PASSIVE TIME SYNCHRONIZATION IN SENSOR NETWORKS USING OPPORTUNISTIC FM RADIO SIGNALS

An Honors Thesis

Presented in Partial Fulfillment of the Requirements for

the Degree Bachelor of Science with Distinction in Electrical and Computer

Engineering of The Ohio State University

By

Gerardo Balderas

*****

The Ohio State University

2011

Examination Committee:

Dr. Lee Potter

Dr. Rajiv Ramnath

ABSTRACT

Time synchronization is a critical piece of infrastructure for any wireless sensor network. It is necessary for applications such as audio localization, beam-forming, velocity calculation, and duplicate event detection. All of which require the coordination of multiple nodes.

Recent advances in low-cost, low-power wireless sensors have led to an increased interest in large-scale networks of small, wireless, low-power sensor nodes. Because of the more stringent power and cost requirements that this technology is driving, current time synchronization techniques must be updated to capitalize on these advances.

One time synchronization method developed specifically for wireless sensor networks is Reference Broadcast Synchronization. In RBS, a reference broadcast is transmitted to sensor nodes that require synchronization. Be recording the time of arrival, nodes can then use those time stamps to synchronize with each other.

This project aimed to make the RBS system even more robust, energy efficient, and cost effective by replacing the reference broadcast with an ambient RF signal (FM, TV, AM, or satellite signals) already prevalent in the environment. The purpose of this project was to demonstrate the viability of using Opportunistic RF synchronization by 1.) quantifying error, 2.) applying this synchronization method in a real world application,

and 3.), implementing a wireless sensor network using Android smart phones as sensor nodes.

Many of the objectives for the project were successfully completed. For convenience and economic reasons, an FM signal was chosen as the reference broadcast. FM Radio Synchronization error was then quantified using local FM Radio stations. The results of this experiment were very favorable. Using 5 second segments for correlation, total error was found to be 0.208±4.499μs. Using 3 second segments, average error was 2.33 ± 6.784μs. Using 400ms segments, synchronization error was calculated to be 4.76 ± 8.835μs. These results were comparable to sync errors of methods currently in widespread use.

It was also shown that Opportunistic RF Synchronization could be used in real world applications as well. Again FM was the RF signal of choice. FM Radio Synchronization was tested in an Audio Localization experiment with favorable results.

Implementation of an Android Wireless Sensor Network according to our specifications, however, could not be achieved. HTC EVO 4G's were programmed to communicate through TCP / IP network connections, record audio with a microphone, and to record FM Radio streams from the EVO's internal FM radio. Although recording these two sources separately as different data tracks was successful, simultaneous recording of these streams could not be accomplished (simultaneous recording is essential for Opportunistic RF Synchronization).

Although the Android smart phone implementation was not a total success, this project still provided data that supported the practical use of Opportunistic RF Synchronization.

ACKNOWLEDGEMENTS

I would like to thank my wife Marsha for her love, patience and help in the field. Her support has always been my inspiration.

I would like to thank and acknowledge Dr. Josh Ash for all of the guidance, patience, and collaboration he has had with me during this project. I learned much under his wing, and this project was the reason for the job offer from Intel.

I would like to thank and acknowledge Dr. Lee Potter for his guidance and leadership. He routinely went out of his way to make sure I was on the "right track."

Finally I would like to thank and acknowledge Alex Boytim for all of the training and support he has given me in field work. We collaborated in three data gathering experiments.

VITA

March 10, 1981……..…….Born – Del Rio, TX

1999-2001…….....Student, U.S. Naval Academy

2008 – present…….Student, The Ohio State University

FIELDS OF STUDY

Major Field: Electrical and Computer Engineering

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

1.1. PROBLEM

Time synchronization is a critical piece of infrastructure for any wireless sensor network. It is used for applications which require the coordination of multiple nodes. Some applications include audio localization, beam-forming, velocity calculation, and duplicate event detection. Recent advances in low-cost, low-power wireless sensors have led to an increased interest in large-scale networks of small, wireless, low-power sensor nodes. Because of the more stringent power and cost requirements that this technology is driving, current time synchronization techniques must be updated to capitalize on these advances.

One time synchronization method in widespread use is called Reference Broadcast Synchronization (RBS). Sensor nodes implemented by RBS record a portion of the broadcast signal that is transmitted from a local node. If any action has to be coordinated between sensor nodes, the reference signal recordings at each node are compared to each other to find a phase offset. This offset is the time difference needed to sync two nodes with each other. (Elson, Girod, & Estring, 2002)

Current RBS methods require the creation and propagation of a reference signal to be used by receivers for time synchronization. The scope of the system is thus limited by the range of the transmitter. This transmitter not only increases power requirements for the system, but it also creates an unnecessary foreign signal within the environment.

We propose a novel method that would drive down the power requirements of a system, increase its scope, and simply its synchronization to single-hop communication.

If a signal already present in the environment is used, a system can save on costs and power by "piggybacking" on its availability. Potential opportunistic signals include FM, TV, AM, or satellite signals already prevalent in the environment. Using an ambient RF signal, such as a radio wave, not only exploits the energy of an already existing transmitter, but it allows the sensor nodes to operate in a completely passive "only when needed" manner. All "chatter" is eliminated creating a stealthy system, and the passivity of the system increases the battery efficiency of the nodes.

1.2. OBJECTIVES

The purpose of this project was twofold. The first purpose was to demonstrate the viability of using a third party ambient RF signal (in this case, from a local FM Radio station) as a means for time synchronization in a sensor network environment.

The second purpose was to demonstrate the economic and convenient practicality of using this technique by implementing its use with popular multiple sensor platforms, Android smart phones.

The HTC EVO 4G, an Android smart phone used in this project, provides a plethora of sensors which includes an FM radio receiver, GPS receiver, Wi-Fi capability, Bluetooth capability, light sensor, accelerometer, compass, still / video camera, and a microphone. What makes the platform even more desirable to work with is that all the sensors are accessible by a Java API (Application Programming Interface). This not only simplifies equipment gathering and setup for the researcher, but it allows quick implementation of a wireless sensor network using nothing but off-the-shelf components.

To fulfill this purpose, three objectives were set forth. The first objective of the project was to quantify the error associated with the FM Radio time synchronization method. The second objective was to demonstrate the use of this time synchronization method with a well known wireless sensor network application, audio localization. The final objective was to design and implement a wireless sensor network composed of Android smart phones.

2. BACKGROUND

2.1. TIME SYNCHRONIZATION ERROR

The goal of time synchronization is to accurately represent data events from multiple sensors (or nodes) on one common timeline. This could mean synchronizing the clocks of nodes to match a global time, or it could mean finding the offsets necessary to synchronize nodes to a moment in time. Regardless of the time sync method, every data event can be represented by a time stamp based on the following equation:

$$\tau_{MEAS} = \tau_{TRUE} + e_{SYNC} + e_{MEAS} \quad (1)$$

where τ_{MEAS} denotes the measured time of event, which is composed of three components, τ_{TRUE}(the true time when the event occurred), e_{SYNC} (the error introduced by the synchronization error), and e_{MEAS}(the error introduced by the measurement method of the application).

Synchronization error is caused by the uncertainties of time that occur when a sender node sends a time stamp to a receiver node. The error can be broken down into four components (refer to Figure 1): Send Time, Access Time, Propagation Time, and Receive Time. (Sivrikaya & Yener, 2004)

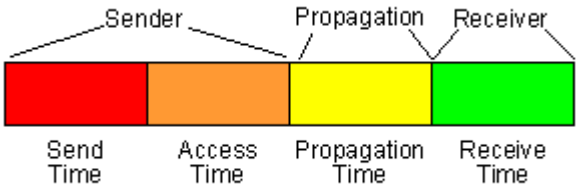

Figure 1: Message Uncertainties that contribute to Synchronization Error

Send Time: This error comes from time spent to construct the message to be sent. Uncertainties stem from scheduling within the operating system and from the time needed to transfer the message to the Network Interface Controller (NIC) for transmission.

Access Time: Each packet faces some delay at the medium access control (MAC) layer before actual transmission. The sources of this delay depend on the MAC scheme used, but some typical reasons for delay are waiting for the channel to be idle or for the time-division multiple access (TDMA) slot for transmission. (Sivrikaya & Yener, 2004)

Propagation Time: This is the amount of time it takes the message to travel from the sender to the receiver. In one method of time synchronization, it is assumed that a reference signal arrives at two sensor nodes at the same time. Because of the finite speed of the signal, there would be some unknown time difference between the arrivals. This is also attributed to this type of error.

Receive Time: This error comes from the time spent in decoding the message.

2.2. CHARACTERISTICS OF TIME SYNCHRONIZATION METHODS

In their paper, Sivrikay and Yener presented a broad set of characteristics used to measure the trade-offs between different synchronization methods. This helps in gauging

the benefits of the technique proposed in this project. The metrics are summarized as follows: (Sivrikaya & Yener, 2004)

Energy efficiency: Energy efficiency is one of the most important metrics for wireless sensor nodes. Since most nodes are battery powered, great care must be taken so that unnecessary transmissions between nodes are not made.

Scalability: Most sensor network applications need deployment of a large number of sensor nodes (some in the thousands). The synchronization method should be just as proficient with a higher number of nodes as with a few.

Precision: Precision is based on the application that the synchronization method is used. For some applications, accuracy down to the microsecond is needed whereas other applications would only require proper ordering of events.

Robustness: Since sensor networks are deployed in the field for long periods of time, sensor nodes have to survive the elements. The synchronization method should continue working even if a few sensor nodes become inoperable.

Lifetime: This requirement describes how long two nodes stay synchronized together after a time synchronization is done.

Scope: Scope describes the end result of synchronization and whether it is for a few local nodes or all the nodes in a topology. Some methods synchronize a global clock that all other nodes synchronize to. In other cases, finding time offsets and working with relative times is desired instead.

Cost and size: Sensor nodes are becoming smaller and more inexpensive. Therefore, it might be unrealistic to attach relatively large or expensive hardware (such as a GPS receiver) to a cheap device. The synchronization method should be economically viable as well.

Immediacy: Immediacy describes how quickly synchronization has to be processed. In some cases, a data event needs to be processed so quickly that the nodes should be pre-synchronized. (Sivrikaya & Yener, 2004)

With these requirements defined, a quick survey of current time synchronization techniques can now be better understood.

2.3. REFERENCE BROADCAST SYNCHRONIZATION

Reference Broadcast Synchronization (RBS) was designed specifically for Wireless Sensor Networks. Rather than synchronize nodes in the sender – receiver traditional fashion, RBS was designed to reduce a large portion of message error (refer to Figure 1) by eliminating the uncertainty of the sender and access time.

In RBS, one node transmits a reference signal (pulse) to two nodes in the surrounding area (refer to Figure 2). Upon receiving this pulse, each receiver records the time based on its internal clock. The two receivers then transmit their timestamps to the other. The difference between the two is considered the offset necessary to synchronize the two. (Roche, 2006)

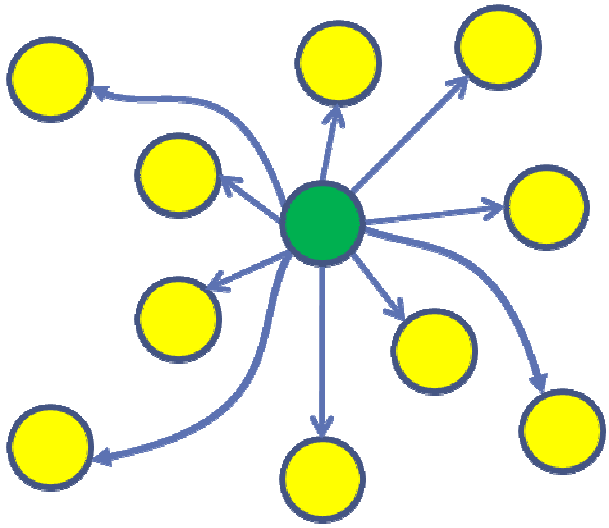

Figure 2: Example of Reference Broadcast Synchronization

Since the reference signal is an RF signal, it is assumed that the reference signal arrives at both nodes at the same time. This is considered a propagation time error, but for short distances (< 300m), it is negligible. (Elson, Girod, & Estring, 2002)

RBS has high precision, good energy efficiency, and good robustness. Scalability and scope become an issue with bigger topologies as multiple reference signal

transmitters are needed. RBS solutions become more complicated for multi-hop environments which occur as the topology is expanded. (Elson, Girod, & Estring, 2002)

2.4. TIMING-SYNC PROTOCOL FOR SENSOR NETWORKS

Timing-Sync Protocol for Sensor Networks (TPSN) is a time synchronization method based on Network Time Protocol (NTP) specifically geared towards Wireless Sensor Networks. Like NTP, sensor nodes synchronize by sender-receiver communications. To reduce the power drain from all of these communications, sensor nodes are assigned numbers based on a hierarchy of levels starting with the 0 level, or the root node.

This occurs during the Discovery Phase. The root node (selected first) transmits a level_discovery packet to the nearest nodes. This packet contains the level of the sender. When other nodes receive the packet, they are then assigned the next number as a level. Those nodes then send a new level_discovery packet to their neighbors. The process repeats until all nodes are assigned a level. (Refer to Figure 3)

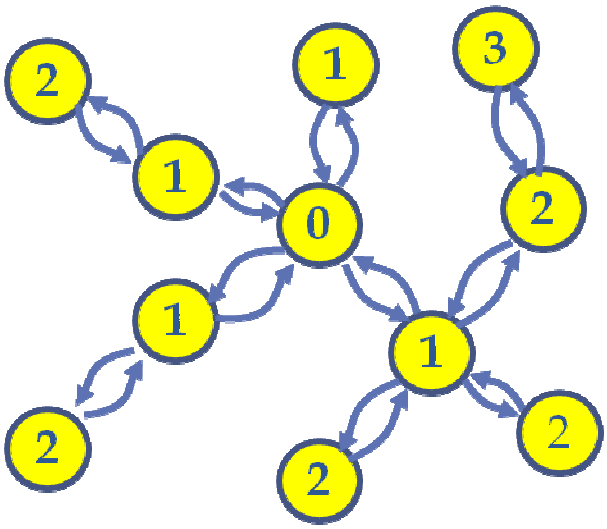

Figure 3: Example of Timing-Sync Protocol

During the Synchronization Phase, offsets between nodes are calculated as illustrated in Figure 4. Node A emits a synchronization_pulse to Node B. The packet contains a time stamp (T1) of its sending time. When Node B receives the message, the node records a timestamp (T2) and then sends back a new packet with a new timestamp, T1, and T2. When Node A receives the message, it then uses all four time stamps to find the offset necessary for Node A to synchronize with Node B.

The root node initiates this whole process by emitting a timing_pulse. This pulse tells all of the lower nodes to synchronize with the root node. After 2-way messaging is complete, level 2 nodes communicate with level 1 nodes to synchronize. The process repeats until all nodes are synchronized. (Ganeriwal, Kumar, & Srivastava, November 5-7, 2003)

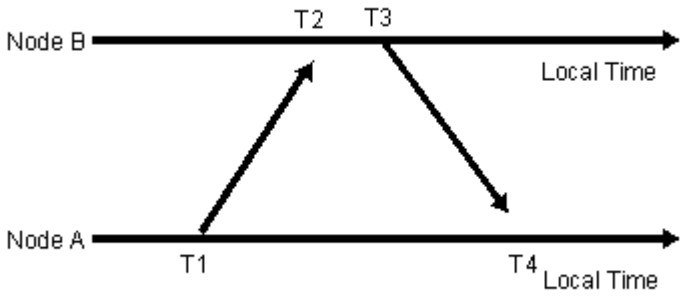

Figure 4: Two way communication between nodes

TPSN was designed as a multi-hop solution, so it can span very large topologies. It has the greatest scope (all nodes synchronized to a global clock), good scalability, good lifetime, and is argued by Ganeriwal to have better precision than Reference Broadcast

Synchronization. By applying time stamps in the MAC (Medium Access Control) level of hardware rather than the software level, sender and receiver error times (refer to Figure 1) are reduced to negligible amounts. Unfortunately the trade off is energy efficiency, system complication, and some robustness. A loss of a few key sensor nodes would require a new Level Discovery phase to be executed.

3. WIDER IMPACT OF OPPORTUNISTIC RF SYNCHRONIZATION

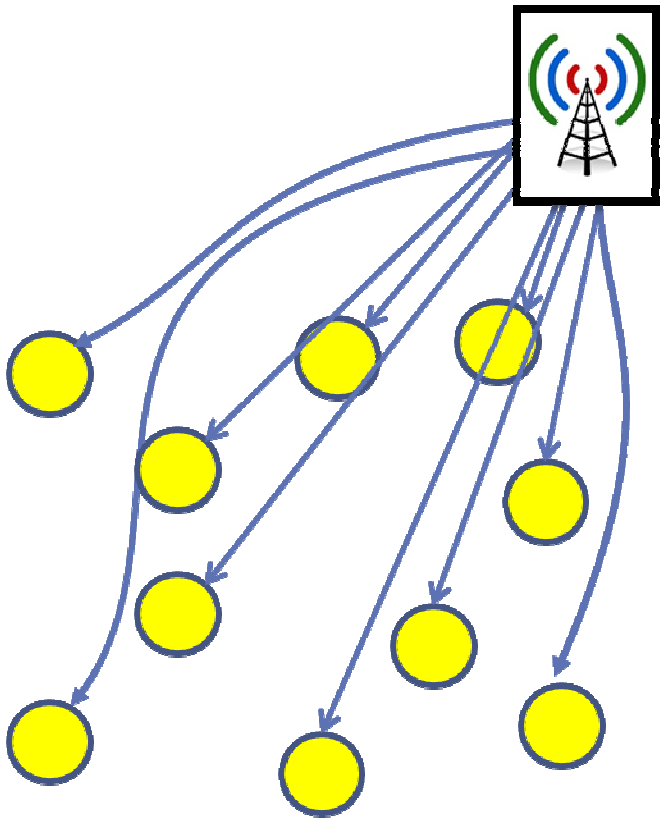

Figure 5: Example of Opportunistic RF Synchronization using an FM signal from a local radio station.

Opportunistic RF Synchronization operates much like RBS except there are a few key differences. In an RBS system, the scope of the network is limited by the range of the transmitter being used for the reference broadcast. By using a 3rd party transmitter, however, the scope of the system can theoretically be increased for miles for low economic (FM receivers for each node are inexpensive) and minimum power costs. With

this increased scope also comes simplicity. Compared with TPSN, the wider range of powerful ambient RF signals would allow a simpler implementation of large scale networks. The time synchronization in a wireless sensor network could easily become a single-hop operation.

The second key difference comes from the reference itself. Rather than a short pulse broadcast, an FM radio tower offers the luxury of a continuous broadcast 24 hours a day / 7 days a week. This increases the passivity and the immediacy of the system by allowing it to synchronize data exactly when an event of significance occurs.

4. OPPORTUNISTIC RF SYNCHRONIZATION IMPLEMENTED WITH FM RADIO

Opportunistic RF Synchronization is accomplished with nodes by correlating their reference broadcast recording with the reference broadcast recordings of other nodes. Since FM radio is prevalent in America and FM receivers are inexpensive, an FM radio signal is a practical choice for an RF reference.

In the following example, a sensor node is represented by a dual-channel recorder. The Left (L) channel is designated for the reference broadcast while the Right (R) channel is reserved for any signals of interest. This example pertains to an audio application, but Opportunistic RF Synchronization can be used with many other sensors. The only requirement is that an RF signal must be recorded at every node while the event of significance is recorded / time stamped.

When a data event occurs, a small snippet of the FM Radio broadcast is recorded and saved to the L Channel of a recorder. This is done for both sensors (refer to Figure

6). To synchronize the two nodes, a time offset has to be calculated. This is done by correlation of the L Channel data (refer to Figure 7). The time offset produced is then used to synchronize the data events recorded by the R Channels.

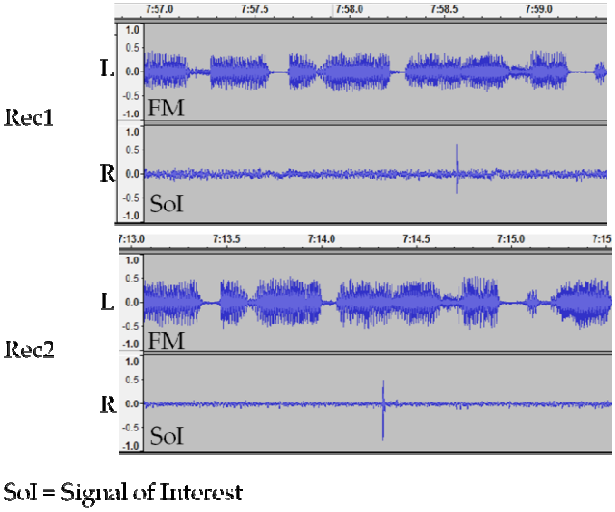

Figure 6: Unsynchronized Nodes (Recorders) with FM Radio Data and a Channel Recording Signal of Interest

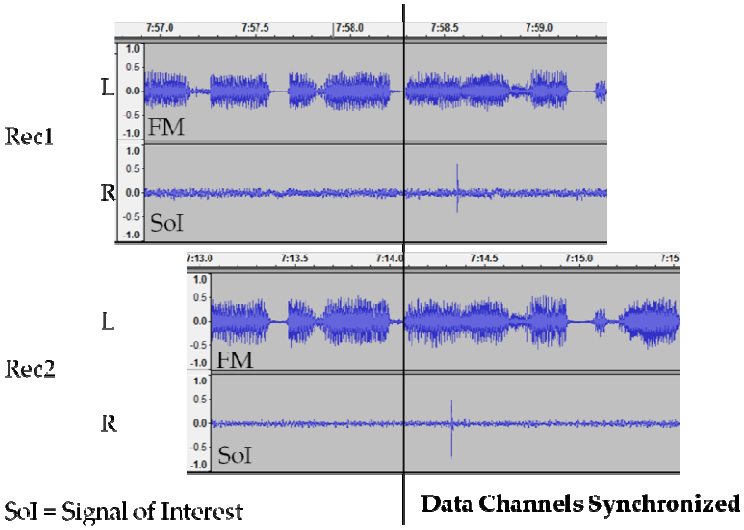

Figure 7: Synchronized Nodes (Recorders) with FM Radio Data. Notice how L Channels are "lined up". This is done by correlation and produces the offset necessary to synchronize R Channel data.

4.1. FM RADIO SYNCHRONIZATION ERROR EXPERIMENT

The following experiment (refer to Figure 8 for diagram of experimental setup) was conducted to quantify the error associated with FM Radio synchronization. It was conducted with two Audio MicroTrack II Portable Digital 2 Track Recorders, two Sanyo FM radios and a function generator. To measure the error, a common reference signal was generated and simultaneously recorded (48000 sample rate) by both recorders. The common reference was generated as a 10 VPP random signal which was fed into the Right Channel of each recorder. This acted as the control variable. Since the same signal arrived at both recorders at the exact same instant, correlating the two R Channel audio tracks with each other would produce the time offset necessary for time synchronization. (Patwari, Ash, Kyperountas, Hero III, Moses, & Correal, 2005)

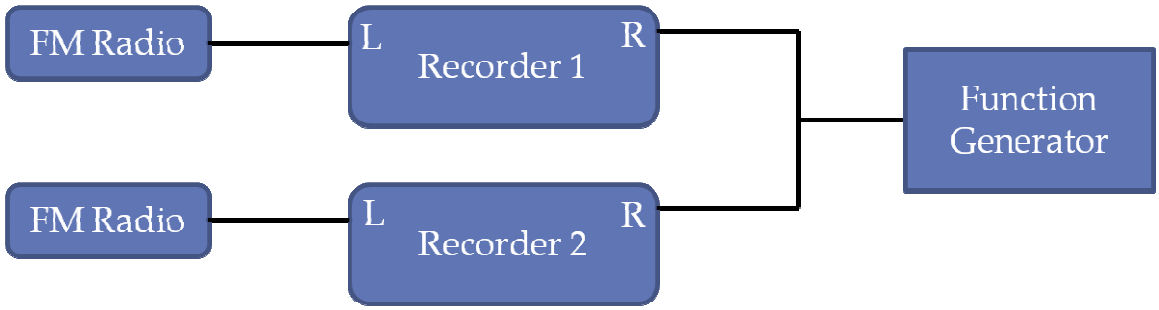

Figure 8: Experimental Setup for FM Radio Sync Error Analysis

At the same time, FM radio audio data was fed to the Left Channels of the recorders. In the ideal situation, the time offsets calculated from Right Channel correlation and from Left Channel correlation would both be the same. The difference found between the two offsets was the time synchronization error.

4.2. RESULTS

Matlab was used to gather and analyze the data for the experiment. Left Channel Data for both recorders was correlated to produce a time offset for time synchronization. The same was done with Right Channel data for both recorders. The two offsets calculated were then compared with each other to find the time difference or sync error. The correlation command used to do this (xcorr) returned results based on samples. The sample rate for all of the data recorded was consistent at 48000 samples per second. Many local radio stations were tested at differing lengths of FM Radio segments. For convenience sake, the error results in Figure 9 were converted from samples to time (microseconds). Each average was calculated with 100 trials.

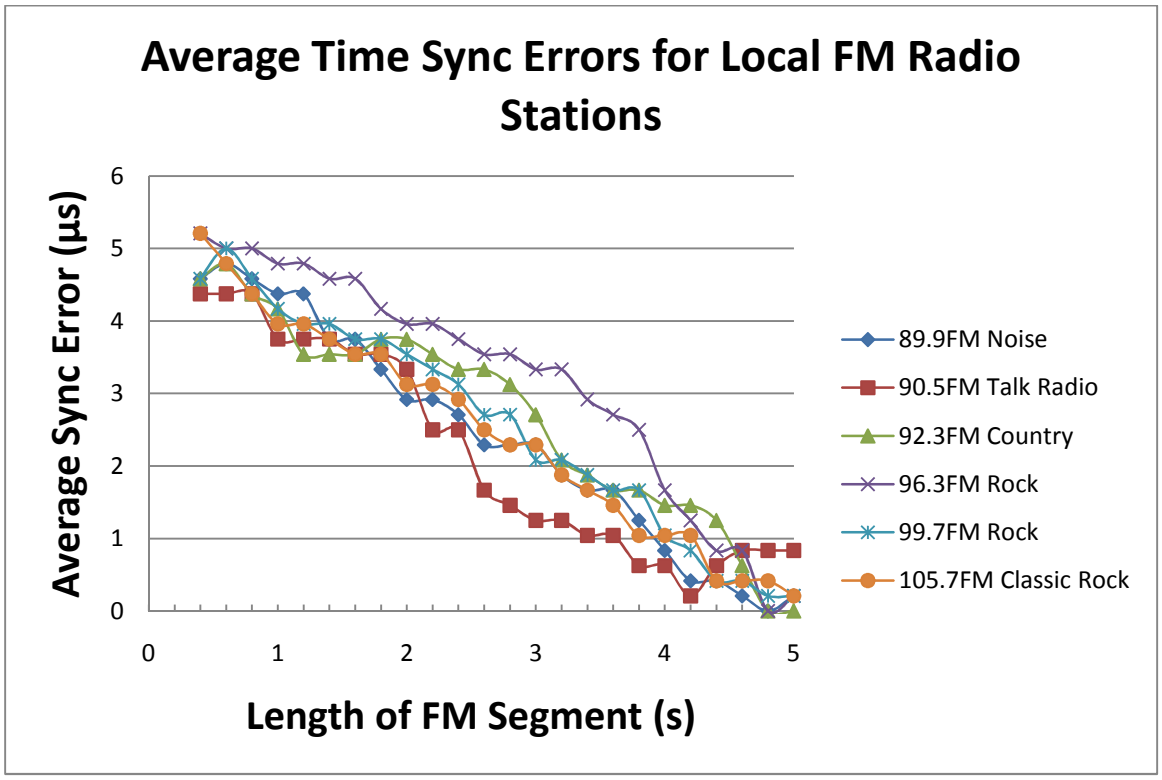

Figure 9: Average Time Sync Errors for Local FM Radio Stations. FM Radio correlations were done with increasing lengths of FM segments. Average calculated with 100 trials.

4.3. DISCUSSION

The results of this experiment were very favorable. Average errors were calculated to be less than 5µs for segments greater than 400ms. Using 5 second segments for correlation, total error was found to be 0.208±4.499µs. Using 3 second segments, average error was -2.33 ± 6.784µs. Using 400ms segments, synchronization error was calculated to be -4.76 ± 8.835µs. Comparing this to error results calculated by Elson, we can readily see that FM Radio Synchronization is a viable synchronization method. Elson showed that in a user-space implementation (he used two Compaq IPAQs using an 11Mbit 802.11 network) that RBS produced an error of 6.29 ± 6.45µs. When kernel timestamps were used instead, performance of RBS improved to 1.85 ± 1.28µs. These results are still very comparable to our results. (Elson, Girod, & Estring, 2002)

5. TIME DIFFERENCE OF ARRIVAL LOCALIZATION

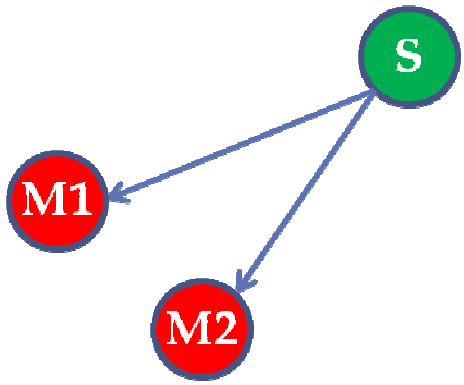

Figure 10: Diagram of Two Microphones listening to an Audio Source

Time Difference of Arrival (TDOA) Localization is a method which can estimate the direction (one microphone pair) or the coordinates (multiple microphone pairs) of an audio source (refer to Figure 10) by using a least squares cost function. This function uses the measured time differences for all combinations of local microphone pairs which

have recorded the audio source signal. (Patwari, Ash, Kyperountas, Hero III, Moses, & Correal, 2005)

In Figure 10 the time of arrival calculated at M1 for the signal emitted by S can be represented by the following equation:

$$\tau_1(\boldsymbol{S}) = \frac{\|\boldsymbol{M_1}-\boldsymbol{S}\|}{v} + \tau_0 \qquad (2)$$

where:

τ_0 = emission time of Audio Source

$\boldsymbol{S}$ = vector representing position of Audio Source

$\boldsymbol{M_1}$= vector representing position of microphone 1

v = speed of sound

Notice that the Time of Arrival is simply the sum of the emission time and the propagation time from source to sensor. The Time of Arrival at microphone 2 is calculated similarly.

$$\tau_2(\boldsymbol{S}) = \frac{\|\boldsymbol{M_2}-\boldsymbol{S}\|}{v} + \tau_0 \qquad (3)$$

By taking the difference between both Time of Arrivals, τ_0 is then cancelled. TDOA for any two microphones can then be represented by the following equation:

$$\tau_{ij}(\boldsymbol{S}) = \tau_i(\boldsymbol{S}) - \tau_j(\boldsymbol{S}) = \frac{\|\boldsymbol{M_i}-\boldsymbol{S}\|}{v} - \frac{\|\boldsymbol{M_j}-\boldsymbol{S}\|}{v} \qquad (4)$$

This equation is then applied to a least squares cost function:

$$\hat{\boldsymbol{S}} = \arg\min \sum_{ij} (\tau_{ij}(\boldsymbol{S}) - m_{ij})^2 \qquad (5)$$

where:

$\hat{\boldsymbol{S}}$ = Estimated Audio Source Position

m_{ij} = measured time difference between microphone *i* and microphone *j*

$\tau_{ij}(\boldsymbol{S})$ = expected time difference between microphone *i* and microphone *j*

5.1. LOCALIZATION APPLICATION

The following experiment was conducted to demonstrate the use of Opportunistic RF Synchronization in a real world application of wireless sensor networks, audio localization. In this experiment, the location of a pre-defined audio source was estimated by using five sensor nodes. Each sensor node was configured as represented by Figure 11.

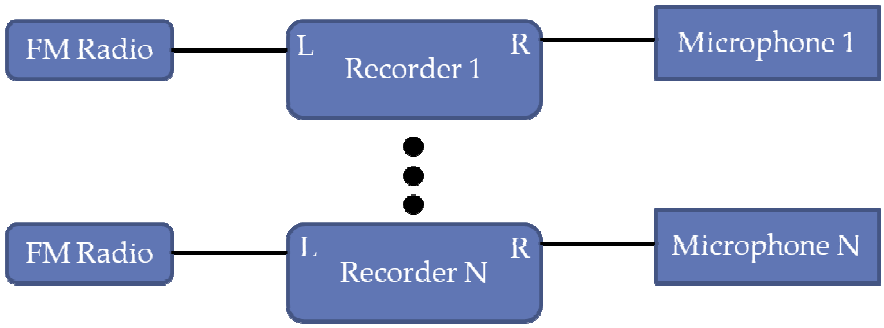

Figure 11: Sensor Node Configuration for TDOA Localization Experiment

Each recorder was connected to its own microphone and to its own FM Sanyo Radio. FM Radio data was fed into the Left channel of recorder while microphone data was fed into the Right channel of the recorder. Recorders were then set out at preset locations (refer to Figure 12). At each source location, the experimenter simulated gunshot blasts by clapping two pieces of plywood together in 5 second intervals. This

was done at each location 30 times. After the experiment, all .wav files recorded were downloaded to a computer for post processing by Matlab.

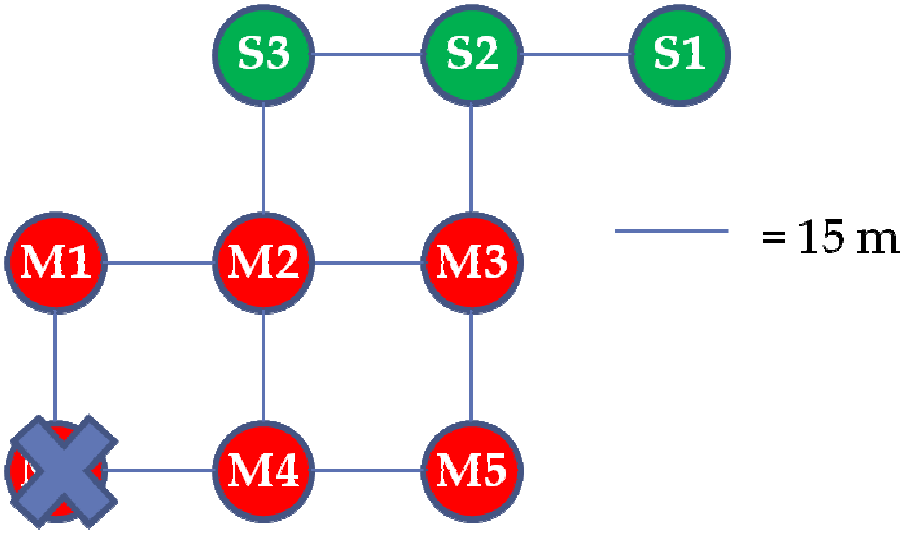

Figure 12: Layout of Sensor Node and Audio Source Locations. M numbered nodes are microphones. S numbered nodes are Audio Sources. One sensor node failed in the field.

5.2. DATA

Using Matlab, the TDOA's for all combinations of microphone pairs were measured for each clap. Correlations were taken from both L Channel and R Channel tracks to find the measured time difference between microphones.

$$m_{ij} = \frac{corr(R_i,R_j) - corr(L_i,L_j)}{sRate} \qquad (6)$$

where:

R_i= Right Channel Data for node *i*

L_i= Left Channel Data for node *i*

sRate = sample rate of data track (48000 samples / second)

These measurements were then used in the least squares cost function to find an estimate of the audio source. Those results can be seen in Figure 13.

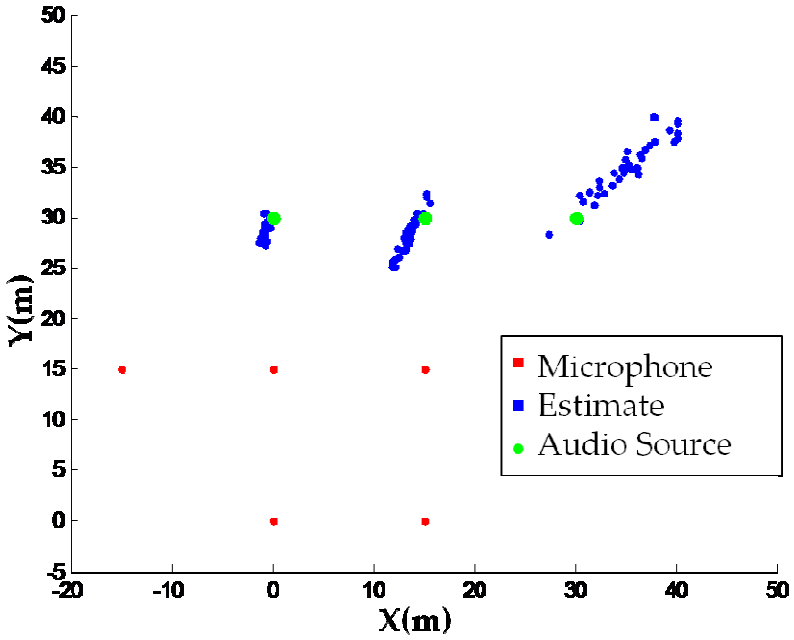

Figure 13: Results for TDOA Localization Experiment

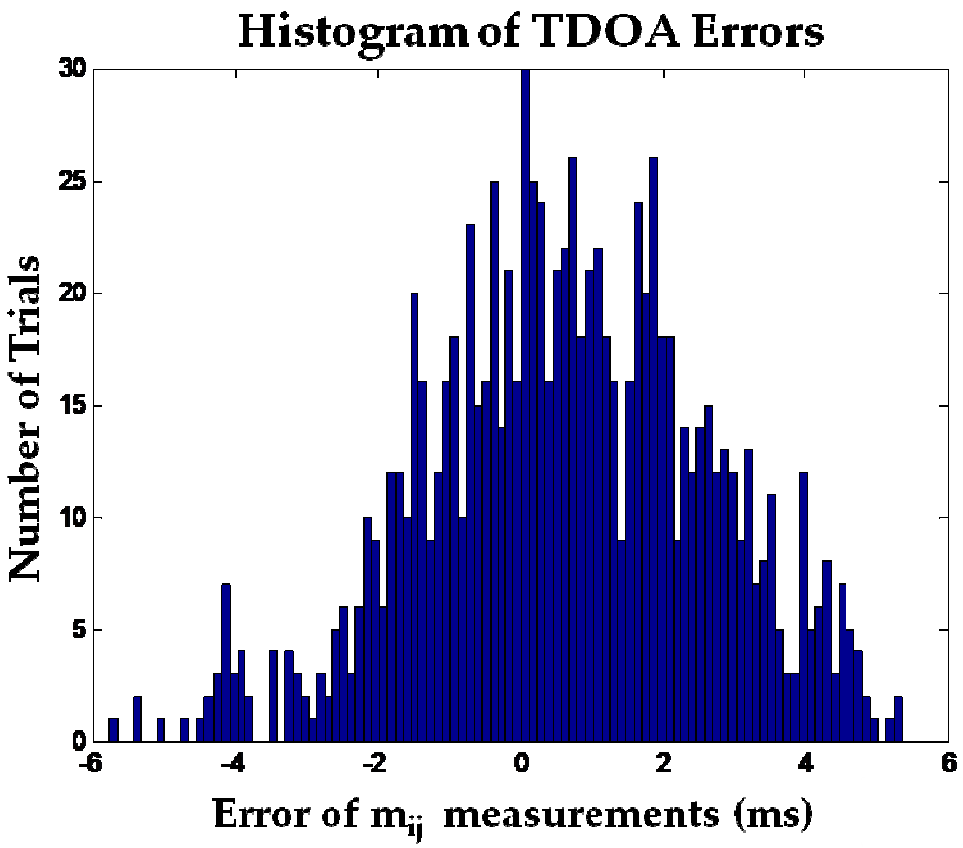

Figure 14: Histogram of Measured TDOA Error using buckets of size 120µs

5.3. DISCUSSION

As can be seen from Figure 13, the results were also very favorable. Based upon microphone topology, estimates fell within expected thresholds. (Patwari, Ash, Kyperountas, Hero III, Moses, & Correal, 2005) Based on temperature estimates (average temperature for Columbus, OH was 53.9 ˚F), a value of 338.599 m/s was used for the speed of sound. Mean error was 0.63052 ms while the standard deviation for error was 0.002s. When applying the standard deviation from this experiment to e_{TOTAL} and the standard deviation from the synchronization error e_{SYNC} to Equation 6, the majority of the error can be seen being contributed by e_{MEAS}. (Because of the small distances

between sensor nodes, any propagation delay error would equate to the tenths of a microsecond and is considered negligible compared to e_{TOTAL})

$$e_{TOTAL} = e_{SYNC} + e_{MEAS} \qquad (6)$$

$$2.0\text{ms} = 0.0068\text{ms} + e_{MEAS}$$

$$e_{MEAS}=1.993\text{ms} >> e_{SYNC}$$

This further validates the use of Opportunistic RF Synchronization and shows that any error found is negligible compared to measurement error. Encouraged by these results, the next phase of the project was undertaken.

6. IMPLEMENTATION USING ANDROID SMART PHONES

6.1. DESIGN PROCESS

The vision for the Android implementation was to have a wireless sensor network composed of Android mobile phones as sensor nodes. To achieve this, two objectives were set forth at the beginning of the design process (refer to Figure 15). First, the smart phones needed the capability to be controlled remotely by a master program installed on a laptop. Second, the phones had to record audio and FM radio simultaneously onto separate audio tracks. The phone was required to place all recorded mono audio data into the R Channel of a .wav file and it had to place all recorded FM radio data into the L Channel of a .wav file.

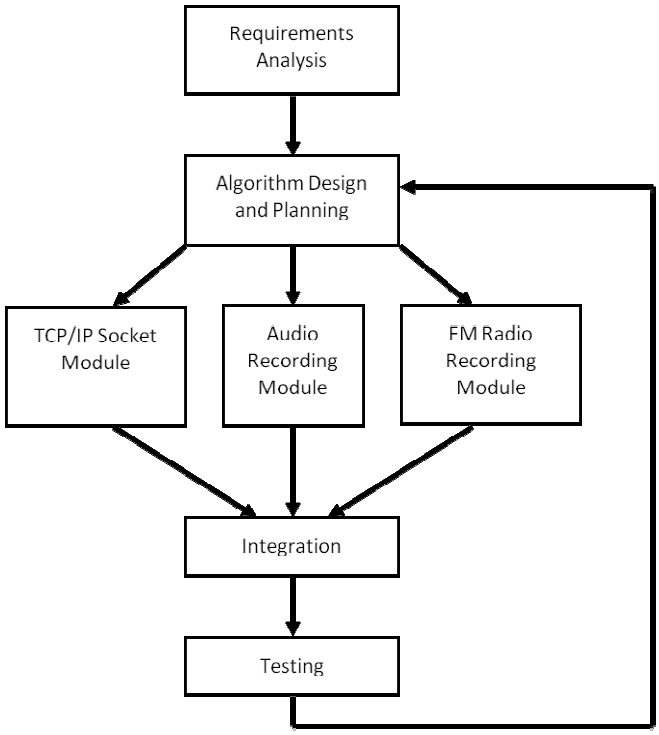

Figure 15: Design Process for OSU Recorder - Audio Localization Application for Android Smart Phone

To proceed with this type of implementation, the development portion of the project was divided up into three phases. Each phase or module would be programmed using the Android Java API (if possible) and all three would be integrated into a single application called OSU Recorder. The first module was the development of TCP/IP Remote Control Socket support for the program. The second phase was the development of the Audio Recording Module. This module was responsible for the microphone recording capability of the project. The last phase was programming the FM Radio Recording Module. This module was known to be more difficult to develop because there wasn't any direct programming access (Java API encapsulates internal functions of Android Operating System) to any recording capabilities of the FM Radio. After all modules were completed, integration and testing would then have commenced.

6.2. TCP / IP MODULE

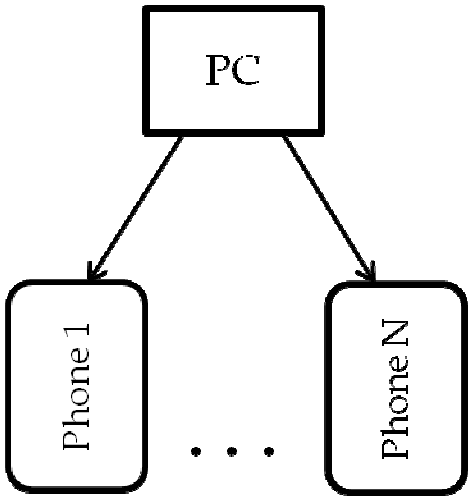

Figure 16: Diagram of Server / Client relationship used by OSU Recorder

The TCP/IP Module was responsible for communication between the server (laptop) computer and a client android device. The TCP/IP Module had two components. Server side software created a unique thread that opened a server socket connection for each mobile device. Server sockets allowed data transmission between the server (laptop) and the mobile devices. The client side software (installed on the phone) automatically attempted to connect to a default IP address. The user also had the option of changing the IP address as well. Once a client connected to the server, the server could remotely control the recording capabilities of the devices. Implementation was completed using Java API.

6.3. AUDIO RECORDING MODULE

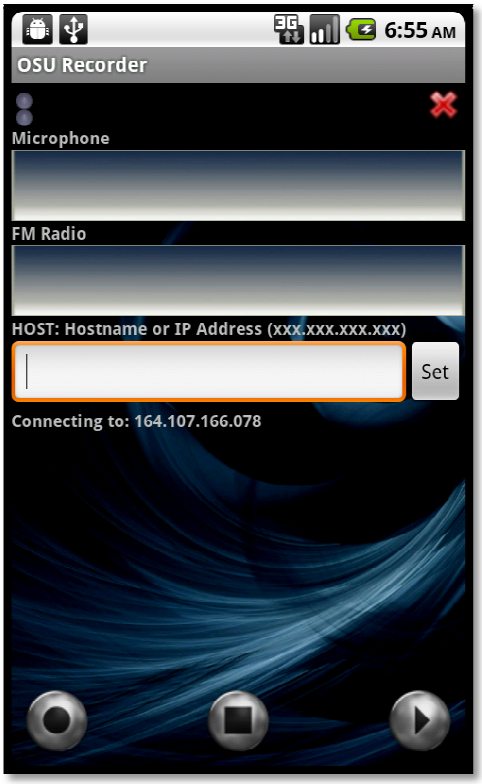

Figure 17: User Interface for OSU Recorder

The Audio Recording Module was the segment of code responsible for capturing audio data using the microphone built into the mobile device. First a Graphical User Interface (GUI) (refer to Figure 17) was developed to allow the user to record audio at the push of a button. OSU Recorder would store the audio as raw PCM data written to a .WAV formatted file for data analysis. Implementation was completed using the Java API.

6.4. FM RADIO RECORDING MODULE

The FM Radio Recording Module proved to be the most difficult part of the design process. This module was responsible for recording an FM radio signal into the R Channel of a .wav file. Since the Java API used by HTC to control the radio was

insufficient for recording, a more in depth look at the FM radio process of the HTC EVO 4G was needed.

The FM Radio capability of the HTC EVO 4G comes from the Broadcom BCM4329 integrated chip. This chip is responsible for Bluetooth, Wi-Fi, and FM Radio capabilities. (See Figure 19) At the highest level, the FM Radio is controlled through a program called HTCFMRadio.apk, a system application preinstalled on the mobile device. Files with an .apk extension are Android's version of a Java .jar file. HTCFMRadio.apk was written in Java, but the program acted as a "wrapper" program that simply called corresponding functions programmed in C, otherwise known as "native" code. (For a better idea of the Android OS Architecture, refer to Figure 20)

Even though Android was purported to be an open source platform, Google still gave 3rd party developers the ability to "hide" their code as compiled .so library files located within the Android framework. Since these files were HTC proprietary files, no source code was available to the public. On top of this no instruction set was publically available for the BCM4329. Getting access to the audio stream produced by the BCM4329 seemed like an impossibility. (Disassembling .so binary files by hand did not seem a practical solution.)

Fortunately, a solution was paved by the MIUI developer group, a hacker team dedicated to open source implementation of the MIUI ROM. ROM stands for "Read Only Memory", but in regards to Android development, a ROM is a custom version of the Android Operating System. They are very popular within the Android community because they personalize the user experience, and at times unlock features that are not

implemented or supported by commercial developers. The MIUI group modified an existing FM Radio program called FM.apk so that it could work with the BCM4325 (the predecessor to the BCM4329). FM.apk called its own open-source version of native code. Because of this source code, rudimentary access to radio functions within the BCM4329 became available. Unfortunately the MIUI ROM was in Chinese, but the program and code was quickly ported over to an American based MOD, CyanogenMOD. After installing CyanogenMOD and downloading the source code for the CyanogenMOD build, the FM Radio was open for study and manipulation.

In likewise fashion as the HTC versions, FM.apk acted as a "wrapper" program that called java methods in package android.hardware.fmradio. This package in turn called functions programmed in native code (C++) located in a library filed named AndroidRuntime.so. This library file was responsible for runtime operations and 3rd party implementations of hardware.

AndroidRuntime.so was compiled from Android_Runtime.cpp and a multitude of other C++ files. One of those files was android_hardware_fm.cpp. This file was responsible for the low-level commands that controlled the BCM4329 chip. In essence, AndroidRuntime.so was used to translate high level queries made by the user to low-level commands used by the HCI and I2s/PCM ports of the BCM4329 (Refer to Figure 8).

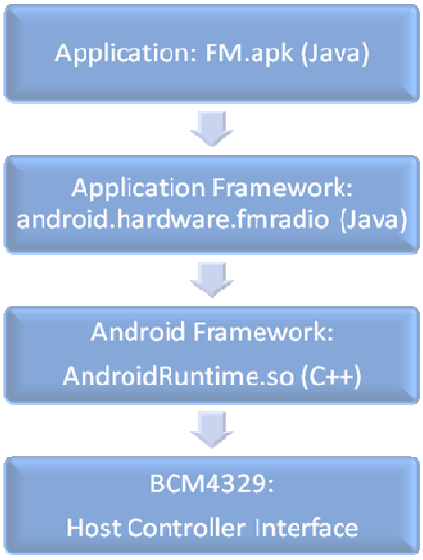

Figure 18: Process Flow for Software Operation of FM Radio implemented in CyanogenMod v6.0.1

Because of the complete access to the infrastructure of the Android Operating System framework (thanks to CyanogenMOD), changes were then made to the files so that the AudioRecord Java API could access FM Radio output as a recording input. This was done by following the chain of software beginning from the AudioRecord API down to the Audio Hardware Interface (refer to Figure 20) and enabling all recording references to the microphone of a connected headset. Since FM Radio on the HTC Evo 4G requires a headset to act as an antenna, the idea was to trick the software into thinking that a headset with a microphone was attached to the mobile device (even though no microphone was connected). With these tweaks, FM Radio recording was a success!

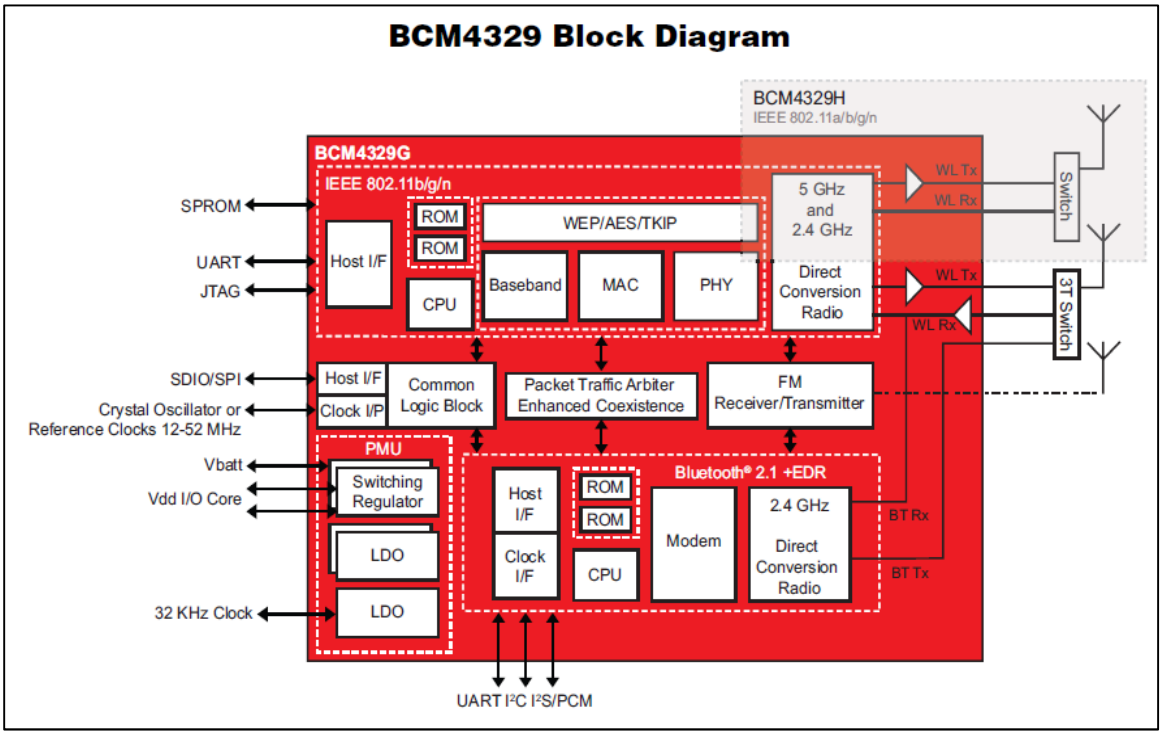

Figure 19: - Hardware Block Diagram for BCM4329 - Ports for FM Radio control are at bottom of figure.

6.5. INTEGRATION

After months of researching (the majority for FM Radio recording), all three components were finally completed. Integration was finally attempted. Unfortunately integration proved to be an impassable roadblock. The Android Operating System runs a service called AudioFlinger which directs all streaming inputs and outputs within the mobile device (refer to Figure 20 for Android Audio Architecture).

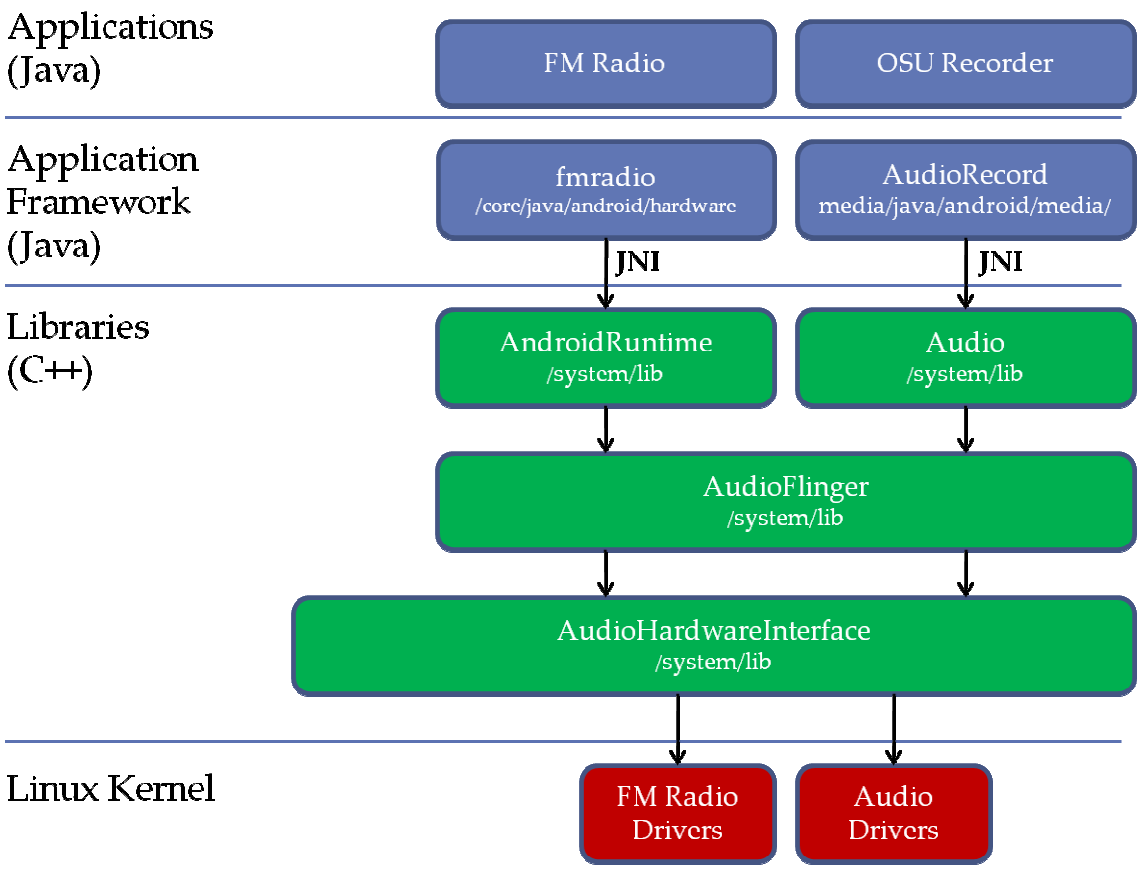

Figure 20: FM Radio and Audio Architecture within CyanogenMod v6.0.1

The Audio Flinger service controls multiple streams of output simultaneously, but the Android developers placed code in AudioFlinger.cpp which prevented more than one instance of an AudioRecord class from recording. The code was commented out and simultaneous recording was attempted anyway. Unfortunately, either one stream or the other would record, but not both simultaneously. After further research the task was deemed too cumbersome (multithreaded programming was essentially needed so that the buffer could alternately record FM Radio and audio) given the time constraints.

7. CONCLUSION

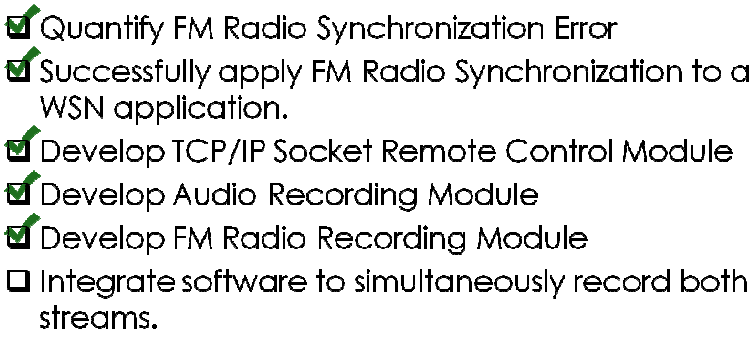

Figure 21: Checklist of Objectives Successfully Completed

Many of the objectives for the project were successfully completed (refer to Figure 21). FM Radio Synchronization error was quantified. Depending on the segment length used for time synchronization, sync error ranged from 5µs (400 ms) to less than 1µs (5 second lengths). These results supported the use of Opportunistic RF Synchronization.

It was also shown that FM Radio Synchronization could be used in real world applications with accuracy. FM Radio Synchronization was used in an Audio Localization experiment composed of five sensor nodes and three audio sources. The results were also favorable (refer to Figure 13).

Implementation of an Android Wireless Sensor Network according to our specifications could not be achieved, however. Developing a TCP/IP Socket Remote Control module, an Audio Recording Module, and an FM Radio Recording Module were all successful, but integration could not be completed. Unfortunately, either one stream

or the other would record, but not both simultaneously. After further research the task was deemed too difficult and cumbersome given the time constraints.

Although the end goal of Android implementation was not reached, results clearly showed the viability of using Opportunistic RF Synchronization. It is not only cost effective, but the accuracy of the system is comparable with other techniques currently in use.

IEEE STANDARDS

The BCM4329 comes equipped with Wi-Fi, Bluetooth, and FM receiver, all compliant with IEEE standards.

BIBLIOGRAPHY

Elson, J. (2003). *Time Synchronization in Wireless Sensor Networks.* PhD Thesis, University of California, Los Angeles.

Elson, J., Girod, L., & Estring, D. (2002). Fine-grained network time synchronization using reference broadcasts. *ACM SIGOPS Operating Systems Review - OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation , 36* (SI), 147 - 163.

Ganeriwal, S., Kumar, R., & Srivastava, M. B. (November 5-7, 2003). Timing-sync protocol for sensor networks. *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems.* Los Angeles: ACM.

Kirkwood, B. (2003). *Acoustic Source Localization Using Time-Delay Estimation.* M.Sc. thesis, Technical University of Denmark, Denmark.

Patwari, N., Ash, J., Kyperountas, S., Hero III, A., Moses, R., & Correal, N. (2005). Locating the nodes: cooperative localization in wireless sensor networks. *IEEE Signal Processing Magazine , 22* (4), pp. 54–69.

Roche, M. (2006, April 23). *Time Synchronization in Wireless Networks.* Retrieved May 11, 2011, from CSE574S: Advanced Topics in Networking: Wireless and Mobile Networking (Spring 2006) Website

Sivrikaya, F., & Yener, B. (2004). Time Synchronization in Sensor Networks: A Survey. *IEEE Network , 18* (4), 45-50.