673                 response.put("userid", userid);
674                 response = defaultUser(response);
675                 updateLastDeviceIP(statement, userid, request, getClientIP(httpRequest));
676           }
677         }else{
678           response.put("result", "Failure: Google+ Login - Get userid!");
679         }
680           }
681     } catch (Exception e) {
682       e.printStackTrace();
683       try {
684         response.put("result", "Error: Google+ Login!");
685       } catch (JSONException e1) {
686         e1.printStackTrace();
687       }
688     }finally{
689       dbDisconnect(connection, statement, rs, ps);
690     }
691       return response;
692     }
693   //UpGeneralSetting - Update general settings
694     @POST
695     @Path("upgeneralsetting")
696     @Consumes(MediaType.APPLICATION_JSON)
697     @Produces(MediaType.TEXT_PLAIN)
698     public String upGeneralSetting(JSONObject request){
699       System.out.println("UpGeneralSetting!");
700       Connection connection = null;
701       Statement statement = null;
702       ResultSet rs = null;
703       PreparedStatement ps = null;
704
705       try {
706         int userid = request.getInt("userid");
```

```
707         boolean ifcsmode = request.getBoolean("ifcsmode");
708         boolean ifturnoffbt = request.getBoolean("ifturnoffbt");
709         int savelength = request.getInt("savelength");
710         int lowbpm = request.getInt("lowbpm");
711         connection = dbConnect();
712         statement = connection.createStatement();
713         int count = statement.executeUpdate
714             (String.format(
715                 "update userinfo set ifcsmode=%s, ifturnoffbt=%s, savelength=%s, lowbpm=%s
                        where userid=%s",
716                 String.valueOf(ifcsmode), String.valueOf(ifturnoffbt),
717                 String.valueOf(savelength), String.valueOf(lowbpm), String.valueOf(userid)
                    ));
718         return count==1 ? "Success: UpGeneralSetting!" : "Failure: UpGeneralSetting!";
719       } catch (Exception e) {
720         e.printStackTrace();
721         return "Error: UpGeneralSetting!";
722       }finally{
723         dbDisconnect(connection, statement, rs, ps);
724       }
725       }
726       //UploadApk
727       @POST
728       @Path("uploadapk")
729       @Consumes(MediaType.MULTIPART_FORM_DATA)
730       @Produces(MediaType.TEXT_PLAIN)
731       public String uploadApk(@FormDataParam("apkFile") InputStream uploadedInputStream,
732           @FormDataParam("apkFile") FormDataContentDisposition fileDetail) {
733         System.out.println("UploadApk!");
734
735         if(fileDetail.getFileName().equals("")) return "No file selected!";
736         if(new File(apkFileLocation).listFiles().length > 0){
737           new File(apkFileLocation).listFiles()[0].delete();
738         }
739         try {
740         OutputStream output = new FileOutputStream(new File
741             (apkFileLocation + "/" + fileDetail.getFileName()));
742         int read = 0;
743         byte[] bytes = new byte[1024];
744         while ((read = uploadedInputStream.read(bytes)) != -1) {
745           output.write(bytes, 0, read);
746         }
747         output.flush();
748         output.close();
749         return "Success: UploadApk!";
750       } catch (Exception e) {
751         e.printStackTrace();
752         return "Error: UploadApk!";
753       }
754       }
755       //GetCurVersion
756       @GET
757       @Path("getcurversion")
```

```
758     @Produces(MediaType.TEXT_PLAIN)
759     public String getCurVersion() {
760       System.out.println("GetCurVersion!");
761
762       String fileName = new File(apkFileLocation).listFiles()[0].getName();
763       return fileName.replace("HeartCarer_", "").replace(".apk", "");
764     }
765     //GetApk
766     @GET
767     @Path("getapk")
768     @Produces(MediaType.APPLICATION_OCTET_STREAM)
769     public Response getApk() {
770       System.out.println("GetApk!");
771
772       File apkFile = new File(apkFileLocation).listFiles()[0];
773       return Response
774           .ok(apkFile)
775           .header("Content-Disposition",
776               ContentDisposition.type("attachment").fileName(apkFile.getName()).build())
777           .build();
778     }
779     //Send feedback
780     @POST
781     @Path("sendfeedback")
782     @Consumes(MediaType.APPLICATION_JSON)
783     @Produces(MediaType.TEXT_PLAIN)
784     public String sendFeedback(JSONObject request){
785       System.out.println("Send feedback!");
786       Connection connection = null;
787       Statement statement = null;
788       ResultSet rs = null;
789       PreparedStatement ps = null;
790
791       String curTime = new SimpleDateFormat("yyyy-MM-dd_HH-mm-ss", Locale.getDefault())
792         .format(new Date(System.currentTimeMillis()));
793       try {
794         int userid = request.getInt("userid");
795         String feedback = request.getString("feedback");
796         connection = dbConnect();
797         statement = connection.createStatement();
798         String insert_userinfo = "insert into userfeedback(userid,feedback,sendtime)
                values (?,?,?)";
799         ps = connection.prepareStatement(insert_userinfo, Statement.RETURN_GENERATED_KEYS)
                ;
800         ps.setInt(1, userid);
801       ps.setString(2, feedback);
802       ps.setString(3, curTime);
803       int count = ps.executeUpdate();
804         return count==1 ? "Success: Send feedback!" : "Failure: Send feedback!";
805     } catch (Exception e) {
806       e.printStackTrace();
807       return "Error: Send feedback!";
808     }finally{
```

```
809       dbDisconnect(connection, statement, rs, ps);
810     }
811     }
812     /**
813      * Add JsonObject with default settings
814      *
815      * @param response: JsonObject to add default settings
816      * @return JsonObject with default settings added
817      */
818     private JSONObject defaultUser(JSONObject response) {
819       try {
820         response.put("emergencynum", "");
821           response.put("emergencymes", "");
822       response.put("ifsendsms", false);
823       response.put("ifappendloc", false);
824         response.put("ifcsmode", false);
825       response.put("ifturnoffbt", true);
826       response.put("savelength", 60000);
827       response.put("lowbpm", 40);
828     } catch (JSONException e) {
829       e.printStackTrace();
830     }
831     return response;
832   }
833     /**
834      * Parse results got from database and put them into JsonObject when logging in
835      *
836      * @param response: JsonObject sent to client eventually
837      * @param rs: Result got from database query
838      * @return JsonObject with result from database added
839      */
840     private JSONObject generalLogin(JSONObject response, ResultSet rs) {
841       try{
842         response.put("userid", rs.getInt("userid"));
843         response.put("emergencynum", rs.getString("emergencynum"));
844         response.put("emergencymes", rs.getString("emergencymes"));
845         response.put("ifsendsms", rs.getBoolean("ifsendsms"));
846         response.put("ifappendloc", rs.getBoolean("ifappendloc"));
847         response.put("ifcsmode", rs.getBoolean("ifcsmode"));
848         response.put("ifturnoffbt", rs.getBoolean("ifturnoffbt"));
849         response.put("savelength", rs.getInt("savelength"));
850         response.put("lowbpm", rs.getInt("lowbpm"));
851       }catch(Exception e){
852         e.printStackTrace();
853       }
854     return response;
855   }
856     private void updateLastDeviceIP(Statement statement, int userid,
857       JSONObject json, String lastip) throws Exception {
858       String curTime = new SimpleDateFormat("yyyy-MM-dd_HH-mm-ss", Locale.getDefault())
859         .format(new Date(System.currentTimeMillis()));
860     statement.executeUpdate(String.format
861         ("update userinfo set lastdevice='%s', lastip='%s', lasttime='%s', version='%s',
```

```
            imei='%s' where userid=%s",
862         json.getString("lastdevice"), lastip, curTime, json.getString("version"), json
                .getString("imei"), String.valueOf(userid)));
863   }
864   //Get client IP address
865     private String getClientIP(HttpServletRequest request) {
866     return request.getHeader("X-Real-IP");
867   }
868   //Connect to database
869     private Connection dbConnect() throws Exception{
870       try {
871         if(initContext==null || ds==null){
872           initContext = new InitialContext();
873             ds = (DataSource)initContext.lookup("java:comp/env/jdbc/TestDB");
874         }
875         return ds.getConnection();
876     } catch (Exception e) {
877       e.printStackTrace();
878       System.out.println("Error: dbConnect()!");
879       throw new Exception();
880     }
881     }
882     //Disconnect database
883     private void dbDisconnect(Connection connection, Statement statement,
884         ResultSet rs, PreparedStatement ps){
885       try {
886         if(connection != null) connection.close();
887     } catch (Exception e) {
888       e.printStackTrace();
889       System.out.println("Error: dbDisconnect()!");
890     }
891       try {
892         if(statement != null) statement.close();
893     } catch (Exception e) {
894       e.printStackTrace();
895       System.out.println("Error: dbDisconnect()!");
896     }
897       try {
898         if(rs != null) rs.close();
899     } catch (Exception e) {
900       e.printStackTrace();
901       System.out.println("Error: dbDisconnect()!");
902     }
903       try {
904         if(ps != null) ps.close();
905     } catch (Exception e) {
906       e.printStackTrace();
907       System.out.println("Error: dbDisconnect()!");
908     }
909     }
910 }
```

Below are the Matlab source codes of the Compressed Sensing reconstruction algo-

rithm.

```
function [a, y_tmp] = myCoSaMP(Phi, u)
[N,d] = size(Phi);
s = d/8; % Sparsity 256
a = zeros(d,1); % a0
v = u;
for i = 1:15
    y = Phi' * v;
    [tmp, index] = sort(abs(y), 'descend');
    Omega = index(1:s/8);
    [tmp, index] = sort(abs(a), 'descend');
    T = union(Omega, index(1:s));
    b = zeros(d, 1);
    a = b;
    b(T) = Phi(:, T)\u;
    [tmp, index] = sort(abs(b), 'descend');
    if i==1
        y_tmp = index(1:s);
    end
    a(index(1:s)) = b(index(1:s));
    v = u - Phi*a;
end
```

Bibliography

[1] A. Maton, J. Hopkins, C. W. McLaughlin, S. Johnson, M. Q. Warner, D. LaHart and J. D. Wright, "Human Biology and Health," *Pearson Prentice Hall*, 1993.

[2] American Heart Association, "Heart Disease and Stroke Statistical Update 2014," *Circulation*, 2013.

[3] M. HC, M. CA and G. SS, "Preventing heart disease in the 21st century: implications of the Pathobiological Determinants of Atherosclerosis in Youth (PDAY) study," *Circulation*, 117(9), 1216-27, 2008.

[4] A. Kumar, "ECG - simplified," *LifeHugger*, Feb. 2010.

[5] J. Masters, C. Bowden and C. Martin, "The Textbook of Veterinary Medical Nursing," *Butterworth-Heinemann*, Jan. 2003.

[6] G. Lanza, "The electrocardiogram as a prognostic tool for predicting major cardiac events," *Progress in Cardiovascular Disease*, vol. 50, pp. 87-111, 2007.

[7] G. Chen and D. Kotz, "A Survey of Context-Aware Mobile Computing Research," *Computer Science Technical Report TR2000-381*, Department of Computer Science, Dartmouth College, Dartmouth

[8] American Heart Associations, "Management of Patients With Ventricular Arrhythmias and the Prevention of Sudden Cardiac Death," *ACC/AHA/ESC Pocket Guideline Based on the ACC/AHA/ESC 2006 Guidelines*, Sept. 2006.

[9] P. R. Kowey and D. Z. Kocovic, "Ambulatory Electrocardiographic Recording," *American Heart Association*, ISSN: 0009-7322, Online 72514.

[10] K. HL and P. J. Podrid, "Role of Holter monitoring and exercise testing for arrhythmia assessment and management," *Cardiac Arrhythmia Philadelphia*, Pa:Lippincott Williams & Wilkins, pp. 165-193, 2001.

[11] A. Kadish, A. Buxton and H. Kennedy, "ACC/AHA clinical competence statement on electrocardiography and ambulatory electrocardiography: a report of the ACC/AHA/ACP-ASIM task force on clinical competence," *Circulation*, 104: pp. 3169-3178, 2001.

[12] M. H. Crawford *et al.*, "ACC/AHA guidelines for ambulatory electrocardiography," *Journal of the American College of Cardiology*, vol.34, Issue 4, pp. 1262-1347, Oct. 1999.

[13] M. Sung, C. Marci and A. Pentland, "Wearable Feedback Systems for Rehabilitation," *Journal of Neuro Engineering and Rehabilitation*, MIT, Jun. 2005.

[14] R. Fensli, E. Gunnarson and O. Hejlesen, "A Wireless ECG System for Continuous Event Recording and Communication to a Clinical Alarm Station," *26th Annual International Conference of the IEEE*, vol. 1, pp. 2208-2211, Sept. 2004.

[15] S. E. de Lucena *et al.*, "ECG Monitoring Using Android Mobile Phone and Bluetooth," *2015 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, pp. 1976-1980, 2015.

[16] S. H. Liu, Y. F. Huang and C. R. Chen, "The Wireless Holter ECG System Based on Zigbee," *2014 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pp. 1816-1820, 2014.

[17] C. Thanawattano, R. Pongthornseri and S. Dumnin, "Wearable Wireless ECG Sensor with Cross-platform Real-Time Monitoring," *2012 IEEE EMBS Conference on Biomedical Engineering and Sciences (IECBES)*, pp. 284-287, 2012.

[18] L. Kai *et al.*, "A System of Portable ECG Monitoring Based on Bluetooth Mobile Phone," *2011 International Symposium on IT in Medicine and Education (ITME)*, pp. 309-312, 2011.

[19] Bluetooth SIG, "Specification Documents," May 2012.

[20] Texas Instruments Inc., "Bluetooth Low Energy," 2012.

[21] J. Decuir, "Bluetooth 4.0: Low Energy," CSR plc, 2010.

[22] R. Davidson *et al.*, "Getting Started with Bluetooth Low Energy," O'Reilly Media Inc., May 2014.

[23] A. Dementyev *et al.*, "Power Consumption Analysis of Bluetooth Low Energy, ZigBee and ANT Sensor Nodes in a Cyclic Sleep Scenario," *2013 IEEE International Wireless Symposium (IWS)*, pp. 1-4, 2013.

[24] M. Siekkinen *et al.*, "How Low Energy is Bluetooth Low Energy? Comparative Measurements with ZigBee/802.15.4," *2012 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, pp. 232-237, 2012.

[25] M. Collotta and G. Pau, "A Novel Energy Management Approach for Smart Homes Using Bluetooth Low Energy," *2015 3rd International Conference on Future Internet of Things and Cloud (FiCloud)*, pp. 558-563, 2015.

[26] R. Faragher and R. Harle, "Location Fingerprinting With Bluetooth Low Energy Beacons," *IEEE Journal on Selected Areas in Communications*, vol. 33, pp. 2418-2428, 2015.

[27] G. Rajagopal *et al.*, "Low Cost Cloud Based Intelligent Farm Automation System Using Bluetooth Low Energy," *2014 IEEE Region 10 Humanitarian Technology Conference (R10-HTC)*, pp. 127-132, 2014.

[28] E. J. Candes and T. Tao, "Near-Optimal Signal Recovery From Random Projections: Universal Encoding Strategies?," *IEEE Transactions on Information Theory*, vol. 52, pp. 5406-5425, 2006.

[29] D. L. Donoho, "Compressed sensing," *IEEE Transactions on Information Theory*, vol. 52, pp. 1289-1306, 2006.

[30] E. J. Candes and M. B. Wakin, "An Introduction To Compressive Sampling," *IEEE Signal Processing Magazine*, vol. 25, pp. 21-30, 2008.

[31] S. S. Chen, D. L. Donoho and M. A. Saunders, "Atomic decomposition by basis pursuit," *SIAM Journal of Scientific Computing*, vol. 20, pp. 33-61, 1998.

[32] J. A. Tropp and A. C. Gilbert, "Signal Recovery From Random Measurements Via Orthogonal Matching Pursuit," *IEEE Transactions on Information Theory*, vol. 53, pp. 4655-4666, 2007.

[33] R. E. Carrillo and K. E. Barner, "Iteratively re-weighted least squares for sparse signal reconstruction from noisy measurements," *43rd Annual Conference on Information Sciences and Systems (CISS)*, pp. 448-453, 2009.

[34] E. J. Candes, J. Romberg and T. Tao, "Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information," *IEEE Transactions on Information Theory*, vol. 52, pp. 489-509, 2006.

[35] E. J. Candes and T. Tao, "Decoding by linear programming," *IEEE Transactions on Information Theory*, vol. 51, pp. 4203-4215, 2005.

[36] D. Schneider, "New camera chip captures only what it needs," *IEEE Spectrum*, vol. 50, pp. 13-14, 2013.

[37] M. H. Firooz and S. Roy, "Network Tomography via Compressed Sensing," *IEEE Global Telecommunications Conference (GLOBECOM)*, pp. 1-5, 2010.

[38] Z. Jin and Y. P. Du, "Application of partial-echo compressed sensing in MR angiography," *2012 5th International Conference on Biomedical Engineering and Informatics (BMEI)*, pp. 26-29, 2012.

[39] W. Hou and C. Zhang, "A compressed sensing approach to low-radiation CT reconstruction," *2014 9th International Symposium on Communication Systems, Networks & Digital Signal Processing (CSNDSP)*, pp. 793-797, 2014.

[40] A. Y. Yang *et al.*, "Fast ℓ_1-Minimization Algorithms for Robust Face Recognition," *IEEE Transactions on Image Processing*, vol. 22, pp. 3234-3246, 2013.

[41] D. L. Marks *et al.*, "Computational photography and compressive holography," *IEEE International Conference on Computational Photography (ICCP)*, pp. 1-8, 2010.

[42] A. M. M. Scaife and Y. Wiaux, "The application of compressed sensing techniques in radio astronomy," *2011 XXXth URSI General Assembly and Scientific Symposium*, pp. 1-4, 2011.

[43] L. F. Polania and K. E. Barner, "A weighted ℓ_1 minimization algorithm for compressed sensing ECG," *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4413-4417, 2014.

[44] L. F. Polania *et al.*, "Compressed sensing based method for ECG compression," *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 761-764, 2011.

[45] A. Smith, "U.S. Smartphone Use in 2015," [Online], Apr. 2015. Available: http://www.pewinternet.org/2015/04/01/us-smartphone-use-in-2015/

[46] International Data Corporation (IDC), "Smartphone OS Market Share, 2015 Q2," [Online], Aug. 2015. Available: http://www.idc.com/prodserv/smartphone-os-market-share.jsp

[47] Texas Instruments Inc., "Bluetooth Low Energy CC2540/41 Mini Development Kit User's Guide," Dec. 2012.

[48] T. Barill, "An ECG Primer," *Nursecom Educational Technologies*, 2003.

[49] Texas Instruments Inc., "Bluetooth Low Energy Deep Dive," May 2011.

[50] Bluetooth SIG, "BLE 101 - Bluetooth Low Energy"

[51] Bluetooth SIG, "Heart Rate Profile," Jul. 2011.

[52] Bluetooth SIG, "Heart Rate Service," Jul. 2011.

[53] "Android (operating system)," [Online]. Available: https://en.wikipedia.org/wiki/Android_(operating_system)

[54] Google Inc., "Application Fundamentals," [Online]. Available: http://developer.android.com/guide/components/fundamentals.html

[55] Google Inc., "Fragments," [Online]. Available: https://developer.android.com/guide/components/fragments.html

[56] Google Inc., "Toasts," [Online]. Available: https://developer.android.com/guide/topics/ui/notifiers/toasts.html

[57] Google Inc., "App Widgets," [Online]. Available: http://developer.android.com/guide/topics/appwidgets/index.html

[58] "Web Services Glossary," [Online]. Available: https://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice

[59] "Types of Web Services," [Online]. Available: http://docs.oracle.com/javaee/6/tutorial/doc/giqsx.html

[60] P. Macherla, "Types of Web Services - Big and RESTful," [Online]. Available: http://theopentutorials.com/post/uncategorized/types-of-web-services-big-and-restful/

[61] "Method Definitions," [Online]. Available: https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html

[62] "Jersey - RESTful Web Services in Java," [Online]. Available: https://jersey.java.net/

[63] "Chapter 3. JAX-RS Application, Resources and Sub-Resources," [Online]. Available: https://jersey.java.net/documentation/latest/jaxrs-resources.html

[64] B.-M. Block and P. Mercorelli, "Conceptual understanding of complex components and Nyquist-Shannon sampling theorem: A design based research in Engineering," *2015 IEEE Global Engineering Education Conference (EDUCON)*, pp. 462-470, 2015.

[65] A. Mishra *et al.*, "ECG Signal Compression Using Compressive Sensing and Wavelet Transform," *34th Annual Int. Conf. of the IEEE EMBS*, 2012.

[66] A. M. R. Dixon *et al.*, "Compressed Sensing System Considerations for ECG and EMG Wireless Biosensors," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 6, pp. 156-166, 2012.

[67] Z. Zhang *et al.*, "Compressed Sensing for Energy-Efficient Wireless Telemonitoring of Noninvasive Fetal ECG Via Block Sparse Bayesian Learning," *IEEE Transactions on Biomedical Engineering*, vol. 60, pp. 300-309, 2013.

[68] S. M. S. Jalaleddine *et al.*, "ECG data compression techniques: A unified approach," *IEEE Trans. Biomed. Eng.*, vol. 37, no. 4, pp. 329-343, Apr. 1990.

[69] Y. Zigel *et al.*, "The weighted diagnostic distortion (WDD) measure for ECG signal compression," *IEEE Trans. Biomed. Eng.*, vol. 47, no. 11, pp. 1422-1430, Nov. 2000.

[70] R. Berinde *et al.*, "Combining geometry and combinatorics: A unified approach to sparse signal recovery," *46th Annual Allerton Conf. on Commun., Control and Computing*, pp. 798-805, 2008.

[71] H. Mamaghanian *et al.*, "Compressed Sensing for Real-Time Energy-Efficient ECG Compression on Wireless Body Sensor Nodes," *IEEE Trans. Biomed. Eng.*, vol. 58, no. 9, Sept. 2011.

[72] D. Needell and J. A. Tropp, "CoSaMP: Iterative Signal Recovery From Incomplete And Inaccurate Samples," *Information Theory and Applications*, Jan. 2008.

[73] T. Last *et al.*, "Multi-component based cross correlation beat detection in electrocardiogram analysis," *BioMedical Engineering OnLine*, Jul. 2004.