Enhancing Plug and Play Capabilities in Body Area Network Protocols

A Major Qualifying Project Report:

Submitted to the Faculty

Of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Bachelor of Science

By:

Ryan Danas

Douglas Lally

Nathaniel Miller

John Synnott

Date: March 10, 2014

Approved:

Prof. Krishna Kumar Venkatasubramanian, Advisor

Prof. Craig A. Shue, Co-Advisor

Abstract

This project aimed to create a plug-and-play protocol for Body Area Networks (BANs). This protocol enables communication between a diverse number of devices and a base station, regardless of equipment manufacturer. Previous BANs rely on either proprietary software, or protocols that are specialized to the physical device. This protocol takes a more universal approach, allowing any arbitrary device to participate in a BAN without introducing any significant overhead or running cost to the operation of that BAN. Unlike previous approaches, any existing motes and the base station will not have to be updated. Only new devices being added to the BAN will have to implement the protocol before connecting. Our protocol introduces overhead that reduced the performance and lifetime of the motes used in our BAN.

Table of Contents

Contents

List of Figures

List of Tables

1. Introduction

Body area networks (BANs) are formed by a group of devices, usually sensors, connected wirelessly in order to provide some benefit to a user. Compared to other types of networks, BANs are extremely short range, limited to the space occupied by one person. BANs are most often used for medical applications; by attaching several types of sensors to an individual, different types of vital signs can be gathered and then analyzed by a medical professional. As the sensors are wireless and usually small, the person can maintain a mobile lifestyle while still keeping track of their medical condition.

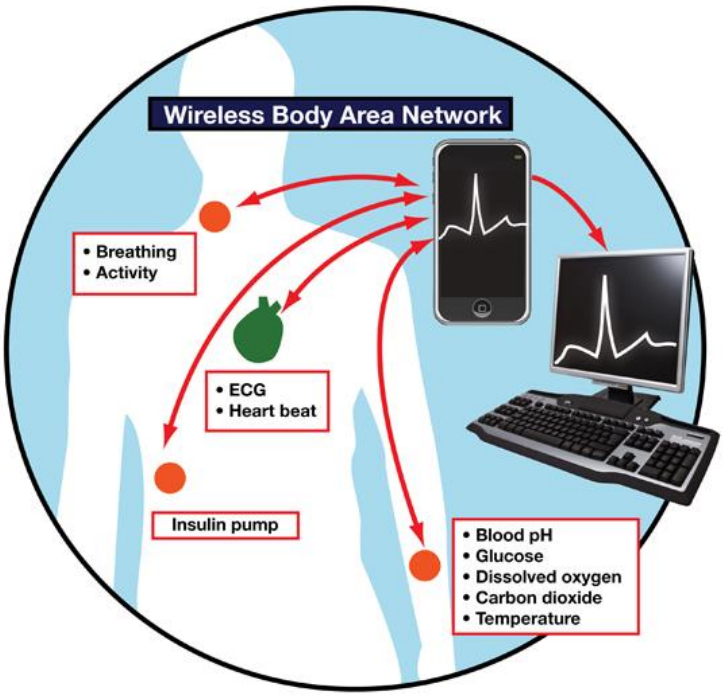


Figure 1: Wireless Body Area Network [1]

Previous BAN technologies have focused on creating a BAN for a specific purpose. CodeBlue [2] is still in development; however it uses a purpose-built set of motes and software to provide medical monitoring. The CodeBlue Technical Report [3] explains that their software is expected to work with the Mica2, MicaZ and Telos mote platforms, along with some specialized hardware platforms. The report does not suggest any method for integrating other mote platforms. The report also suggests CodeBlue was designed only to support a particular set of medical devices. Expanding the set of supported motes does not seem to be a major focus of the BAN design.

Although the technology, sensors, and wireless protocols have been evolving steadily over the last decade, there has not yet been a BAN designed for hardware flexibility. The goal of this MQP is to create a protocol that would allow a BAN to be "plug-and-play"; where sensors can be easily added and removed while minimizing the effort required by the user to reconfigure the BAN. This protocol is hardware agnostic, requiring only that a device be able to implement this protocol before being added to the BAN. This would minimize the time between the release of a new device, and when that device could be integrated in a BAN, while limiting the amount of work required support the changes. Additionally, the protocol is flexible to the hardware changes, not requiring protocol updates every time the BAN is modified.

We achieved such a protocol by changing which device is the keeper of functional information: as in which sensors are available, what each sensor is capable of, and what data can be sampled from each sensor. Some previous BANs [4] embed parts of this information in the base station, and other parts in the motes. Particular identifiers in messages would tell both the mote and base station which functionality this message applies to. We have moved all of this functional information to the mote. The mote inherently needs to know what sensors it has, what they are capable of, and what data it can sample from its sensors. However, now that this information is not assumed by the base station, there has to be a mechanism for the motes to communicate parts of this information to the base station. Most of the plug and play protocol is designed to allow motes to teach the base station. To communicate the most complex information, we had to define micro languages and grammars, which the base station is responsible for parsing. These grammars should support teaching the base station almost anything that could possibly be communicated. If not, the grammars can be expanded, as long as the discussing parties understand the changes.

Introducing this sort of flexibility comes at the price of performance. Based on our test results, a BAN operating with this protocol will have lower throughput, shorter battery life, and latency associated with the base station learning about the motes. Depending on the hardware being used and lifetime requirements, our protocol may not be viable in every situation. However, it is technically capable of operating in any realistic scenario.

Overall, we believe our body area network protocol has taken some major steps towards a practical, plug and play BAN. The most important of these design goals are as follows:

- An application layer protocol that does not inherently rely on static message identifiers.
- A protocol that can support new sensors, motes, and commands without changes to the mote firmware or base station application.

- A flexible base station learning language that can be expanded easily through changes to a few grammars.
- A BAN platform that is flexible enough to support any type of research or real world application.

The rest of this paper is structured as follows: Section 2 describes the basis of our BAN platform. Section 3 presents our BAN system model. Section 4 describes our plug and plug BAN protocol. Section 5 details our performance testing, associated results, and highlights areas our testing did not cover. Section 6 provides an analysis of our plug and play protocol. Section 7 provides the conclusions of our work. Section 8 outlines areas of future work. Section 9 contains references. Finally, the appendices conclude the document.

2. BAN Hardware Investigation

Before implementing our plug and play protocol, we looked into several mote and base station platforms. The mote platforms needed to be flexible enough for our protocol nuances. Mote hardware with a wide array of open source documentation with a solid foundation was preferred. We also wanted the base station to be a device common to most of the public today. Smart accessories seemed to be the most common, practical, and flexible development platform. Following are our investigation results and hardware choices for this project.

2.1 Base Station Platform

The first priority for the project was selecting what platform the base station application was going to run on. The three top candidates were Apple's iOS, the Pebble watch, and Google's Android.

iOS [5] offered a visually pleasing interface, but beyond that was ill suited for what the project required. iOS has never provided good support for working close to hardware, and Apple itself is generally not willing to provide assistance. However, it is worth noting that an iOS version of the base station software could be created.

The Pebble [6] watch platform would have been a nice selection, because it would be easy to offer information at a glance, but both the hardware and software seemed inadequate for the purpose of a base station. The Pebble only offers Bluetooth connectivity, and so would require intermediate hardware

to communicate with devices using a different protocol. In addition, the hardware which is included in the Pebble is not very powerful, and may have issues with the type of processing that the project will require.

The platform that we decided to use was Android [7]. It runs on a multitude of devices, and would likely be able to run on hardware that is using multiple wireless protocols. Further, it has a robust developer community.

2.2 Sensing Platforms

Initial research into existing and available sensing platforms for usage and integration in the BAN was done in preparation for the project. Development of sensors and sensing platforms is beyond the scope of this project and therefore it was decided to make use of an existing platform. Of the relevant sensing platforms available today, Emotiv [8] and Shimmer [9] were the two major platforms considered for the project.

2.2.1 Emotiv

The wireless neuroheadset systems developed and marketed by Emotiv [8] were one of the main candidates we researched to potentially utilize as the sensing platform for the project. The two core products offered by Emotiv are the EPOC [10] and EEG [11] wireless neuroheadsets. Both systems consist of an array of sensors: 14 EEG sensors, 2 addition references, as well as a gyroscope. The purpose of these devices is to measure electrical activity in the brain of the device wearer. In addition to the sensors, Emotiv provides multiple SDK options [12] and a suite of software for detecting and interpreting headset sensor data. Through the analysis of the Emotiv systems we determined that it was not the ideal sensing platform for the project. A number of concerns were raised about the Emotiv products, some of which included: the capabilities and sensors offered by the headsets were not varied enough for what we wanted in the BAN, the wireless protocol implemented by the headsets was labeled as proprietary and the group wanted to utilize wireless technologies commonly used in body area networks such as ZigBee [13] and Bluetooth [14], Emotiv did not provide substantial information about their SDK and API, and the pricing and licensing options for the products were deemed too expensive or too restrictive. We compared the Emotiv sensing platform to the Shimmer sensing platform and ultimately decided to utilize the Shimmer.

2.2.2 Shimmer Sensing Platform

The Shimmer Wireless Sensing Platform has been a widely used Body Area Network platform for several years. The sensor motes run off of a Texas Instruments MSP430F1611 microcontroller [15] the MSP430 [16] line is a heavily used embedded system controller in both academic and commercial applications [17]. The stock firmware for these motes is TinyOS based. Daughter boards, or extension boards with additional sensors, can be added on to the main board for additional functionality. Additional sensors are separated into three groups: Kinematic sensors usually record inertial or movement measurements; Biophysical sensors measure personal medical data, such as heart rate; ambient sensors measure the world around you, such as temperature and humidity. Sensor options include: ECG, EMG, GSR, 9DoF, GPS, Strain Gauge, and Accelerometer. The Shimmer platform also comes with several base station platforms, such as stock LabView, Matlab, Android, and Windows applications [18].

For the most part, the Shimmer platform fits most of our needs. The sensor data and firmware is highly accessible and extensible; there is no layer of obscurity added for commercial benefits. There is a lot of software available freely after purchasing a kit, which should give us a good foundation to start development. The platform is popular and well received among most researchers who have used these motes [19]. Their site and social media is active, with frequent updates and further development. Sample documentation is highly detailed, and support resources are highly available, such as their Harvard hosted mailing list.

However, the platform is not perfect. Each main board only supports one daughter board each, meaning we will need several shimmer motes to support multiple extension sensors. Shimmer does not support the Bluetooth adapter they use, and any problems or questions must be answered by the third party manufacturer. The 802.15.4 and Bluetooth adapters cannot be used simultaneously, but can be used in tandem with clever firmware coding. The MSP430F1611 may be compact, but at the price of only having 48 KB of flash memory, and 10KB of RAM.

Overall, Shimmer is well received as a wearable medical platform and boasts more open source resources than its competitors such as Sunspot and the Telos platforms [20]. Even with these major concerns and preliminary thoughts, this device would be a great addition to the BAN. We decided this was the best mote platform for this project.

2.2.3 Operating System Platforms

Among the selection of operating systems, which were considered for potential use on the sensor devices, TinyOS [21], Contiki [22], and RIOT [23] stood out. TinyOS was initially developed as a project at UC Berkeley [24]. The operating system is open source and is developed and supported by a large community around the world [25]. The Contiki OS was created by Adam Dunkels [26] and is currently an open source software with a large development community [27]. The RIOT operating system was first detailed in two papers by Baccelli et al.: "RIOT: One OS to Rule Them All in the IoT" published in December 2012 [28], and "RIOT OS: Towards an OS for the Internet of Things" published in April 2013 [29]. RIOT OS launched publicly in 2013 and is an open source community project.

TinyOS was the first OS that was considered, simply because it is the oldest and most well-known. Being the oldest, it has been thoroughly tested, and appears solidly developed and implemented. TinyOS uses an event driven code style, which can help for creating clean code. An unfortunate side effect of this style is TinyOS' inability to use multithreading.

Contiki was originally developed by Adam Dunkels, who was aiming to connect low power devices to the Internet. Dunkels made heavy use of some of his previous projects in the development, leading to an operating system which functions differently from TinyOS. Nothing is event driven, instead favoring support for multithreaded programming. Special attention has been paid to power consumption as well. Contiki includes a low power implementation of IPv6 for wireless communication, and uses a low overhead implementation of multithreading called protothreads both of which were written by Dunkles. Unfortunately, low power implementations often sacrifice feature richness, and Contiki is no exception.

RIOT is similar to Contiki in its reliance on multithreading. However, RIOT uses a full implementation, rather than just protothreads. It also supports a wide range of protocols. Unfortunately these features can drain the battery life, and it's not entirely certain that we need the features that are offered.

In the end, TinyOS was chosen as the platform for further development. Because TinyOS has been around so long, the code has been well tested, and a strong community has had the time to build up. With a better community comes better support and by extension an easier time solving problems that will arise in development.

3. BAN System Model

Before designing our plug and play protocol, we needed to determine exactly how a BAN should behave. Creating a complete system model allowed us to evaluate the specific behavior of both the motes and base station in a BAN. We would later use this behavior as a guide when designing our protocol; any behavior represented by the model must be supported by the protocol. Our model is fairly high level, and deals with how the motes connect and disconnect to the base station, how to view data that the motes have sent to the base station, and how to run commands on the motes from the base station. No additional BAN capabilities are assumed or inferred beyond those described in the model; any BAN that conforms to this model is guaranteed to be capable of implementing and using our plug and play protocol.

3.1 BAN Workflow

The BAN workflow that was developed was made to provide an overview of what will need to be included in our application at different levels. In doing so, the goal was to streamline implementation and provide a detailed outline of how the BAN architecture would operate.

It includes a basic overview, including some specifics about what features the base station would ideally be capable of, but leaving out some important details which would need to be figured out when the time came to actually develop the framework. Additionally, everything from a user's perspective all the way down to details just shy of implementation is included. The workflow also covers basic features like setting up a BAN, to adding and removing motes, to receiving data, to tearing down a BAN, as well as more complex information such as how data will be displayed to the user in different contexts.

- Starting a BAN.
    - Gather devices which will be members of the initial BAN configuration.
    - Ensure that the base station software is installed on the device that you would like to use as the base station.
    - Launch the base station application.
    - An option will be presented to connect a mote to the BAN.
- Connecting a mote to the BAN.
    - The user selects the option to add a mote to the BAN.
    - If Bluetooth is not enabled on the base station, the user will be prompted to enable it.
    - The mote that you are trying to pair must be turned on and discoverable.
    - The base station will scan for any discoverable Bluetooth devices.

        - An option to re-scan for devices will be available.
    - The user is presented with a list of discovered devices.
    - The user selects a device from the list to begin that they want to add.
    - The base station pairs with the device.
        - Pairing is accomplished with Bluetooth SSP.
    - The device is added to a list of BAN members.
    - The user may follow the procedure outlined above to add up to 6 more devices (as limited by the Bluetooth piconet specification).
- Viewing BAN sensor data.
    - The user opens a list of available sensors and selects the one(s) to be displayed.
        - Availability is determined by which devices are turned on, have Bluetooth enabled, and have been paired and added to the BAN
    - The application displays the requested data to the user in numerical format that is updated in real-time.
    - The user can select an individual data value in order to view a graphical representation of the data from that sensor.
- Disconnecting a mote from the BAN.
    - The user opens a list of connected motes and selects a device to disconnect.
        - The mote that will be disconnected is not needed for this process.
    - The user confirms their wish to disconnect the device.
    - If the device is currently sending data to the base station, the base station will instruct the device to stop sending data.
    - The base station un-pairs from the device and the device is removed from the BAN member list.
- Purging a BAN.
    - The user selects the option to purge the BAN.
    - The user is warned that all connected devices will be unpaired.
        - The devices do not need to be present for the purge to happen.
    - The user confirms their wish to purge the BAN.
    - The base station instructs all connected devices that are sending data to stop sending data
    - The base station un-pairs from all devices that are part of the BAN and the devices are removed from the BAN member list.

3.2 BAN Design, State Machines & Transition Tables

The following diagrams and tables help separate behavior into a finite number of states, each of which hopefully pertain to the single responsibility principle. Once these behavioral states are defined, how each device transitions from state to state is the only thing left ambiguous. This can then be defined by figuring out what transitions exist, and what is necessary for each of them to occur.

To State ⇒ From State ⇓	1. Idle	2. Discoverable	3. Paired	4. Connected	5. Command & Inquiry	6. Streaming
1. Idle	N/A	Automatically transitions every DISCOVERABLE_DELAY * 2^DISCOVERABLE_BACKOFF seconds	N/A	N/A	N/A	N/A
2. Discoverable	Transitions back after DISCOVERABLE_TIME seconds	N/A	Base station requests to pair with the mote; a pairing request from any device is accepted	N/A	N/A	N/A
3. Paired	Base station unpairs from the mote.	N/A	N/A	Base station initiates a connection with the paired mote.	N/A	N/A
4. Connected	Base station unpairs from the mote, or the Base station does not request to secure the connection.	N/A	Connection between mote and base station is terminated.	N/A	A bluetooth connection is initiated by the base station with the mote.	N/A
5. Command & Inquiry	Base station unpairs from the mote.	N/A	Connection between mote and base station is terminated.	N/A	N/A	Base station requests mote sensor data.
6. Streaming	N/A	N/A	Connection between mote and base station is terminated.	N/A	Base station instructs mote to cease streaming data.	N/A

Table 1: Mote Transition Table

The mote has a total of six states: Idle, Discoverable, Paired, Connected, Command & Inquiry, and Streaming. In the idle state, the mote does nothing and no sensors are active. In the discoverable state, the mote will advertise its presence for pairing, as well as respond to scanning requests. The mote will transition back and forth between these states with an exponential backoff in order to preserve battery life. If the mote is reset, the exponential delay will also be reset. Once a base station pairs with the mote, the

mote will enter the paired state. At this point in time the mote remains undiscoverable and waits for the base station to either un-pair or for it to initiate a socket connection. If a connection is opened, the mote will enter the connected state. The mote will then automatically transition into the command & inquiry state. In this state, the mote waits to receive control messages from the base station. Once the mote gets the message, it will process it and send a response. One of the control messages will explicitly tell the mote to start sending sensor data. At that point the mote will enter the streaming state, and will no longer be configurable. If the mote needs additional configuration, it must be told to stop streaming and then the configuration details must be sent. This is done deliberately to maintain the single responsibility principle for each state. The mote will then leave these final states if the base station ever disconnects or un-pairs from the mote.

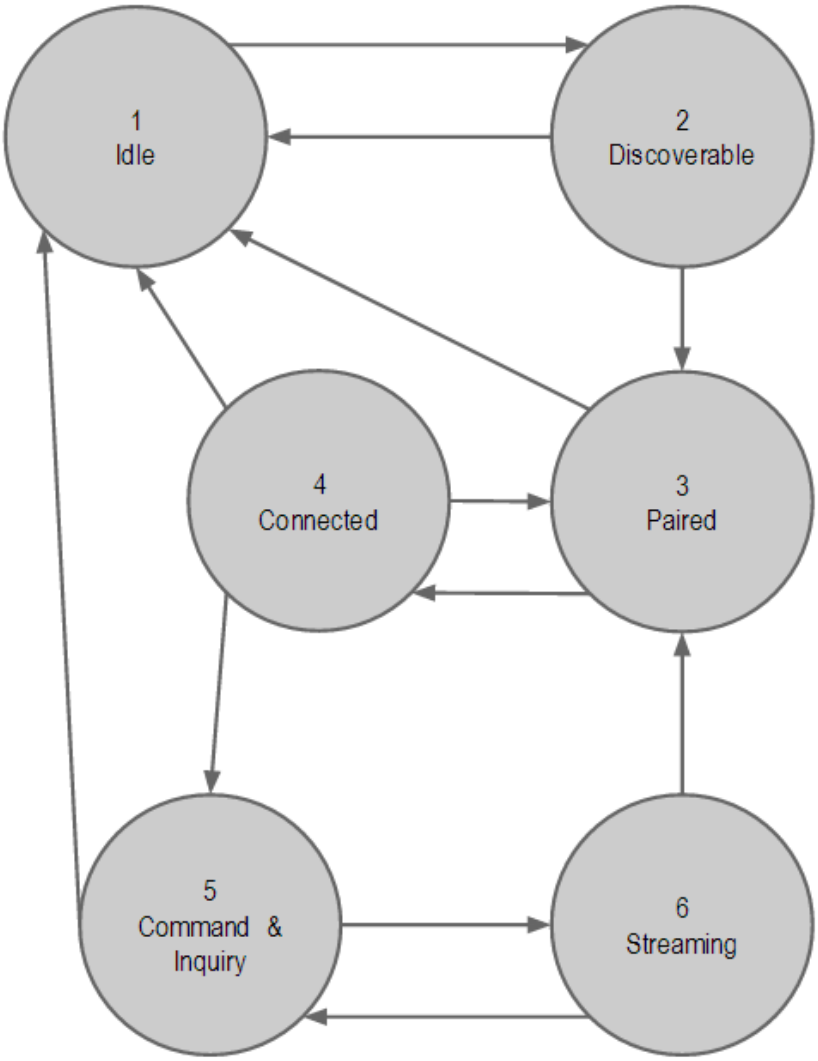


Figure 2: Mote State Machine

To State ⇒ From State ⇓	1. Idle	2. Discovery	3. Paired	4. Connected	5. Command & Inquiry	6. Mote Response	7. Mote Data
1. Idle	N/A	User initiates a scan for discoverable Bluetooth devices from the base station.	N/A	N/A	N/A	N/A	N/A
2. Discovery	Base station scanning phase has terminated.	N/A	User selects a discoverable device from a list of scanned deviced to initiate a pairing request with the selected device.	N/A	N/A	N/A	N/A
3. Paired	All paired motes are unpaired.	User initiates new scan for discoverable Bluetooth devices.	N/A	Base station initiates a connection with a paired mote.	N/A	N/A	N/A
4. Connected	Base station unpairs from the mote.	N/A	Connection between base station and a mote is terminated.	N/A	Mote accepts the base station's initiated bluetooth connection.	N/A	N/A
5. Command & Inquiry	Base station unpairs from the mote.	N/A	Connection between base station and a mote is terminated.	N/A	N/A	User selects to begin displaying sensor data from the paired mote(s).	Base station sends an inquiry or command to a mote.
6. Mote Data	N/A	N/A	Connection between base station and a mote is terminated.	N/A	User selects to stop displaying mote data on the base station.	N/A	N/A
7. Mote Response	N/A	N/A	N/A	N/A	Mote acknowledgement is received by the base station.	N/A	N/A

Table 2: Base Station Transition Table

The base station has a total of seven states: Idle, Discovery, Paired, Connected, Command & Inquiry, Mote Data, and Mote Response. The first five states act as the counterpart to the first five states of the mote; the only difference is that the base station is the active initiator to state transitions for the passive mote. The last two states convey the differing behavior between the mote and base station. In the mote data state, the base station can display the data being streamed from the mote. In the mote response state, the base station will simply wait for a response from the control message it just sent. The key difference here is that the transitions occur from user input on the base station, and acknowledgements from the mote.

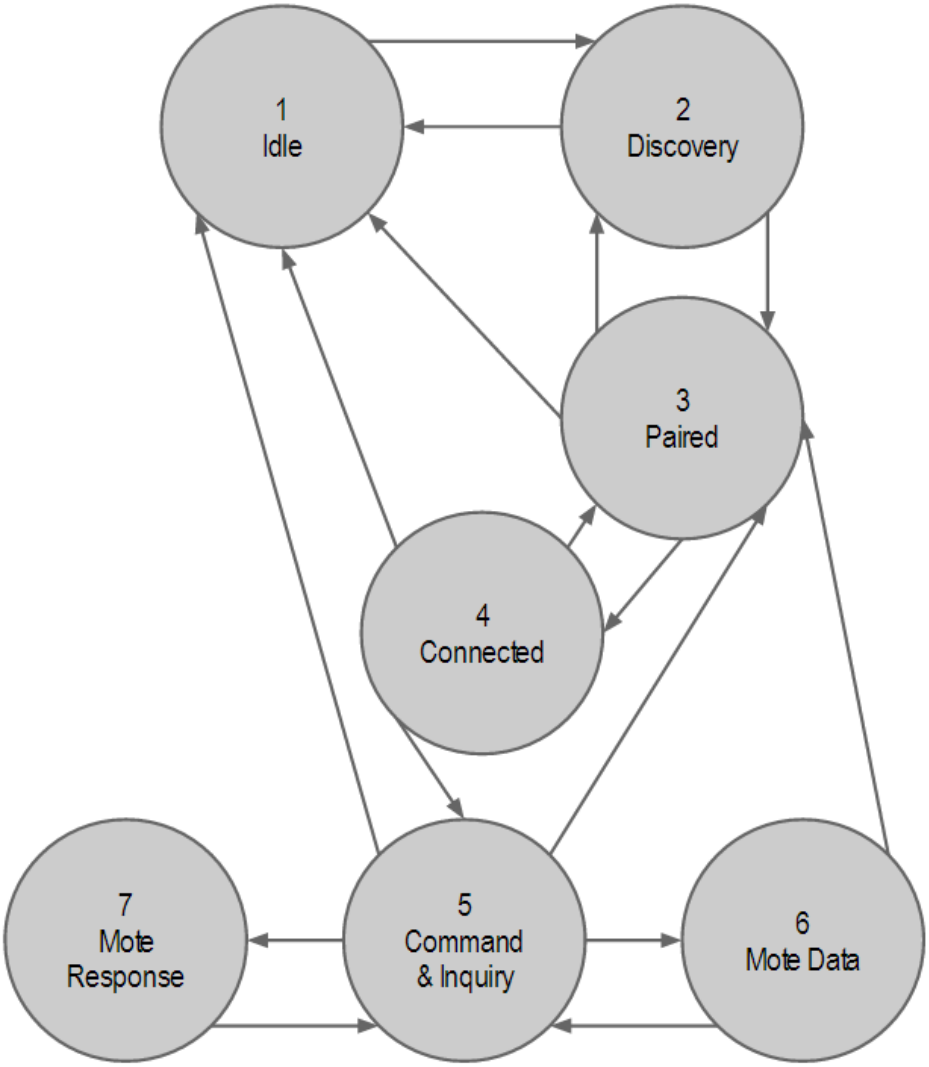


Figure 3: Base Station State Machine

4. Plug and Play Body Area Network

Before a practical BAN can be made, a platform flexible enough for such extensive research must be created. We decided to direct our focus on an extensible BAN and application layer protocol. This approach implemented a BAN flexible to additions of functionality. This BAN also provided practical benefits, such as the base station software not requiring updates with each change to the body area network.

4.1 Plug and Play Restrictions in Previous BANS

We aim to achieve a "plug and play" Body Area Network. In this context, plug and play means that the BAN should be flexible to the addition of new devices, motes, and sensors. The user should not have to update the base station every single time a slight change is made to the BAN configuration. Our research indicates that previous BANs do not include this goal in their scope; since their implementation is used for isolated research, and not for general commercial purposes. Our Application Layer Protocol addresses these issues: to begin transitioning the BAN from a research lab to a more commercial environment.

Body Area Network implementations such as Shimmer's BtStream ignore flexibility and extensibility as a design goal. Due to the lack of standardization with BAN protocols, and lack of focus for making BANs for everyday use, achieving these goals becomes quite challenging.

Currently, if a new sensor is added to the BtStream firmware, each preexisting device would need to be updated with the new information. If a mote is not updated, it could drop or possibly forward information it does not understand. Even worse, the base station would need to be updated if the user wants to be able to receive data from the new sensor at all.

The key design decision that causes these issues is the global context of information. In BtStream, every single device in the BAN is aware that a certain packet ID defines a particular command. Moreover, these packets also define what sensors currently exist in the BAN, and what can be done with them. If a new sensor or behavior is added, a new packet ID will have to be created, and dispersed to the devices in the BAN. This means each device will most likely have to be re-flashed or updated. These changes could be as simple as editing a configuration file, or as complex as rewriting a large portion of the firmware. However, this information is usually only pertinent to a single mote and its base station. If the mote was the keeper of its specific knowledge, and the base station was able to learn from the mote, the global context can be eliminated.

4.2 Application Layer Protocol for a Plug and Play BAN

Our application layer protocol attempts to resolve the issues just described, by removing the global context of information within the BAN. By making the motes the keepers of their own information they will as a result need a framework to communicate this information to the base station. The base

station will in turn need a generic way of understanding and parsing the information it learns from each connected mote.

4.2.1 Packet Structures

The following packet structures provide a foundation for motes to teach the base station about their sensors and commands, and to facilitate communication thereafter.

5-byte Overall Application Layer Header

Starting with 4-byte Packet Information

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Total Packet Size (in bytes)															
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Sequence Number															

Followed by 1-byte Message ID

0	1	2	3	4	5	6	7	MSG ID
0	0	0	0	0	0	0	0	Mote Data
0	0	0	0	0	0	0	1	Sensor Inquiry
0	0	0	0	0	0	1	0	Command Inquiry
0	0	0	0	0	0	1	1	Command Params Inquiry
0	0	0	0	0	1	0	0	Command Returns Inquiry
0	0	0	0	0	1	0	1	Data Inquiry
								Not Used
								...
								Not Used
1	1	1	1	1	1	1	1	BaseStation Learned Command

Figure 4: Application Layer Header

The application layer header, shown in Figure 4 above, consists of basic packet information, such as total packet size and packet sequence number, as well as a message ID. The message ID provides a means to identify and differentiate between packet types within the protocol. Compared to the BtStream protocol, there is a significantly smaller number of hard coded messages required. The seven currently defined messages types act as the framework for any mote to speak to the base station and teach it how to interact with a mote. The available sensors and behavior are not defined in global identifiers, but within the responses from a particular mote. If a new mote can implement and send the proper responses to these requests, then it should be able to join and operate in the BAN.

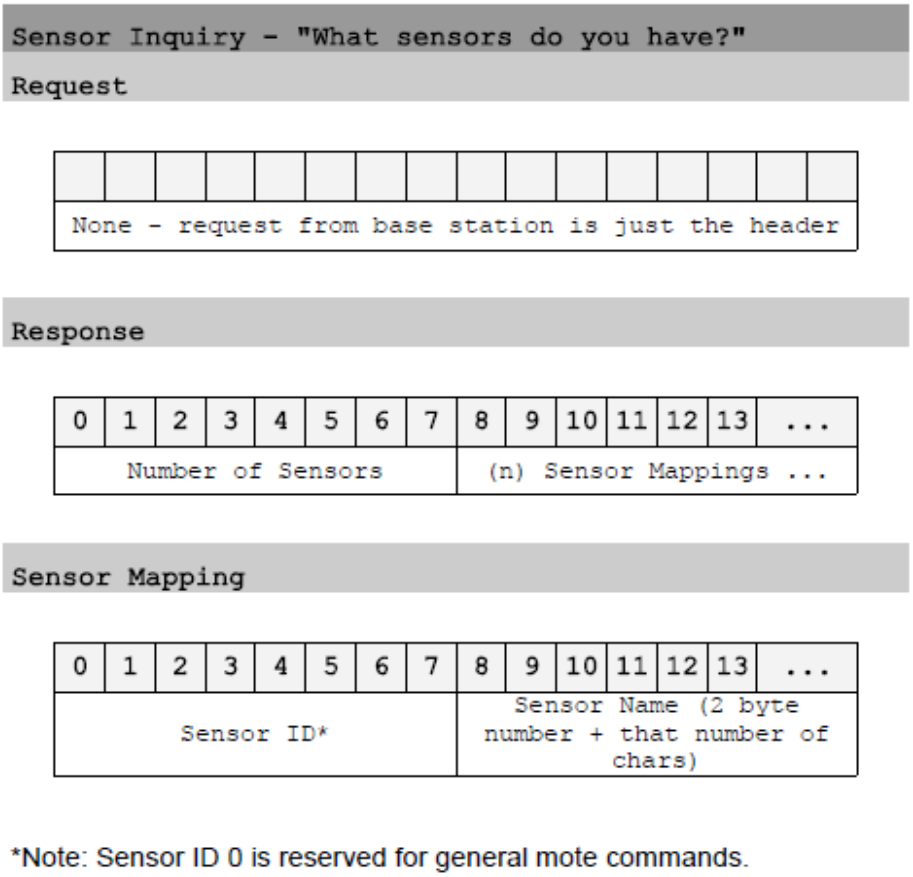


Figure 5: Sensor Inquiry Packet Structure

The first of the framework packets, the Sensor Inquiry show above in Figure 5, allows a base station to interview a mote to discover its available sensors. The mote must respond with a sensor mapping for each sensor it has available. This mapping consists of a sensor ID and the sensor's name. Note that this sensor ID is only applicable to this mote, and is not global to the BAN. The base station can

differentiate this sensor ID from other motes' sensor IDs by recording a unique group of sensor mappings for each mote. However, within the local context of a particular mote, sensor ID 0 is reserved to specify the general mote device and not a specific sensor on the mote. This restriction was made to provide a means with which to handle global actions on a mote, for example setting the overall sampling rate of the mote. The sampling rate cannot be set on a per-sensor basis and is therefore set by using the general mote sensor ID.

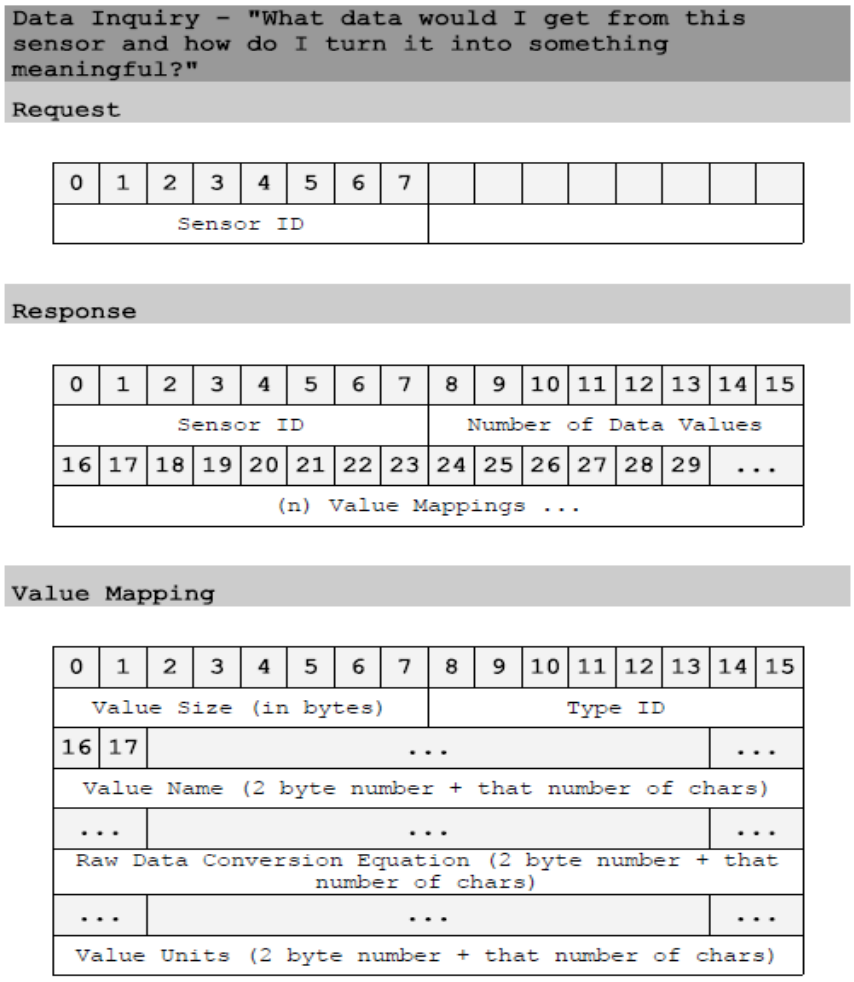


Figure 6: Data Inquiry Packet Structure

The Data Inquiry request packet provides the base station with a way to query a mote for information pertaining to the data of the mote's sensors. The response packet enables the mote to specify what type of data a sensor will produce and send to the base station. The structure of these packets can be seen above in Figure 6. Each value mapping consists of the size, type, name and units of the data sent. Additionally, a raw data conversion expression is included. Most motes are embedded systems, with

limited computational resources. Most BANs, including this one, prefer to send raw data to the base station for processing; the base station is usually a more powerful device that can handle luxury computations. In other BANs, such as BtStream, this conversion would be hard coded for a particular global data identifier. This conversion equation is needed in order to transform raw data into something meaningful for the BAN user.

For this equation, we have defined a preliminary grammar. The Raw Data Conversion Equation is an ASCII string representing a mathematical expression. Any instances of 'x' in the expression will be replaced by the raw data value. The other symbols should consist of operators and numerical constants. Basic arithmetic operators will be initially supported, but the supported operators could be expanded if necessary. The PEMDAS order of operations will also be assumed initially. As long as the motes and base station are in agreement, this grammar can be changed or expanded.

Command Inquiry - "What commands are available for this sensor?"

Request

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Sensor ID															

Response

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Sensor ID								Number of Commands							
16	17	18	19	20	21	22	23	24	25	26	27	28	29	...	
(n) Command Mappings ...															

Command Mapping

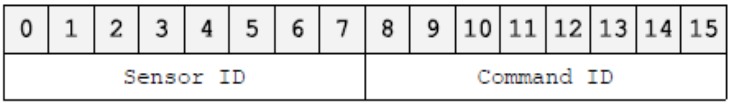
0	1	2	3	4	5	6	7	8	9	10	11	12	13	...
Command ID								Command Name (2 byte number + that number of chars)						

Figure 7: Command Inquiry Packet Structure

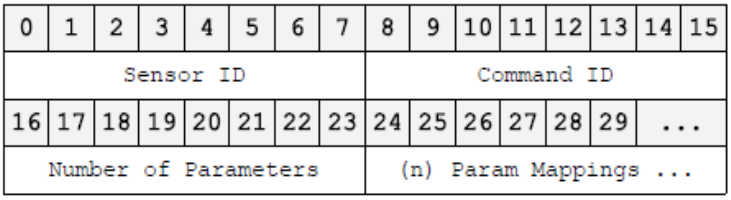
The base station learns about the commands available for the mote and each sensor through use of the Command Inquiry request packet, depicted above in Figure 7. The response by the mote provides the base station with the name of each command and an identifier for the command. Just as a sensor ID is associated with a particular mote, a command ID is associated with a particular sensor on a mote. As such, different sensors may have the same Command ID(s), but this is not a guarantee that those commands are functionally equivalent. A command ID is locally significant to a sensor on a mote. Due to the complexity of commands, getting the input and output are split up into separate packets.

Command Params Inquiry - "What are the specific parameters that need to be sent with this command and what do they mean?"

Request



Response



Param Mapping

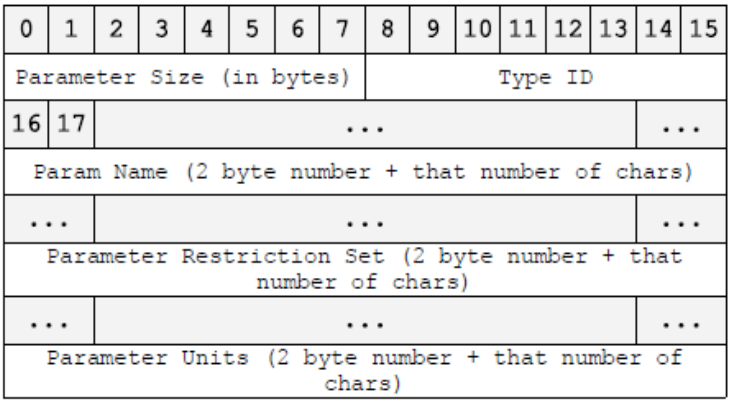


Figure 8: Command Parameters Inquiry Packet Structure

The Command Parameters Inquiry packet, shown in Figure 8 above, enables the base station to ask a mote for information regarding a particular command. The response by the mote specifies what parameters need to be sent with a particular command. This information is contained with parameter mappings. Included with each mapping is the parameter name, size, type, and units. In addition, a parameter restriction set is specified. Similarly to the raw data conversion equation, a grammar has been defined to restrict the subset of valid values a parameter can have.

Any values included must be numerical constants matching the parameter type specified. These constants can be defined as a range: 'i - j' conveys 'i to j inclusively'. They can also be defined as a set: 'i, j, k' conveys 'only exactly i, j, or k are valid'. Combining the range and set grammar was not permitted. In implementation, when values are defined as a set, these restriction set values are treated as enumerations by both the base station and mote firmware. For example, a sensor sensitivity restriction set of "5, 10, 15, 25" would be treated as an enumeration and mapped accordingly to values 0, 1, 2, 3. Therefore, when the base station wishes to change the sensor sensitivity to 10 it would send the value 1 as the command parameter in the command packet. Just like the previous grammar, if additional specifications are required, the language can be expanded as long as the mote and base station are in agreement.

Command Returns Inquiry - "What are the return values of this command? What does the stuff I'm getting back from the mote mean?"

Request

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Sensor ID								Command ID							

Response

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Sensor ID								Command ID							
16	17	18	19	20	21	22	23	24	25	26	27	28	29	...	
Number of Return Values								(n) Return Mappings ...							

Return Mapping

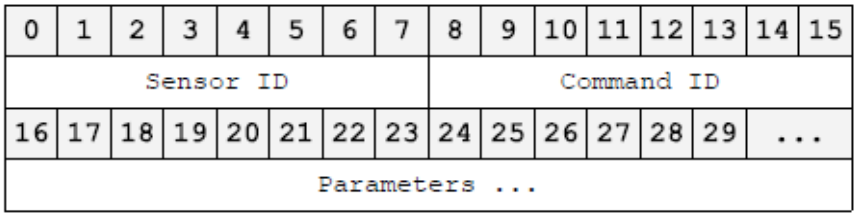
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Return Size (in bytes)								Type ID							
16	17	...												...	
Return Name (2 byte number + that number of chars)															
...		...												...	
Raw Return Conversion Equation (2 byte number + that number of chars)															
...		...												...	
Return Units (2 byte number + that number of chars)															

Figure 9: Command Returns Inquiry Packet Structure

The Command Returns Inquiry packet, displayed in Figure 9 above, specifies the return values the mote will respond with when a command is sent from the base station. The definition of this packet is similar to that of the data inquiry packet, as its mapping is exactly the same as the data inquiry packet. Since both packets are data responses from the mote, their mappings require the same information. However, we thought it would be best to separate streaming data and command returns from the mote.

BaseStation Learned Command - "Let me call the command I learned about from my inquiry packets"

Request



Response (Same as Mote Data Response)

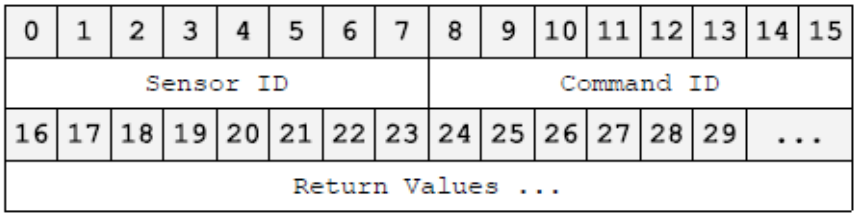


Figure 10: Base Station Learned Command Packet Structure

The Base Station Learned Command packet, shown in Figure 10, provides the base station with a means to execute a command it has learned from a mote. If a command executed has any associated return values those values will be included in the response from the mote. Otherwise, the bare response with just the sensor ID and command ID acts as an acknowledgement of the request.

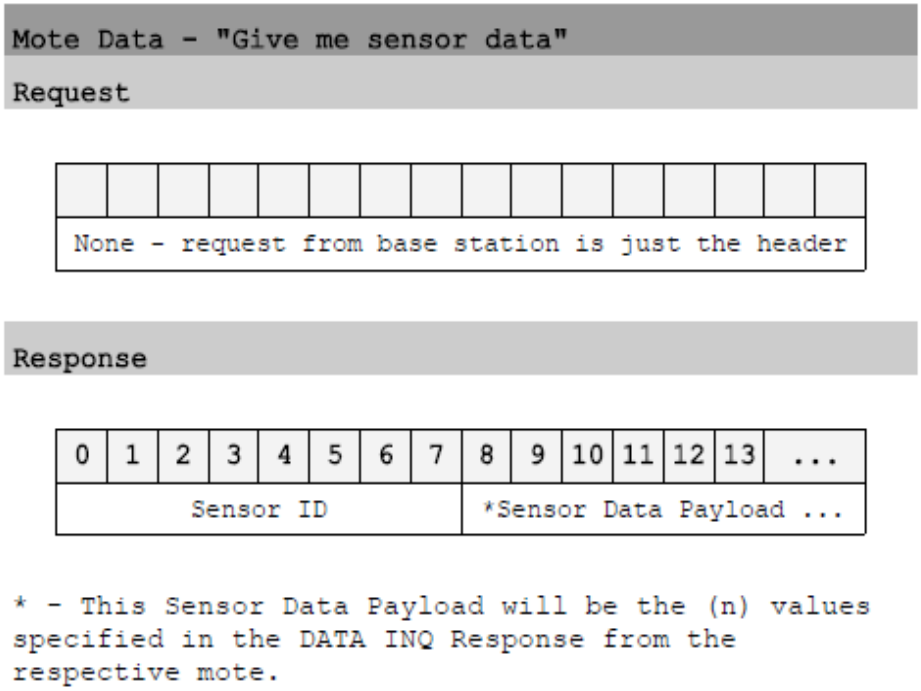


Figure 11: Mote Data Packet Structure

The Mote Data request packet is utilized to ask a mote within the BAN for sensor data. This packet, show in Figure 11 above, is sent to request data once the base station has connected to a mote and learned how to interact with it by following the previously described protocol. The request packet doubles as both a means to ask a mote to send and to stop sending sensor data. If the mote is not currently sending data, then the request will initiate streaming of data. If the mote is currently sending data, the request will halt further sending. The Mote Data response packet contains the sensor data for any sensors active and sampling on the mote at the time of the request. This data takes the form of the sensor ID followed by the sample data for that sensor. The sensor ID provides a means to differentiate between data from different sensors on the same mote and it used by the base station to separate and process data according to sensor type.

Data Types

1-byte Type ID

0	1	2	3	4	5	6	7	Type ID
0	0	0	0	0	0	0	0	Unsigned Integer
0	0	0	0	0	0	0	1	1s Complement Integer
0	0	0	0	0	0	1	0	2s Complement Integer
0	0	0	0	0	0	1	1	Signed Magnitude Integer
0	0	0	0	0	1	0	0	Half-Precision IEEE Floating Point (16-bit float)
0	0	0	0	0	1	0	1	Single-Precision IEEE Floating Point (32-bit float)
0	0	0	0	0	1	1	0	Double-Precision IEEE Floating Point (64-bit float)
0	0	0	0	0	1	1	1	Varchar (String)
0	0	0	0	1	0	0	0	Byte Array
								...
								Not Used

Varchar (String) Specification

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Varchar Size in bytes (ASCII chars)															
16	17	...													...
Null Terminated Character Array															

Figure 12: Protocol Data Types

Many of the protocol packets described include a type ID. Due to the fact that bytes sent back and forth over the network can be interpreted in several different ways, type IDs were needed to standardize their interpretation per packet. Figure 12 above defines a preliminary set of ways to interpret a byte sequence. More can be defined as needed. However, for general use, the byte array type should be flexible enough for most interpretations not defined above.

The protocol outlined accomplishes our aforementioned design goals. This application layer protocol does not inherently rely on static message identifiers for the motes and base station to

communicate meaningfully. The only specified message identifiers define the learning process for the base station application. Since this protocol supports virtually every data type, input restriction, and output conversion, the identifiers are unlikely to change. It is possible that the restriction set and conversion equation grammars would need to be expanded. This achieves our next design goal: a base station learning language that can be expanded easily through changes to a few grammars. Most of the unprecedented learning situations will occur when commands and sensor data are described. Each of these is dependent on a respective grammar. Since these grammars are merely a predetermined language, they can be expanded as long as both parties are in agreement. These grammars are already fairly general, and will probably not have to be updated for most applications.

The most common change to the BAN will be the addition of new sensors, motes, or commands. The current protocol version supports these without changes to the current motes' firmware or base station application: achieving yet another goal. As long as a new mote can talk to the base station in the predetermined language, it can teach the application everything it needs to know. New sensors and commands need only be appended to the existing inquiry responses. Finally, this protocol is flexible for any type research or real world application because it supports virtually any nuances. The grammars and inquiry structure we defined allow for an unlimited set of capabilities to be incorporated into any BAN that implements our protocol.

4.3 Firmware Implementation

Firmware for the sensor motes was implemented in a variant of the C programming language called nesC [30], the language that the TinyOS operating system is written in. We chose to utilize TinyOS and leverage nesC for our firmware implementation as those technologies were utilized by Shimmer Research for their applications. In addition, the Python programming language was utilized to make testing scripts to simulate interaction with a base station device.

Implementation was carried out on modified versions of the virtual machine development environment provided by Shimmer Research with the Shimmer platform development kit. These virtual machines came pre-loaded with the TinyOS source code and many of the required libraries necessary for TinyOS development and helped streamline the development set-up process. We utilized the Eclipse IDE [31] and YETI 2 Eclipse Plug-In [32] during development. The YETI 2 plug-in provides an extension to the Eclipse IDE and enables nesC editor support and built-in tools for TinyOS development. A Git repository hosted on WPI's FusionForge [33] provided version control for the development.

4.3.1 Mote Firmware Architecture

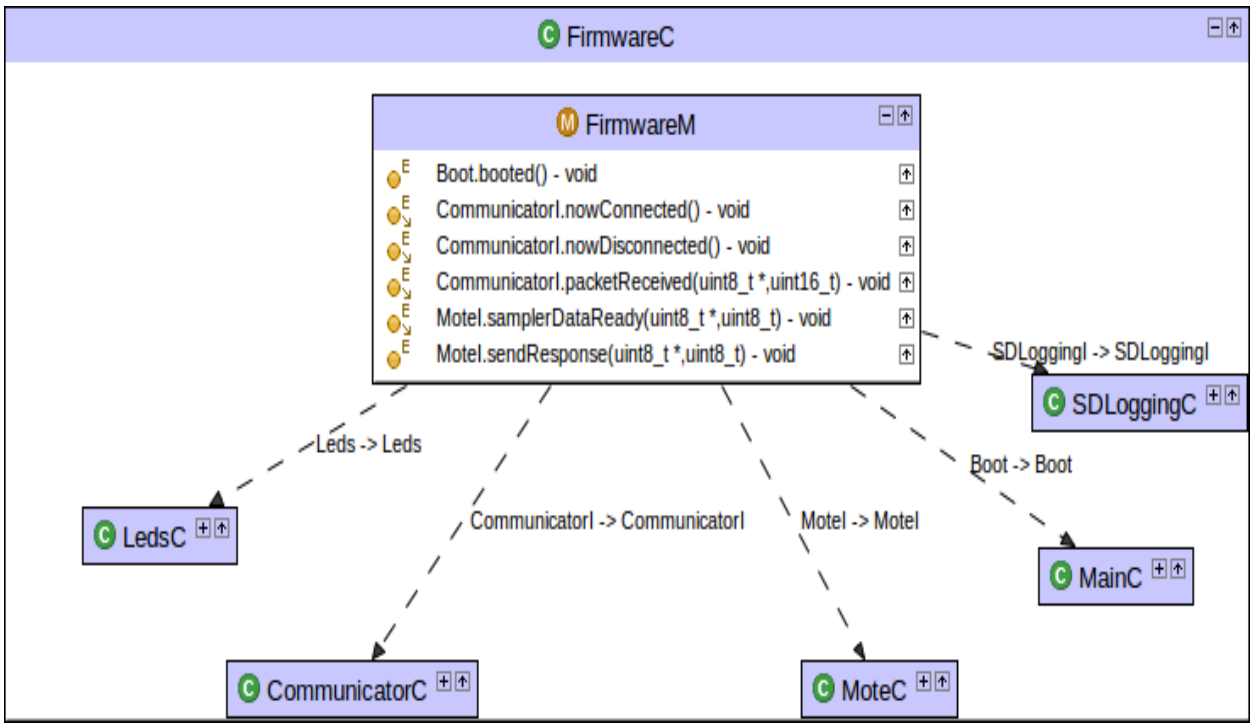


Figure 13: Firmware Component Diagram

The main firmware component, shown above in Figure 13, is split up into two major components. The Communicator deals with sending and receiving data while the Mote handles processing data. The firmware module itself is quite lightweight, as it only deals with delegating tasks or managing interactions between the Communicator and Mote components. The SD card logging module is initialized within the firmware at boot time and then can be utilized elsewhere by other components and modules.

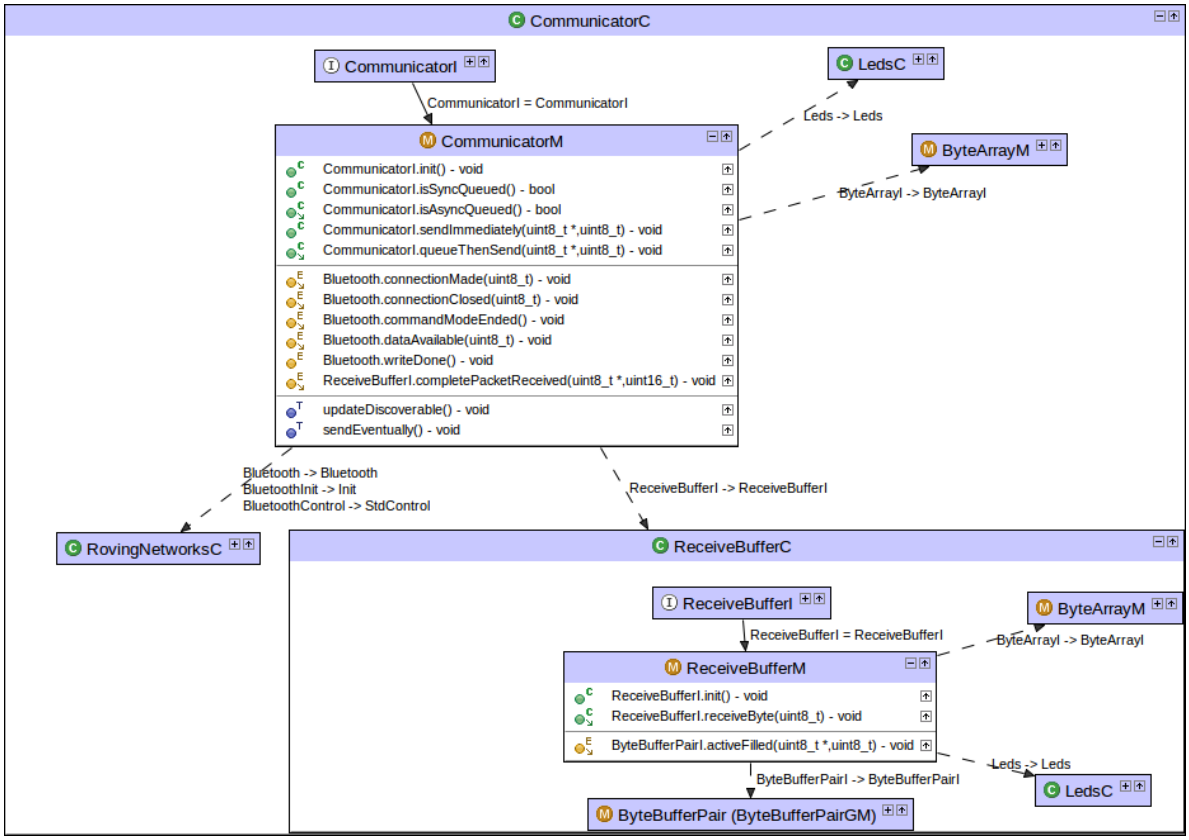


Figure 14: Communicator Component Diagram

The Communicator component, shown in Figure 14, handles the sending and receiving of data on a mote. Currently, motes in the BAN utilize the Bluetooth protocol and as a result the Communicator implements Bluetooth related functionality such as connection events (e.g. when a connection is received and when a connection is closed). Bluetooth could be replaced with another communication protocol if desired and with limited changes needed. When the Communicator receives data it stores the data in a double-buffer data structure called the ReceiveBuffer. Once an entire packet is received and stored by the ReceiveBuffer, the Communicator then hands the packet off to the Mote component for parsing. All packets leaving a mote are passed to the Communicator component for transmission.

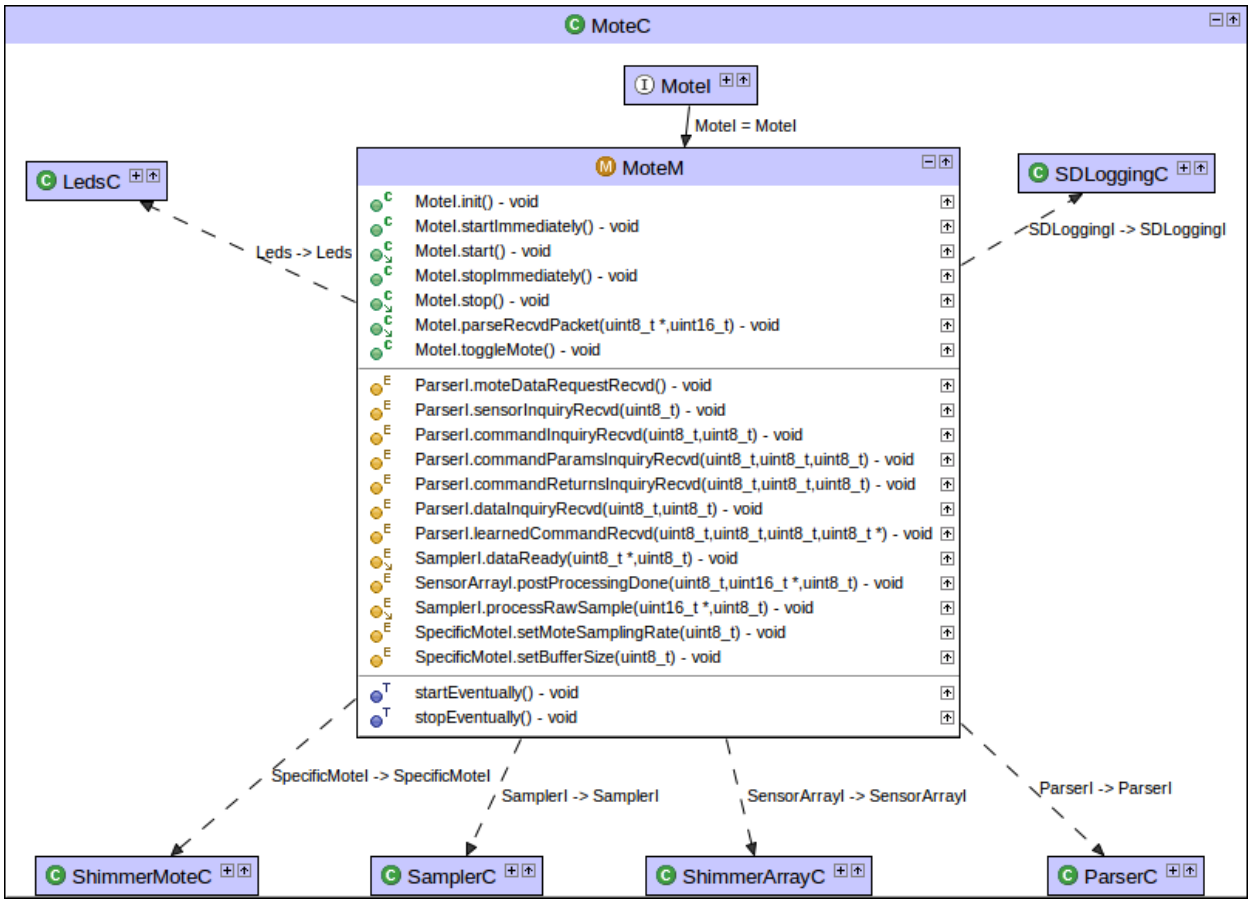


Figure 15: Mote Component Diagram

The Mote component, shown in Figure 15, handles incoming request processing, delegating requests to sensors, forwarding responses back to the Firmware component for transmission, as well as handling sensor data sampling. These three main behaviors are split into the parser, sensor array, and sampler components respectively. In addition, the Mote component handles requests made not to a specific sensor but to a mote in general, e.g. setting the sampling rate of the mote which is global to the entire mote.

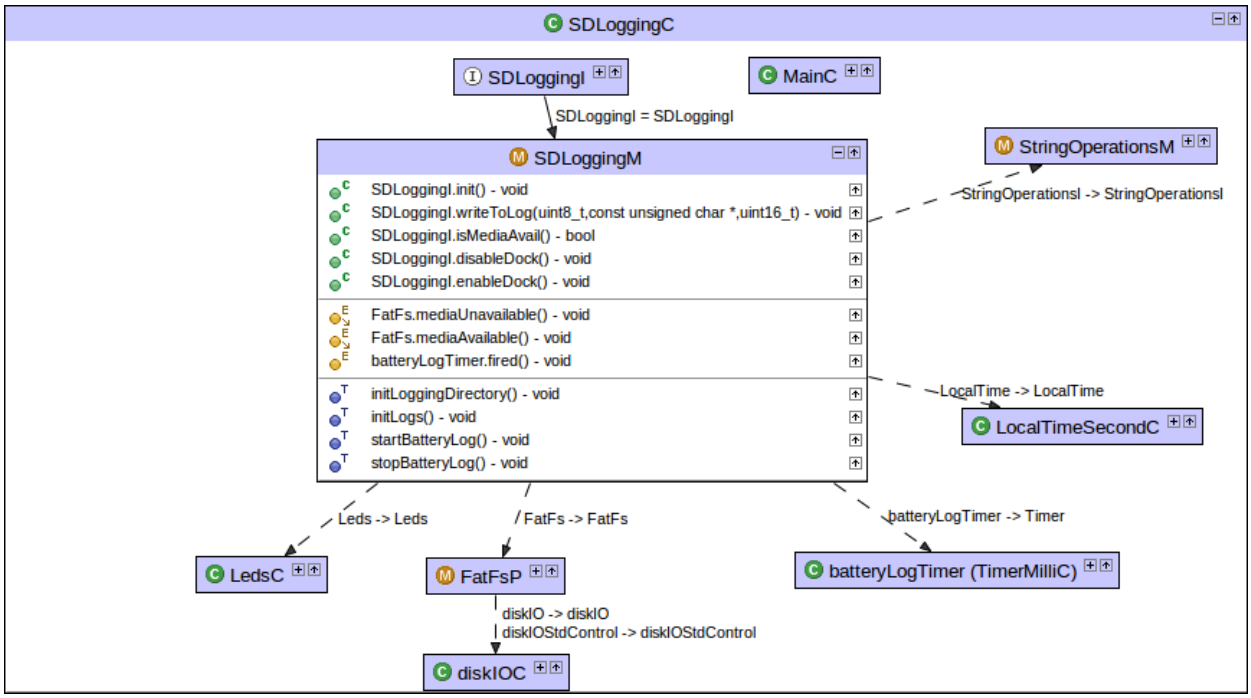


Figure 16: SD Card Logging Component Diagram

The SDLogging component, shown in Figure 16, provides basic logging capabilities to the firmware. Currently, the firmware creates a "/logs" directory in the root directory of the SD card on the mote. Four log files are created within the logs directory at firmware boot time: an information log, a warning log, an error log, and a battery log. This module can be expanded or modified easily to facilitate firmware logging needs.

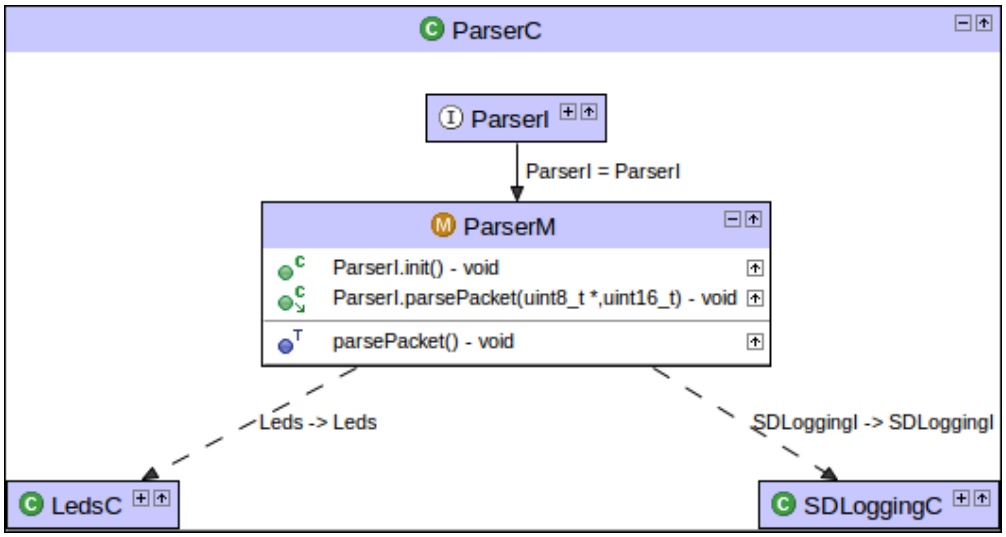


Figure 17: Parser Component Graph

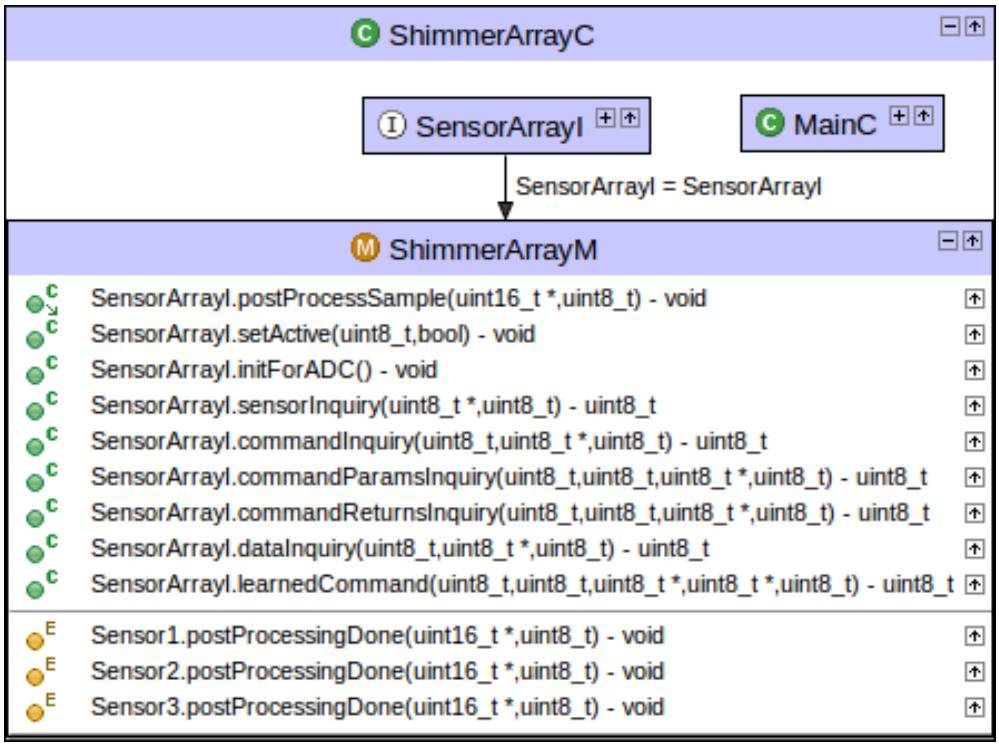


Figure 18: Sensor Array Component Diagram

The Parser and ShimmerArray components, shown in Figure 17 and Figure 18 respectively, both work underneath the mote component. Received packets are sent to the Parser component and initial parsing is done to figure out what type of packet was received. Based on the type of packet parsed,

actions are delegated to other components by first informing the Mote component that a specific packet type has been parsed. Any information in the packet that is required by a component down the line is handed back to the Mote from the Parser. The Mote component then calls the appropriate commands, defined in the sensor array component, in order to process the request parsed by the parser. The sensor array abstracts functionality common to all sensors on a mote. Requests handled by the sensor array are returned to the Mote component once completed. For example, if a sensor inquiry request is sent to a mote the sensor array will construct the response packet and then hand it off to the Mote component for handling.

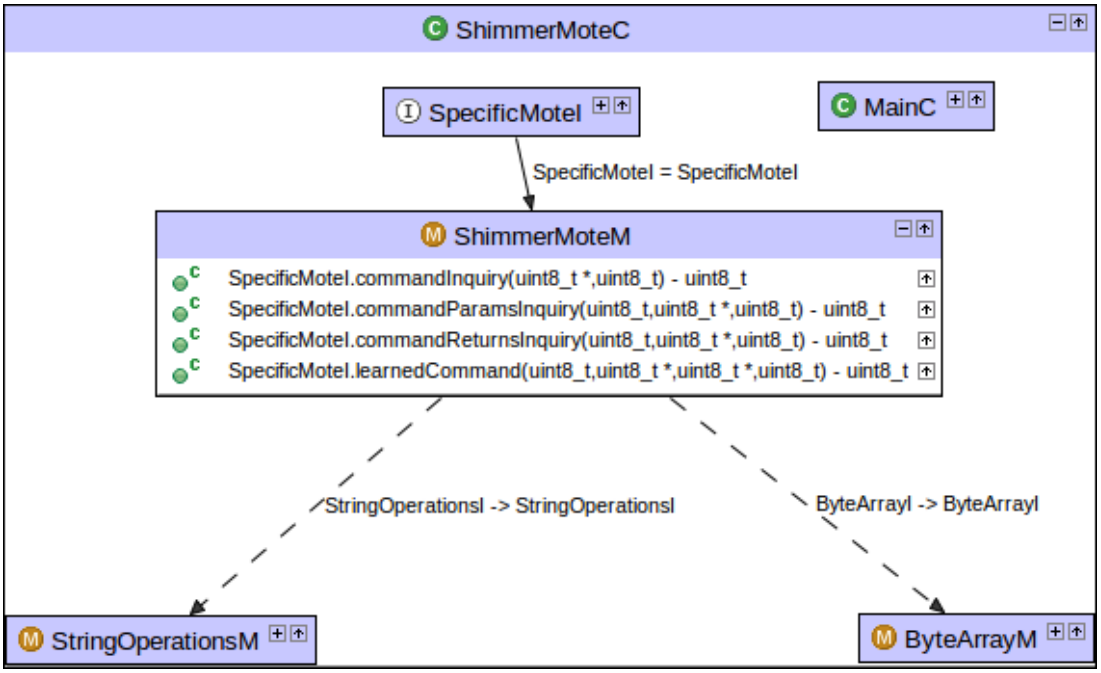


Figure 19: Shimmer Mote Component Diagram

The ShimmerMote component, shown in Figure 19 above, is utilized to handle any global requests made to the mote that do not pertain to a specific sensor. Global requests such as setting the sampling rate or buffer sizes of the mote are examples of requests this component handles.

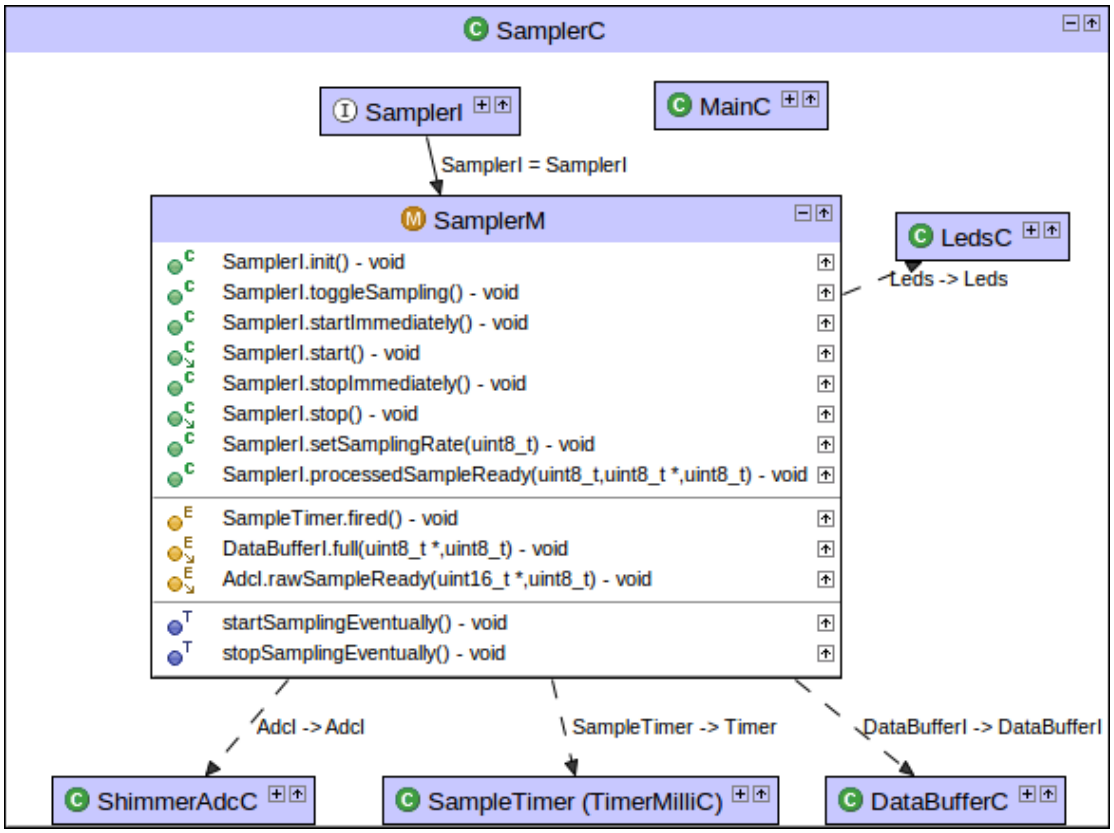


Figure 20: Sampler Component Diagram

The Sampler component, shown in Figure 20 above, handles aspects related to sensor data sampling and sample aggregation. When a non-zero sampling rate is set and when instructed to start, the module will tell the ADC component to begin sample at the proper rate. Any sensors that have been initialized for ADC sampling will be included in the ADC samples. Once the ADC finishes taking a raw sample, the Sampler will ask the mote to process the sample. The processed samples will then be stored in by a buffer in the DataBuffer component. When this buffer is filled with the configured number of samples, it will signal the Sampler that buffer is full. The Sampler will then delegate to the Mote to handle the samples that are ready to be sent out.

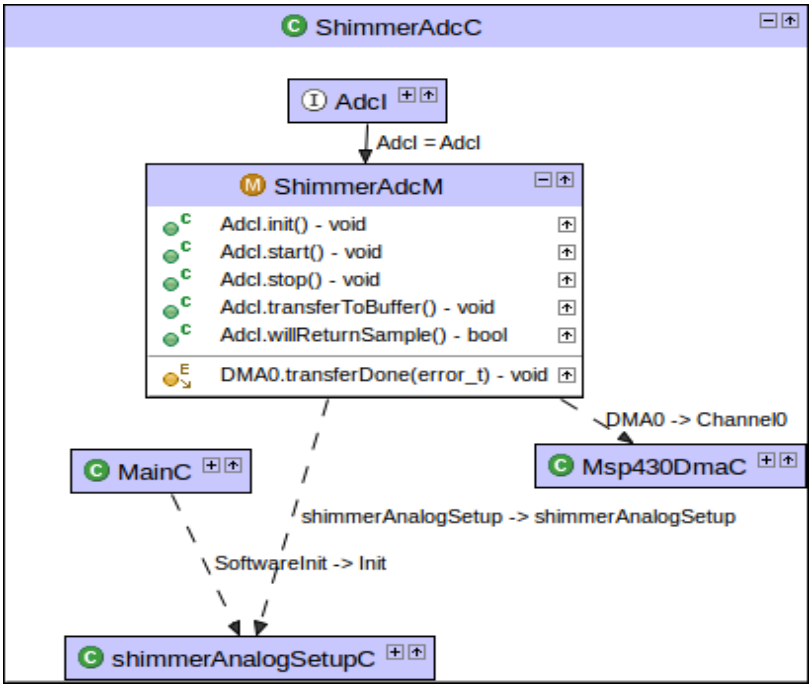


Figure 21: Shimmer ADC Component Diagram

The ShimmerADC component, show above in Figure 21, provides functionality to interact with the MSP430's ADC hardware in order to facilitate data sampling from sensors on the motes. Currently, as our firmware only supports Shimmer sensors and hardware, the ADC exclusively utilizes the shimmerAnalogSetup component. That component enables Shimmer sensors to be added to the ADC inputs in preparation for sampling and is used by all Shimmer sensor modules.

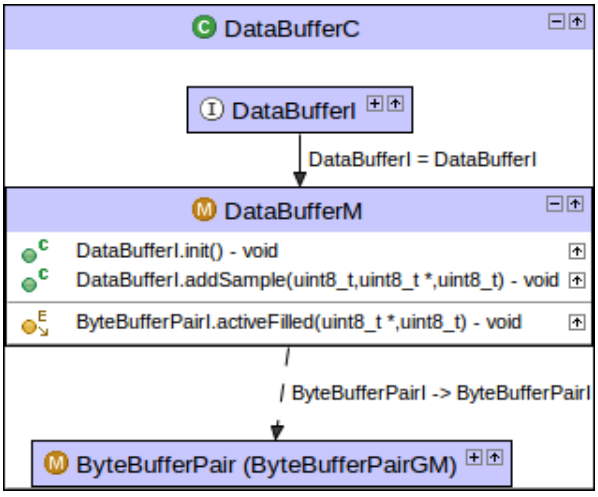


Figure 22: Data Buffer Component Diagram

The DataBuffer component, shown in Figure 22 above, is comprised of a double-buffer structure and is utilized to store sensor samples received by the ADC. The Sampler component utilizes this component to store the ADC samples. Once the data buffer is full, the buffer is handed off and any data post processing required by specific sensors is carried out by individual sensor modules.

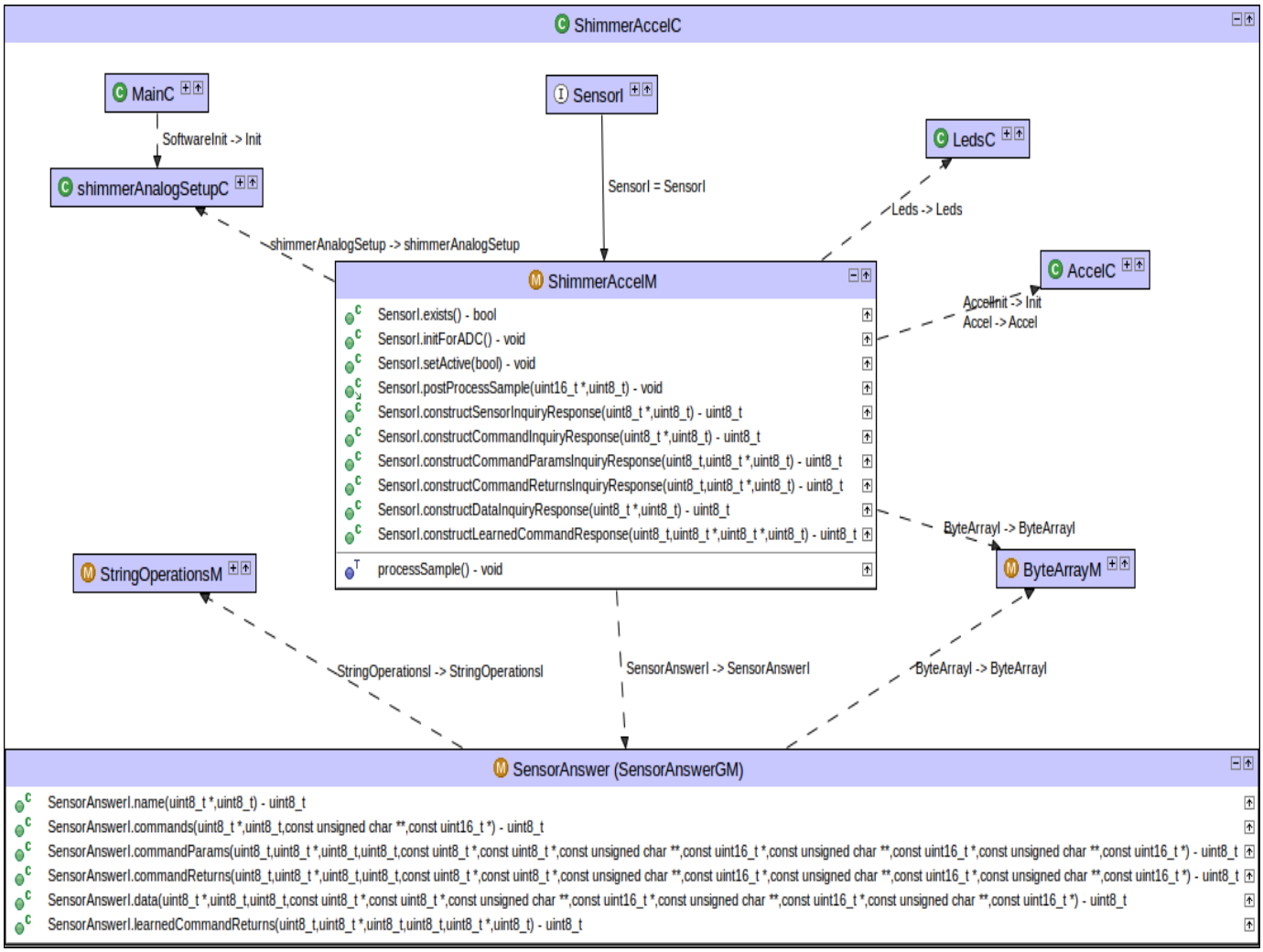


Figure 23: Shimmer Example Sensor Module Component Diagram

The ShimmerAccel component, see Figure 23 above, is an example of one of the sensor modules created to provide functionality specific to a sensor on a mote. Each sensor module provides the same interface and general functionality. A sensor module handles constructing protocol responses to requests made to the mote, as well as carries out sensor data post processing if necessary.

4.4 Base Station Implementation

The base station application was created in the Android SDK [34], a freely available modification of the Eclipse IDE [31]. Source control was provided by a Git repository hosted by WPI's FusionForge [33].

4.4.1 Application Architecture

The base station app consists of five major components.

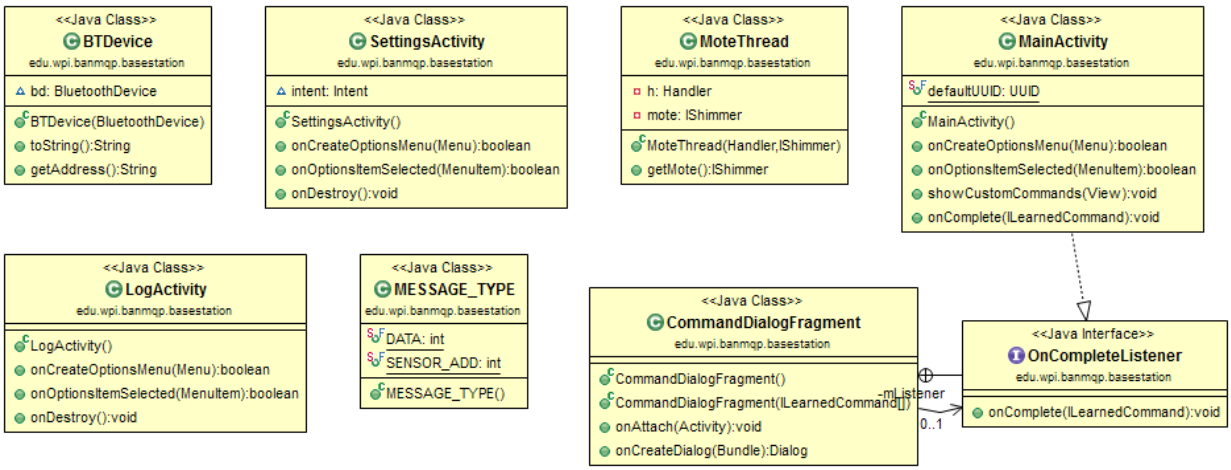


Figure 24: UML Diagram of the Application Frontend Component

The Application Frontend is responsible for the Graphical User Interface (GUI) of the app and can be seen in Figure 24 above. This is the first of two components that will be heavily dependent on implementation platform. There are three main Activities, or Views, in the application. These are the MainActivity, which displays connected sensors and any data being received from them, the SettingsActivity, which shows a list of possible sensors to pair with, and the LogActivity, which is used to display internal application logging. Finally, the CommandDialogFragment is used to show a pop-up list of commands for a sensor selected in the MainActivity. Each MoteThread is associated with one sensing platform, and is responsible for communicating with that platform. When needed, the MoteThread can pass messages to the MainActivity to trigger updates to the GUI, such as when new data is received or a new sensor is connected. These messages are assigned a MESSAGE_TYPE. Finally, the

BTDevice was created as a wrapper class for the built in android BluetoothDevice to provide specialized functionality in some methods.

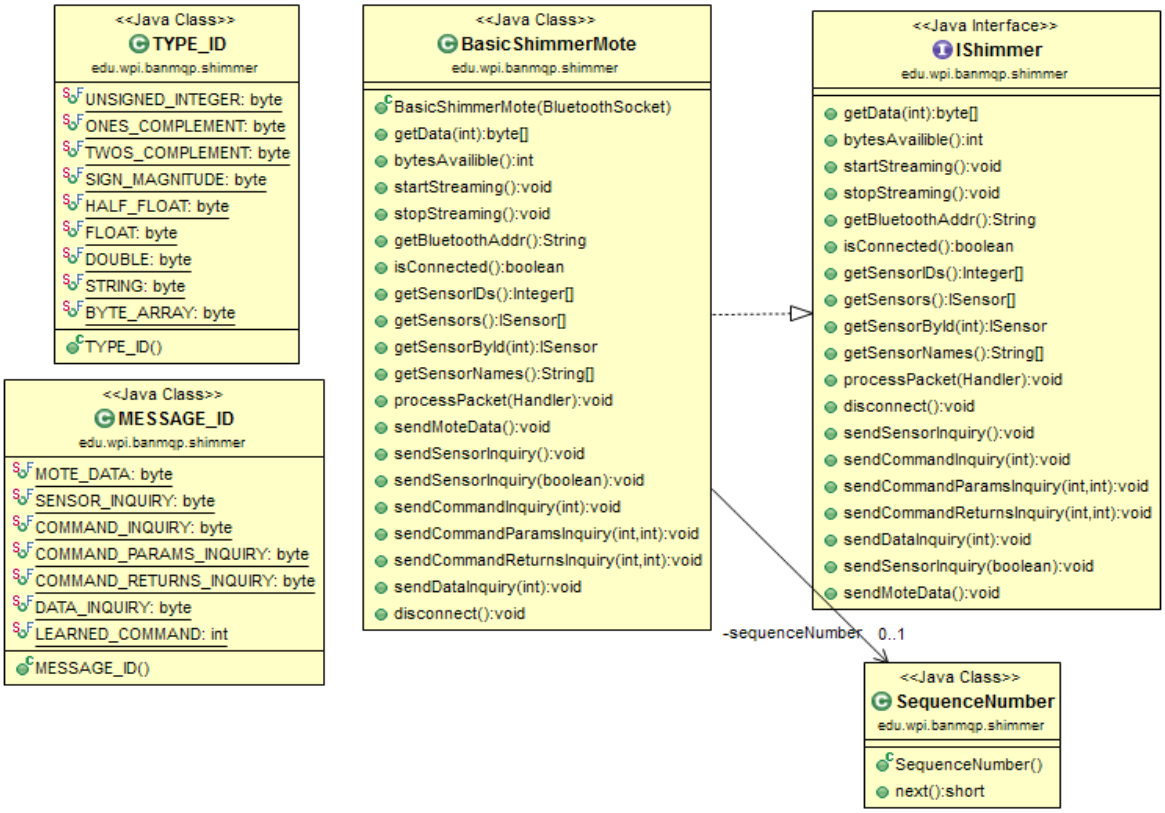


Figure 25: UML Diagram of the Mote Component

Each sensing platform, or mote, is represented by a BasicShimmerMote (shown in Figure 25), which implements the more general IShimmer. This interface could probably be implemented for motes created by other companies as well, however each implementation will probably be platform specific. Each BasicShimmerMote is constructed by passing the Android BluetoothSocket object that represents the connection to the mote. Each mote contains all the logic needed to send, receive, and process packets. A new static SequenceNumber is instantiated by each BasicShimmerMote to make sure that the sequence number applied to outgoing packets is incremented properly. The SequenceNumber.next() method is thread-safe. Finally, the TYPE_ID and MESSAGE_ID classes provide definitions for constants defined in the communication protocol.

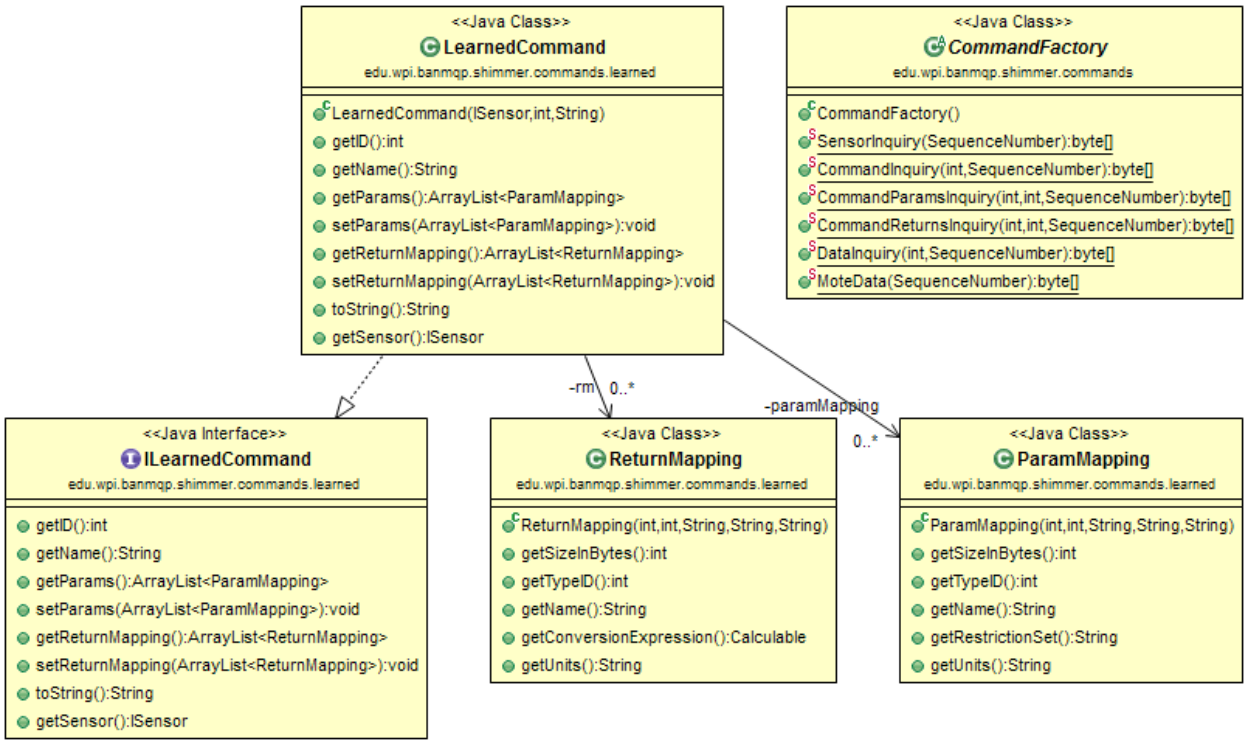


Figure 26: UML Diagram of the Command Component

The Command Component, shown in Figure 26 above, is responsible for creating and processing any arbitrary command that the base station learns from the communication protocol. The CommandFactory provides static methods for generating the commands that are already built into the protocol. LearnedCommand, an implementation of ILearnedCommand, represents a single learned command. Each learned command may have any number of associated ReturnMapping and ParamMapping, which allow the base station to process return values and assign parameter values respectively.

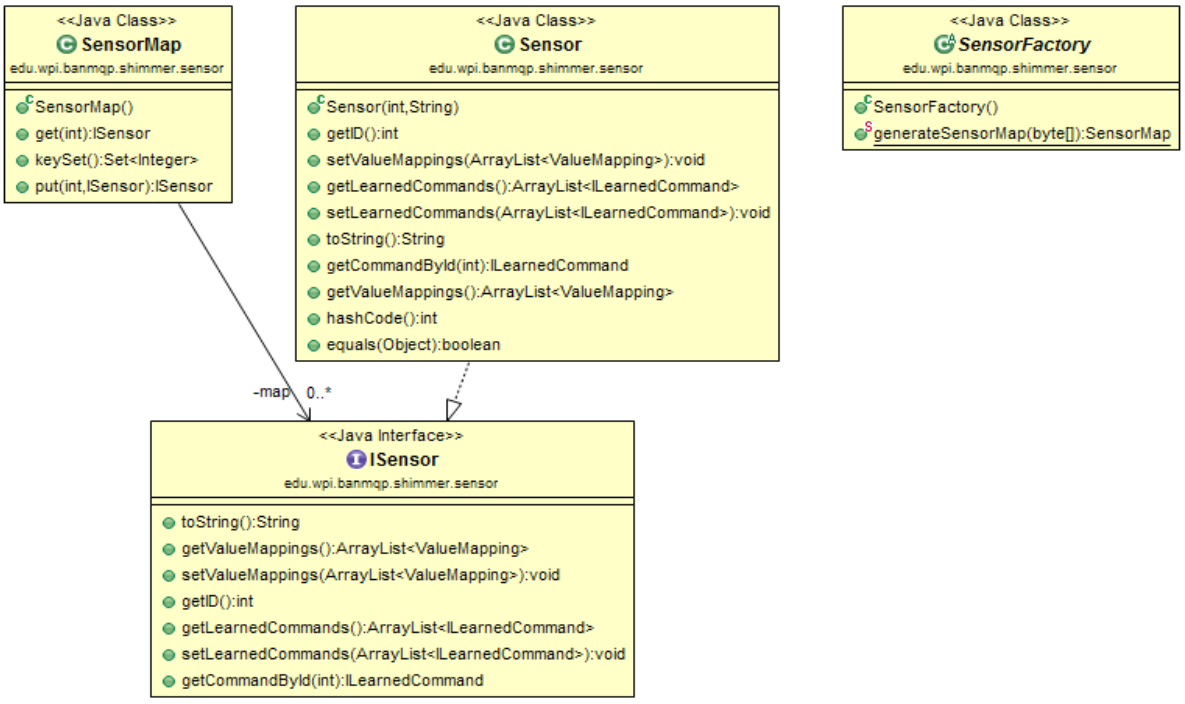


Figure 27: UML Diagram of the Sensor Component

The Sensor Component deals with each individual sensor, as opposed to the entire mote and is depicted in Figure 27 above. A SensorMap is a wrapper for the Java HashMap structure, however any underlying Map structure that provides a key set could be used. The ISensor interface provides the needed commands for each individual sensor, and is implemented by the Sensor class. Finally, the SensorFactory was made to separate the logic for generating Sensor objects from protocol data. This logic could also be handled in the IShimmer processPacket method.

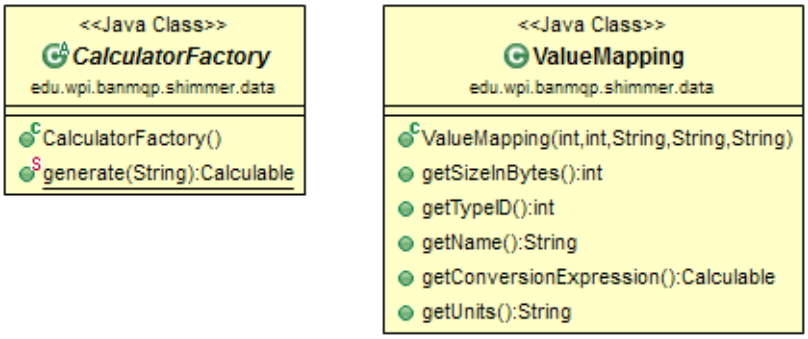


Figure 28: UML Diagram of the Data Component

The Data component, shown in Figure 28 above, is fairly simple. It consists of a ValueMapping, which contains the necessary information to parse and format mote data, and a CalculatorFactory, which creates the required Calculable objects to convert mote data into a more readable form. Please see Section 4.4.1.2 for more information about Calculable objects

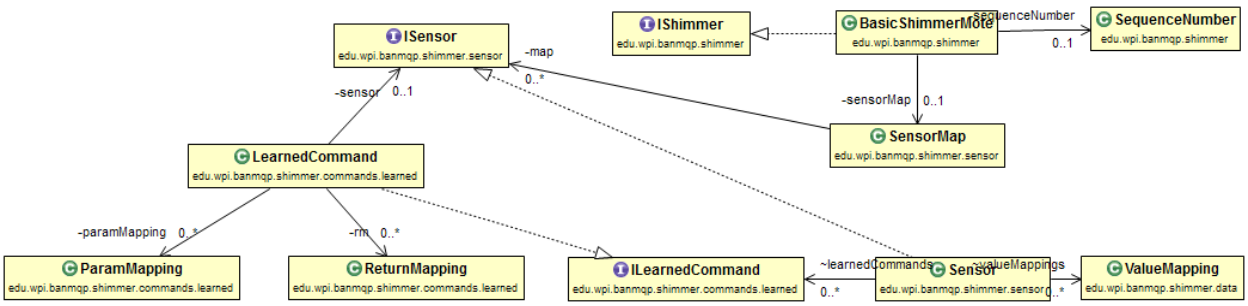


Figure 29: UML Diagram of Component Interconnection

Figure 29 (above) provides a high level view for how each component is connected.

4.4.2 Technologies Utilized

As a part of the base station implementation, we utilized some third party libraries to expedite development. The following are descriptions of the libraries used, and how they were integrated into the application.

4.4.1.1 Expj4

As part of the protocol, the base station is required to convert mote data. The Conversion Expression provided as part of the Data Inquiry had to be parsed into a mathematical expression capable of being evaluated multiple times for different variable values. The exp4j [35] library is a small implementation of Dijkstra's Shunting Yard Algorithm, which in this case is used to create Calculable objects based on Strings of mathematical expressions. By defining a character to represent the raw data value, it is possible to use this library to easily and quickly create a Calculable object for any valid

mathematical expression. This object can then be reused each time a new data value is received from a mote and transform that value into an appropriate unit and scale. This implementation uses the character ‘x’ to denote the location where the raw mote data should be substituted in the expression.

4.4.3 User Interface Design

In order to convey the plug and play abilities of our protocol, we needed a user interface to clearly display the functioning of our design goals. While user experience was not a priority, we still required an interface to make our protocol usable. The following details the choices user interface decisions we made.

4.4.3.1 Graphing

Although displaying information in real time can be helpful, displaying data in a pleasing and helpful manner can help a user better understand what it means without the need for more in-depth analysis. This is why graphing was included in the base station software. We determined a set of features that our method of plotting would need to offer: it would need to be real time, be well used, and have a license that would allow it to be included in the base station software.

The most obvious option was to simply roll our own implementation of graphing. However, upon seeing the selection of available libraries it became clear that there were already options that perfectly fit the application requirements.

One of the first libraries investigated was AChartEngine [36]. ACE has a large number of plotting methods, which is nice from a usability point of view. It claims to be well optimized for dealing with a large number of values at a time. It also has a decent following on google code and Facebook. However, Real time plotting doesn’t seem to be offered, and the project has not been updated in seven months at the time of writing.

ChartDroid [37] offers most of the features that the base station requires. However, by design it is impossible to include it within our base station. Rather than importable libraries, ChartDroid requires the user to install external libraries and edit configuration files, with very little benefit in return.

GraphView [38] is a simple and efficient implementation of basic graphing features. It was not used in the base station because it can only display bar charts. More importantly it requires all values

beforehand, making it thoroughly useless for real time plotting. Java Charts for Android (Java[TM] Charts for Android) was included in the list of libraries to consider, but was quickly ruled out because the license does not permit free educational use.

AndroidPlot [39] was determined to be the best library to use for this project. It uses the Apache 2.0 license, so it can be included in the base station. It offers a decent selection of unique graph types, so there is a good chance that it will be able to handle whatever data is generated by the BAN. It is in active development, and is used in hundreds of applications, so it has been well tested.

We decided to use the best option available, and integrated AndroidPlot. The combination of useful features, active development, robust community and compatible license made it a perfect fit. The graphs were added as a scroll view, so that users can quickly view the incoming data from every sensor. With this implementation, users are provided the easy to interpret overview that graphs provide.

4.4.4 BAN Base Station Application

Below are several screenshots demonstrating the operation of our base station application. Note that on the home screen, each sensor will be grouped by background color. Each mote may have multiple sensors, as seen in Figure 33.

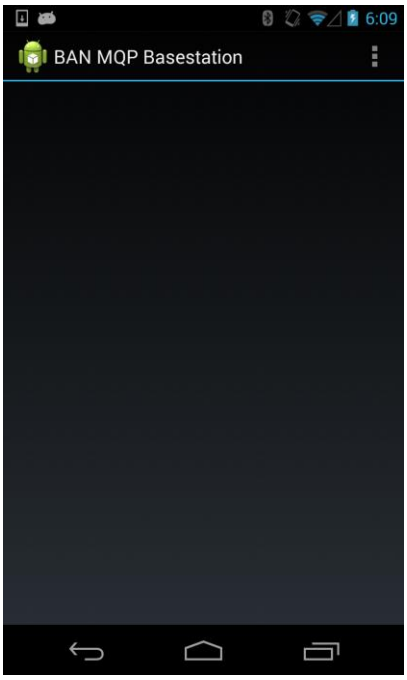


Figure 30: Base Station Application Home Screen

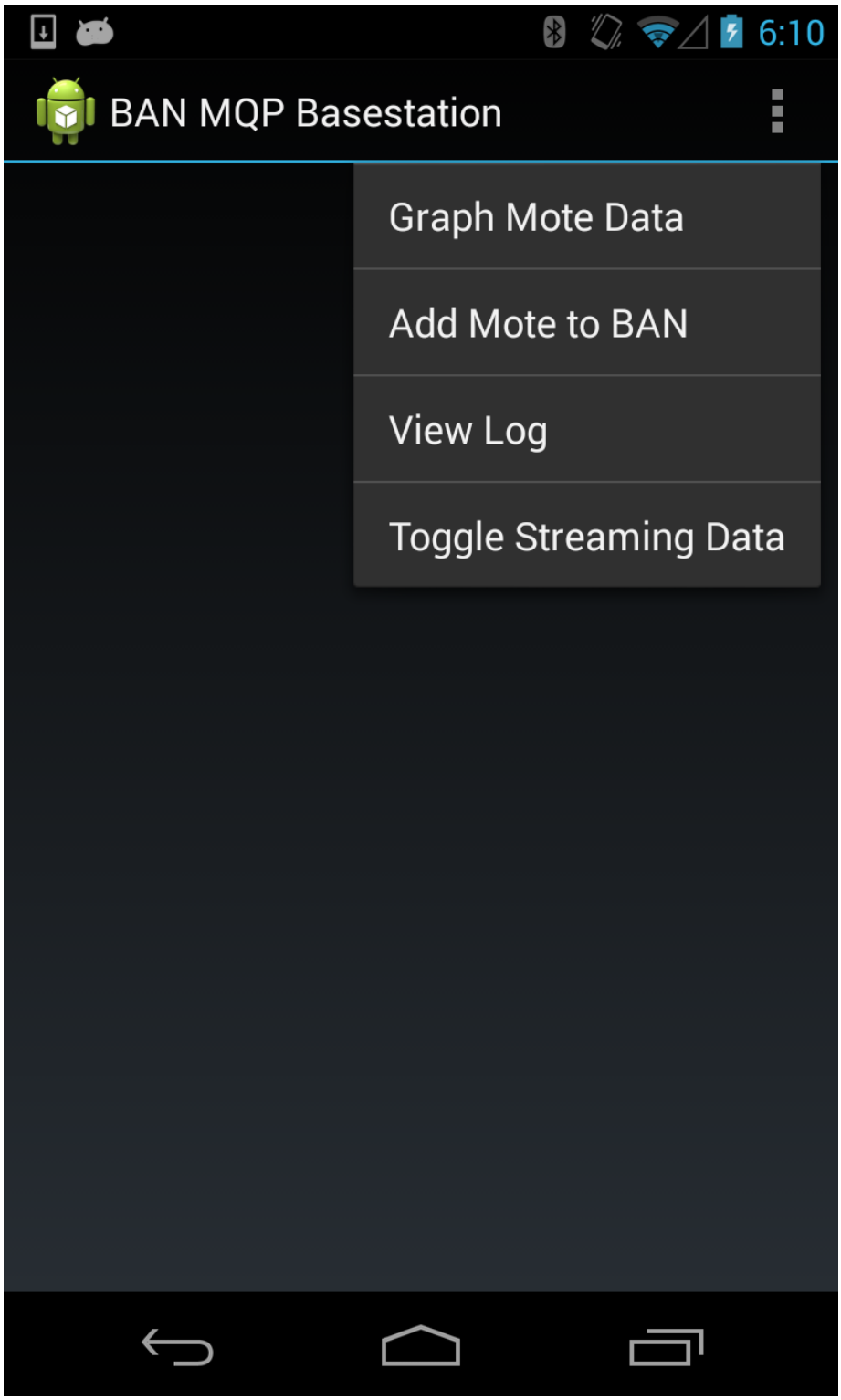


Figure 31: Base Station Application Option Menu

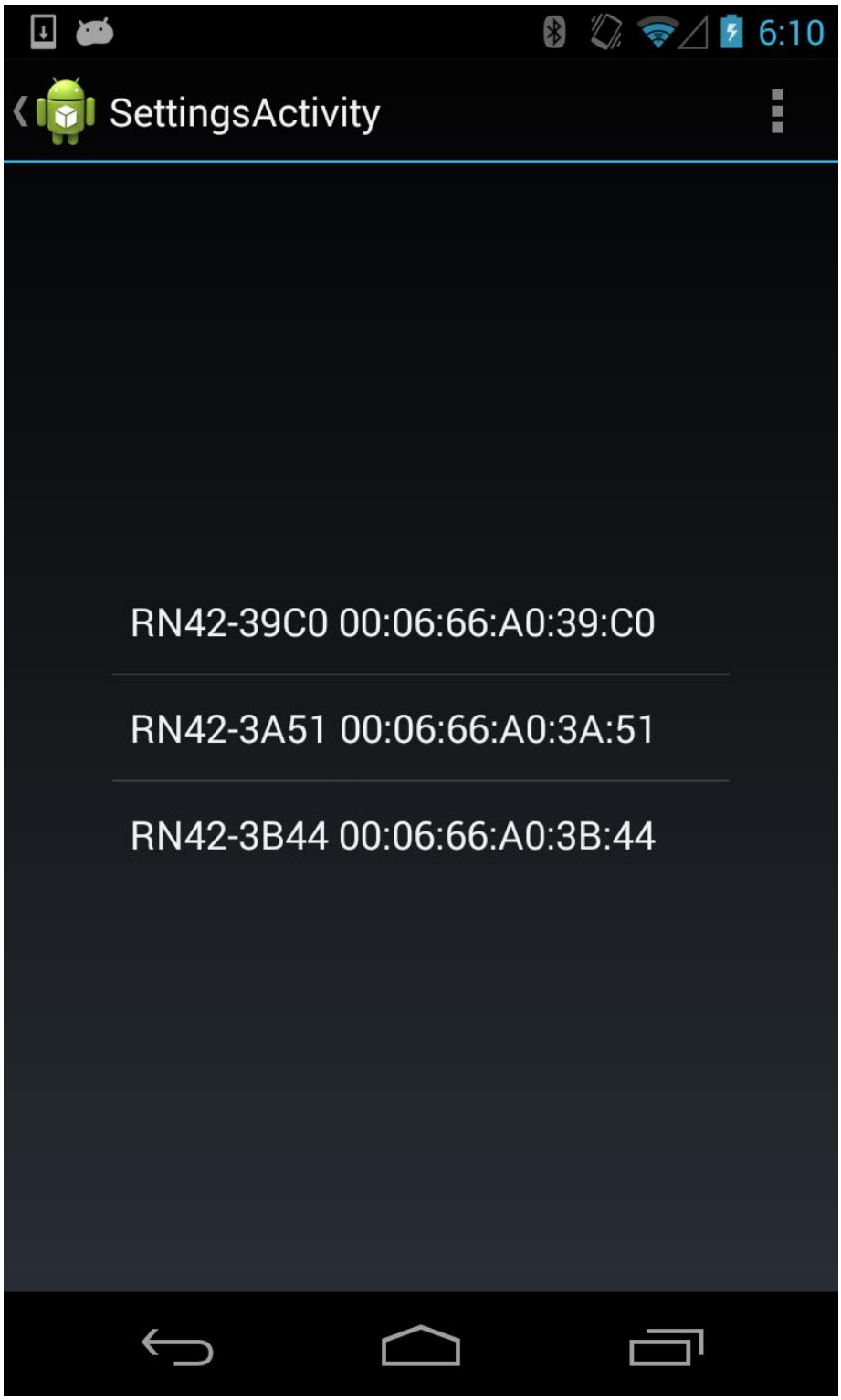


Figure 32: Base Station Application Add Mote to BAN

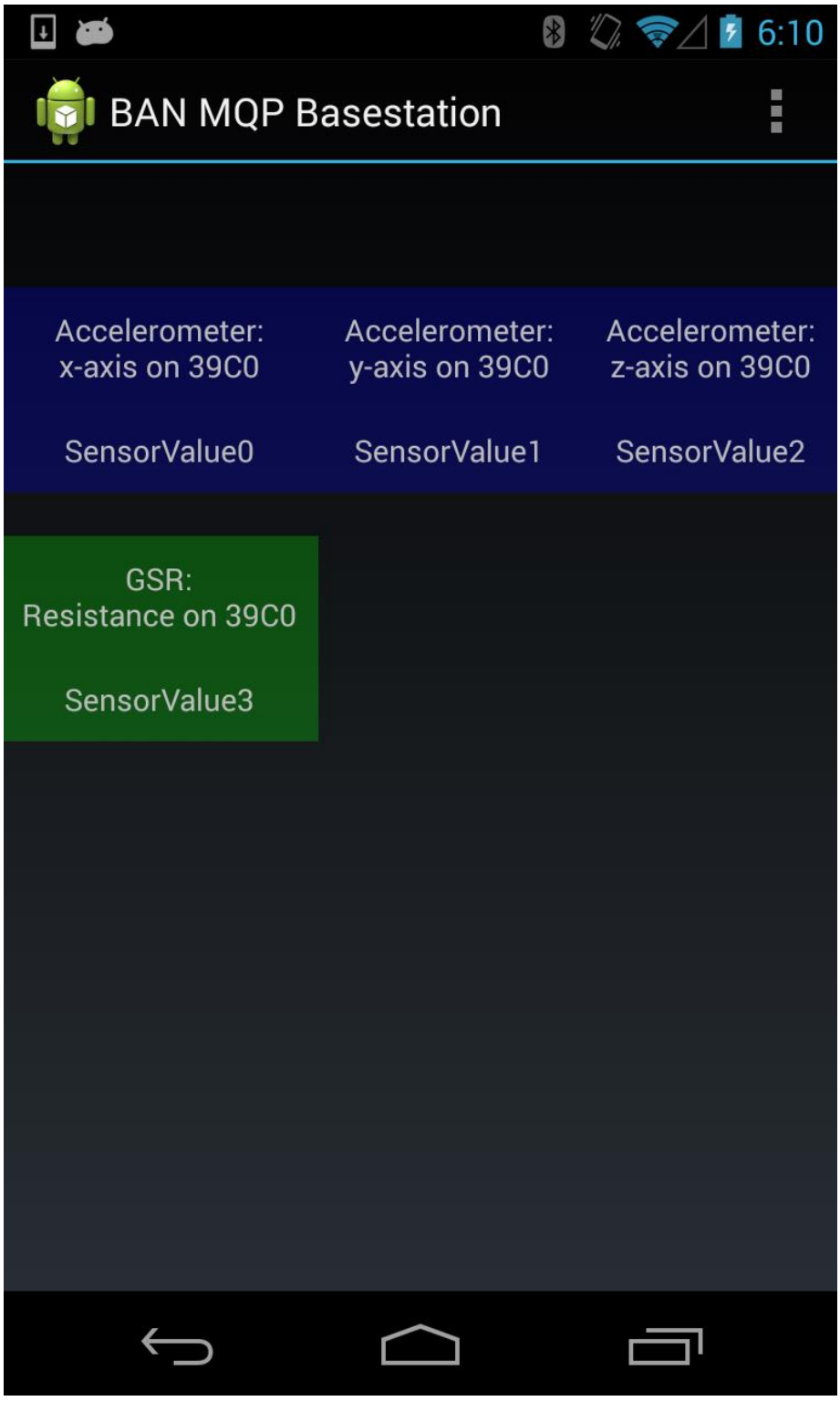


Figure 33: Base Station Application With One Mote Added

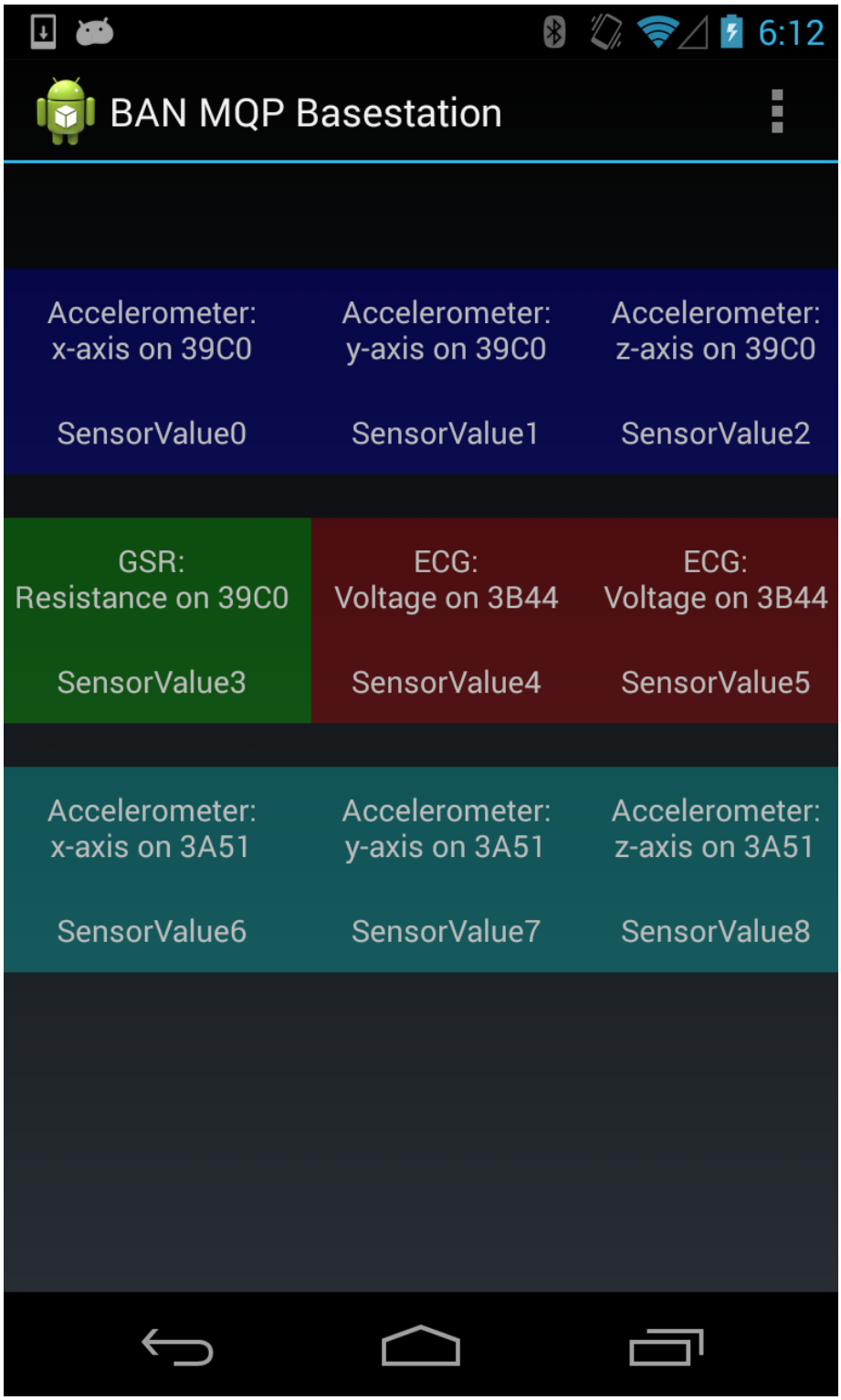


Figure 34: Base Station Application With Three Motes Added

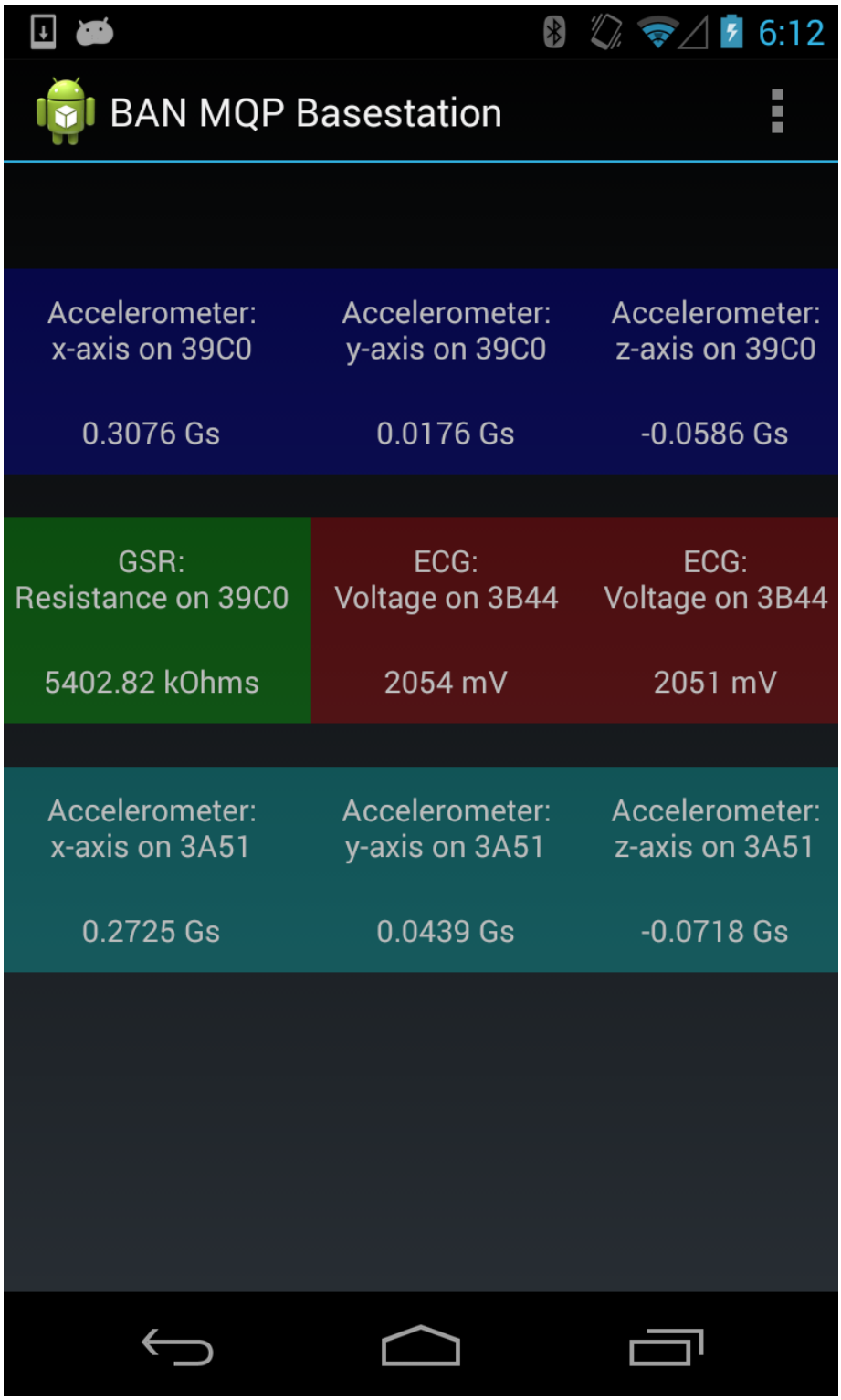


Figure 35: Base Station Application Recieving Data

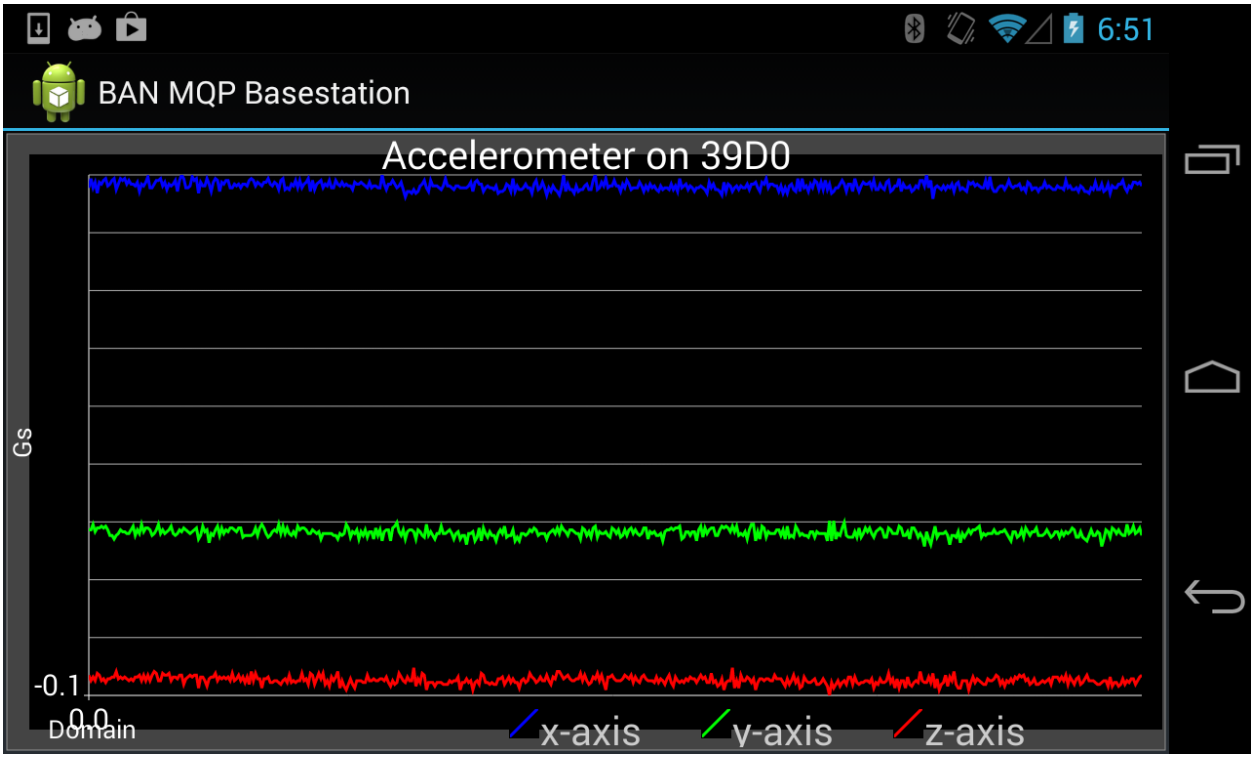


Figure 36: Base Station Application Graphing Resting 3-Axis Accelerometer

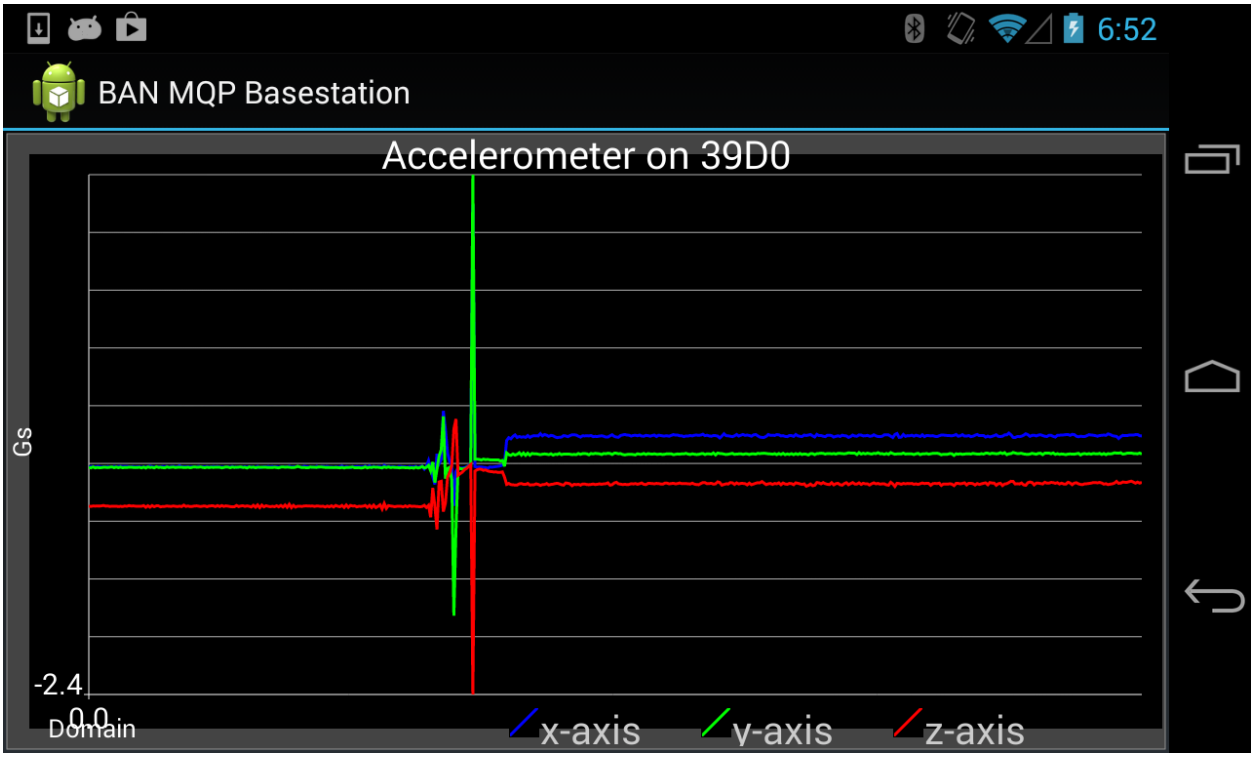


Figure 37: Base Station Application Graphing Dropped 3-Axis Accelerometer

5. Plug and Play BAN Performance Analysis

With such a radical new protocol design, we expected there to be performance costs involved. In order to gauge these costs, we performed a set of testing scenarios. For each scenario, results were gathered. Some of these results could be directly compared with our mote platform's published results. Other metrics helped map out the limits and restrictions of our plug and play BAN. Any metrics we were unable or did not test are also included, in the event they may affect future research.

5.1 Testing Scenarios

The majority of these tests were aimed to directly compare the costs of our Plug-and-Play design to the same platform without these design changes. Comparing to other BAN platforms would not be useful as the hardware changes alone would mean results could not be directly compared. The other tests were focused on displaying the costs of a Plug-and-Play design, as well as the limits of our BAN.

5.1.1 Shimmer Results

These were the results Shimmer published on their BtStream Firmware, a lightweight mote system that allows for data collection from almost all available sensors.

5.1.1.1 BAN Lifetime

Shimmer did not have any base station results. For comparison purposes, we assumed the base station battery life was not the minimum of the BAN. Since the BAN lifetime was defined as the minimum lifetime of a node, Shimmer mote lifetimes were enough to calculate this metric. The following table displays data gather by Shimmer documenting the battery life duration of sensor motes under varying configurations [40].

Sensors Enabled	Sampling Rate (Hz)	Battery Life Duration (hh:mm)
Accel	51.2	15:00
Accel	102.4	13:00
Accel & Gyro	51.2	12:00
Accel & Gyro	102.4	11:00
Accel, Gyro & Mag	51.2	10:30
Accel, Gyro & Mag	102.4	10:30
Accel & EMG	512	13:36
Accel & EMG	1024	12:05
EMG	512	13:50
EMG	1024	13:38
Accel & GSR	51.2	15:12
GSR	10.2	17:55

Table 3: Shimmer BtStream Firmware Lifetime

5.1.1.2 Shimmer BAN Throughput

Shimmer did not have any base station results. For comparison purposes, we assumed the base station was able to process at the given sampling rates without issue. The following table displays throughput and data rate limitations data provided by Shimmer for their BtStream firmware [4].

Packet Size (bytes)	Sensors Active (and byte costs)	Max Sampling Frequency (Hz)	Throughput	Goodput
9	Data Packet (x1) + Accelerometer (x6) + Timestamp (x2)	1024	9.216 KB/s	6.144 KB/s
11	Data Packet (x1) + Accelerometer (x6) + EMG (x2) + Timestamp (x2)	512	5.632 KB/s	4.096 KB/s
13	Data Packet (x1) + Accelerometer (x6) + ECG (x4) + Times	512	6.656 KB/s	5.12 KB/s

	tamp (x2)			
15	Data Packet (x1) + Accelerometer (x6) + Gyroscope (x6) + Timestamp (x2)	512	7.680 KB/s	6.144 KB/s
16	Data Packet (x1) + Accelerometer (x6) + Gyroscope (x6) + Heart Rate (x1) + Timestamp (x2)	512	8.192 KB/s	6.656 KB/s
17	Data Packet (x1) + Accelerometer (x6) + Gyroscope (x6) + Expansion (x2) + Timestamp (x2)	512	8.704 KB/s	7.168 KB/s
18	Data Packet (x1) + Accelerometer (x6) + Gyroscope (x6) + Expansion (x2) + Heart Rate (x1) + Timestamp (x2)	512	9.216 KB/s	7.68 KB/s
19	Data Packet (x1) + Accelerometer (x6) + Gyroscope (x6) + Expansion (x2) + Expansion (x2) + Timestamp (x2)	512	9.728 KB/s	8.192 KB/s
20	Data Packet (x1) + Accelerometer (x6) + Gyroscope (x6) + Expansion (x2) + Expansion (x2) + Heart Rate (x1) + Timestamp (x2)	512	10.24 KB/s	8.704 KB/s
21	Data Packet (x1) + Accelerometer (x6) + Gyroscope (x6) + Magnetometer (x6) + Timestamp (x2)	256	5.376 KB/s	4.608 KB/s

Table 4: Shimmer BtStream Firmware Throughput

5.1.2 Mote Lifetime Test

The purpose of the mote lifetime test was to measure the battery life of the BAN motes; specifically when a mote is active, sampling, and streaming data to the base station. We tested how

different sensor configurations affect the battery life of motes within the BAN. To do so, we ran motes in varying states.

This test was run with the following sensor configurations:

- Accelerometer
- Accelerometer and Gyroscope
- Accelerometer, Gyroscope, and Magnetometer
- Accelerometer and GSR
- GSR

Method

1. Charge mote on power station until battery is full
2. Charge the base station until battery is full
3. Flash motes with project firmware for applicable onboard sensors
4. Setup BAN operation with each device no more than a meter apart
5. Start the base station BAN application
6. Have the base station pair with the mote
7. Execute all protocol inquiries
8. Turn on the applicable onboard sensors
9. Set the sampling rate to the current value Shimmer used in their results for the current configuration.
10. Set the buffer size to 1; this should be the default (un-buffered sending)
11. Record the initial timestamp on the base station
12. Record and update the timestamp on the base station when data was last received
13. Collect data from the mote until it stops sending data

Metric Comparisons

The following metric comparisons were made directly from the test. Our expected outcome is included.

Sensor Configuration vs. Mote Battery Life

The battery life can be calculated by comparing the initial timestamp, and the timestamp of the last received data packet. The mote stopped sending data because the battery life was expended. The elapsed time since the base station requested data to the last data received is synonymous with the mote lifetime. Because this Plug and Play firmware had larger headers, we expect the battery life to be reduced, since the mote must send out larger packets per sample.

5.1.3 Control Message Latency Test

The control message latency test was designed to measure the amount of time required by a mote within the BAN to process and respond to protocol request packets. In particular, our testing was focused on the control messages used to initially teach the base station how to interact with a mote. Specifically: Sensor Inquiry, Command Inquiry, Command Parameters Inquiry, Command Returns Inquiry, and Data Inquiry messages. In doing so, we tested how different sensor configurations affect the time it takes the mote to construct an inquiry response and send it to the base station.

This test was run with the following sensor configurations:

- Accelerometer
- Accelerometer and Gyroscope
- Accelerometer, Gyroscope, and Magnetometer
- Accelerometer and GSR
- GSR
- Accelerometer and ECG
- ECG
- Accelerometer and EMG
- EMG

Method

1. Flash motes with project firmware for applicable onboard sensors
2. Setup BAN operation with each device no more than a meter apart
3. Start the base station BAN application
4. Have the base station pair with the mote
5. Execute each protocol inquiry one at a time
6. Have the base station record a timestamp for when the request was sent
7. Once the response is received, record another timestamp

Metric Comparisons

The following metric comparisons were made directly from the test. Our expected outcome is included.

Sensor Configuration vs. BAN Inquiry & Control Message Latency

To conduct this test, the base station sent out every possible plug and play protocol message to learn how to interact with the mote for data. This communication occurred at the beginning of each connection. For each of these packets, the time it took to receive a response was recorded. This information was graphed to see the relation of the sensors active on the firmware to how long it takes to process and respond to a Base Station request. We expected the variance in latency to be minimal, but would be greater than the latency of a static packet being sent. This test was repeated 1000 times for each inquiry type and sensor configuration.

5.1.4 Base Station Battery Life Test

Description

The base station battery life test was carried out to determine how long the base station would last under different operating circumstances. In doing so, we tested and observed how processing data affects the battery life of the base station platform.

This test was run with the following configuration:

- Base station running but processing no data
- Base station running and processing enough data to consume 100% of the CPU capacity

The results were graphed side-by-side as a bar graph.

Method

1. The base station platform will be fully charged
2. The base station platform will be disconnected from external power, and the base station application will be launched with the configuration described in one of the independent variable states
3. The base station application will log to a file the time that the application was launched
4. The base station application will log the current time and battery percentage every time a data packet is received from the mote or every 60 seconds, whichever is sooner.
5. Once the battery on the base station platform fails, the difference between the start time and the last logged periodic time will be the total time the battery lasted ± 60 seconds
    a. If the base station lasts longer than the motes used for testing, total battery lifetime will have to be extrapolated

6. The base station platform will be fully recharged before re-running the test for each additional independent variable state

5.1.5 BAN Throughput Test

Description

The BAN throughput test was utilized to determine the maximum sampling rate of motes in the BAN. In doing so, we then utilized the max sampling rate for a mote to calculate other statistics such as: transmission rate (Hz), throughput (KB/s), and goodput (KB/s). In particular, we were interested in seeing how our plug and play protocol overhead affects the max sampling rate and as a direct result the effective throughput and goodput of the motes.

This test was run with the following sensor configurations:

- Accelerometer
- Accelerometer and EMG
- Accelerometer and ECG
- Accelerometer and Gyroscope
- Accelerometer, Gyroscope, and Magnetometer

To determine the max sampling rate the test started at the one half the lowest sampling rate Shimmer recorded. Then, a binary search algorithm, similar how the size of an unbounded array is calculated, was utilized to narrow down the maximum sampling rate. If the range between the previous working rate and next rate to test became smaller than 32 Hz the test was halted. The results were graphed separately for each sensor configuration. Each sensor configuration had a different but constant packet size that we used to calculate throughput once the maximum sampling rate was determined.

Method

1. Charge mote on power station until battery is full
2. Charge the base station until battery is full
3. Flash motes with project firmware for applicable onboard sensors
4. Setup BAN operation with each device no more than a meter apart
5. Start the base station BAN application
6. Have the base station pair with the mote
7. Execute all protocol inquiries

8. Turn on the applicable onboard sensors
9. Set the sampling rate to the ½ the lowest sampling rate Shimmer recorded
10. Set the buffer size to 1 (un-buffered sending)
11. Attempt to receive streamed data on the base station
12. Verify whether sampling rate currently testing works
13. Continue to narrow down maximum sampling rate until found

Metric Comparisons

The following metric comparisons were made directly from the test. Our expected outcome is included.

Sampling Rate vs. Mote Throughput

Once the maximum sampling rate has been pinpointed, the throughput can be calculated by multiplying the sample rate with the size of each sample. Since the plug and play firmware has larger headers for each sample, we expect the maximum sampling rate to be less than Shimmer's sampling rate. However, the throughput should be about the same, because the packet sizes will have increased. Although, the goodput of plug and play BAN will most likely be less than that of Shimmer's BAN, because the increased packet sizes are not from samples, but from protocol headers.

Other Metrics

The following metrics were collected indirectly from the test.

Maximum Sampling Rate

A by-product of this test was calculating the limiting sampling rate for each sensor configuration. Any further tests should not exceed these maximum sampling rates. If they do, the tests should expect probable failure of the BAN.

5.2 Test Results

The following section details the results of our testing scenarios and is organized by results for each testing scenario described in the previous section.

5.2.1 Mote Lifetime Test Results

The comparison of mote lifetime between our firmware and BtStream revealed that our protocol and firmware has a negative impact on battery life. This is due to the additional overhead required when implementing our protocol. Data packets require the addition of a header, which was not included in BtStream. For details about the test scenario, see 5.1.2 Mote Lifetime Test

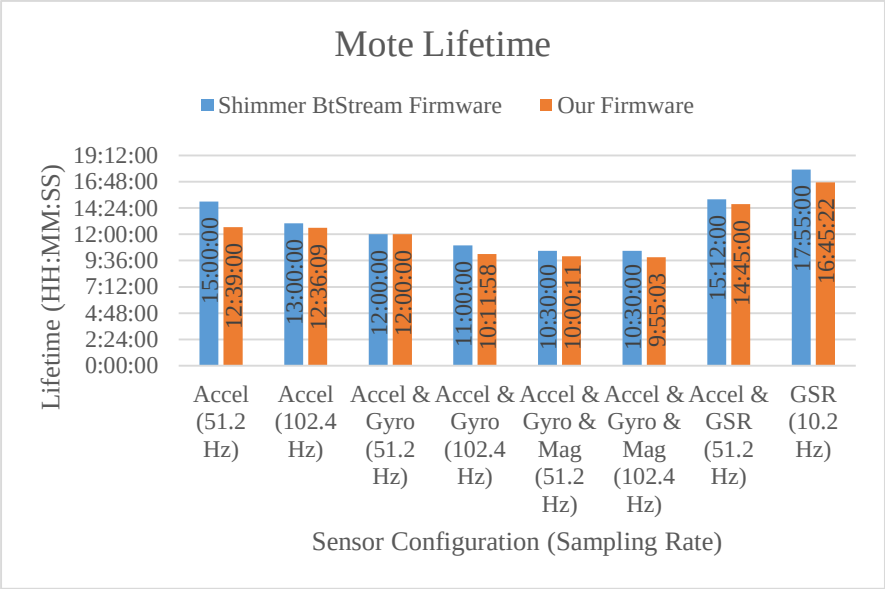


Figure 38: Mote Lifetime Graph

5.2.2 Control Message Latency Test Results

Below are the results from the test scenario described in section 5.1.3 Control Message Latency Test. These results summarize the latency associated with motes processing different types of control messages.

5.2.2.1 Sensor Inquiry Latency Test Results

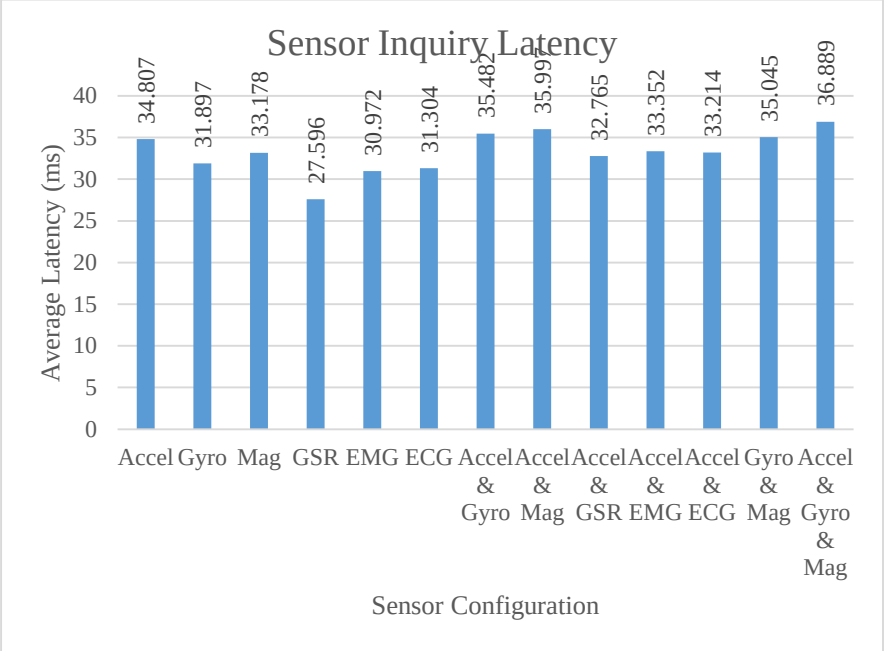


Figure 39: Sensor Inquiry Latency Graph

The average latency required to process and respond to a sensor inquiry varied depending on the mote configuration. In general, a higher number of sensors caused a higher latency. For details about the test scenario, see 5.1.3 Control Message Latency Test.

5.2.2.2 Command Inquiry Latency Test Results

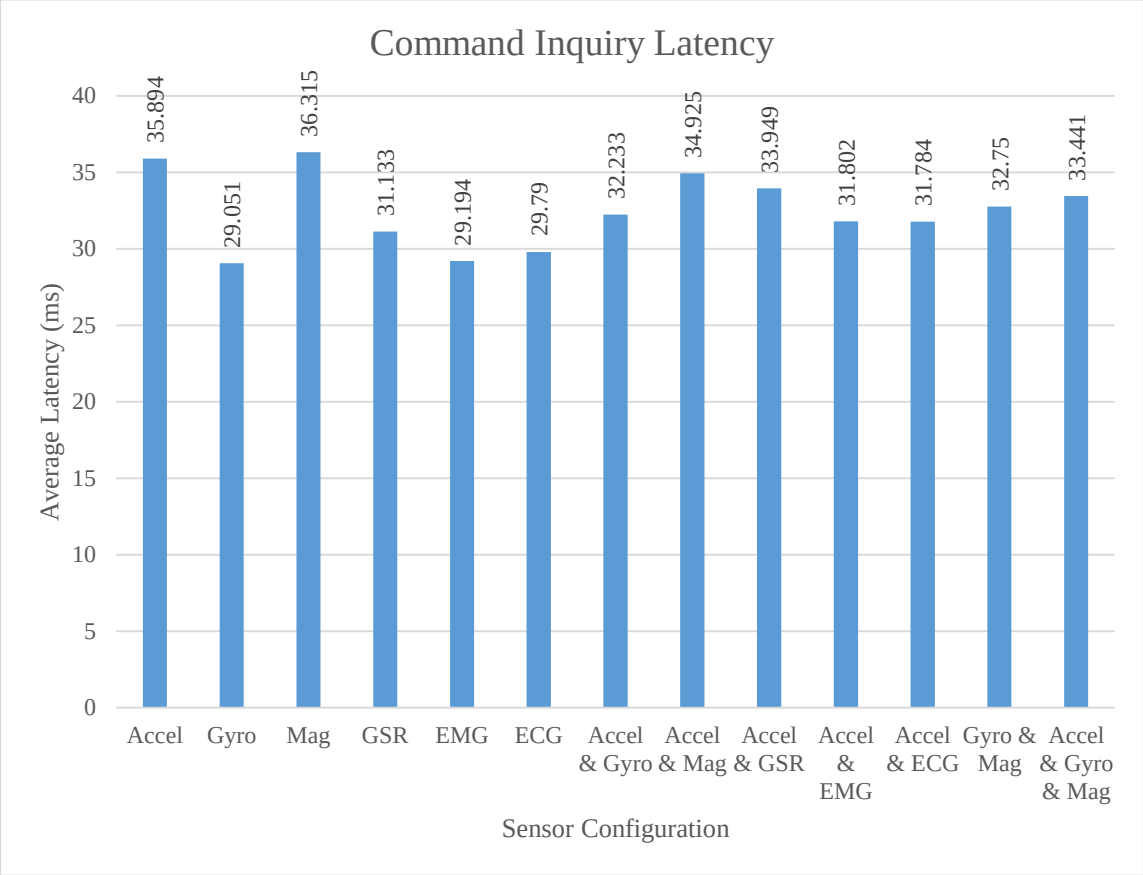


Figure 40: Command Inquiry Latency Graph

The time required to process command inquiries was fairly uniform regardless of mote configuration. Each configuration except for the gyroscope, EMG, and ECG had at least one command. Those three sensors had no commands implemented, so their responses were much shorter. For details about the test scenario, see 5.1.3 Control Message Latency Test.

5.2.2.3 Command Parameters Inquiry Latency Test Results

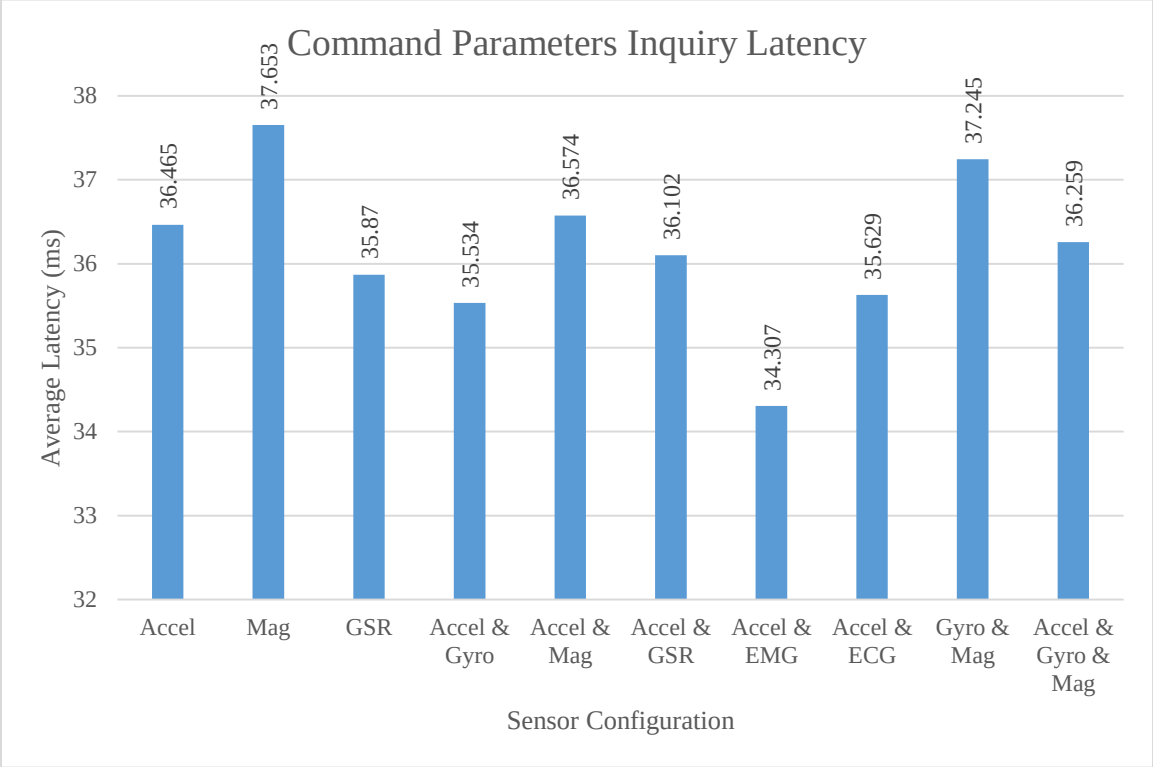


Figure 41: Command Parameters Inquiry Latency Graph

The latency for command parameters inquiry was fairly consistent. The gyroscope, ECG, and EMG had no commands implemented, and therefore had no command parameters. For details about the test scenario, see 5.1.3 Control Message Latency Test.

5.2.2.4 Command Returns Inquiry Latency Test Results

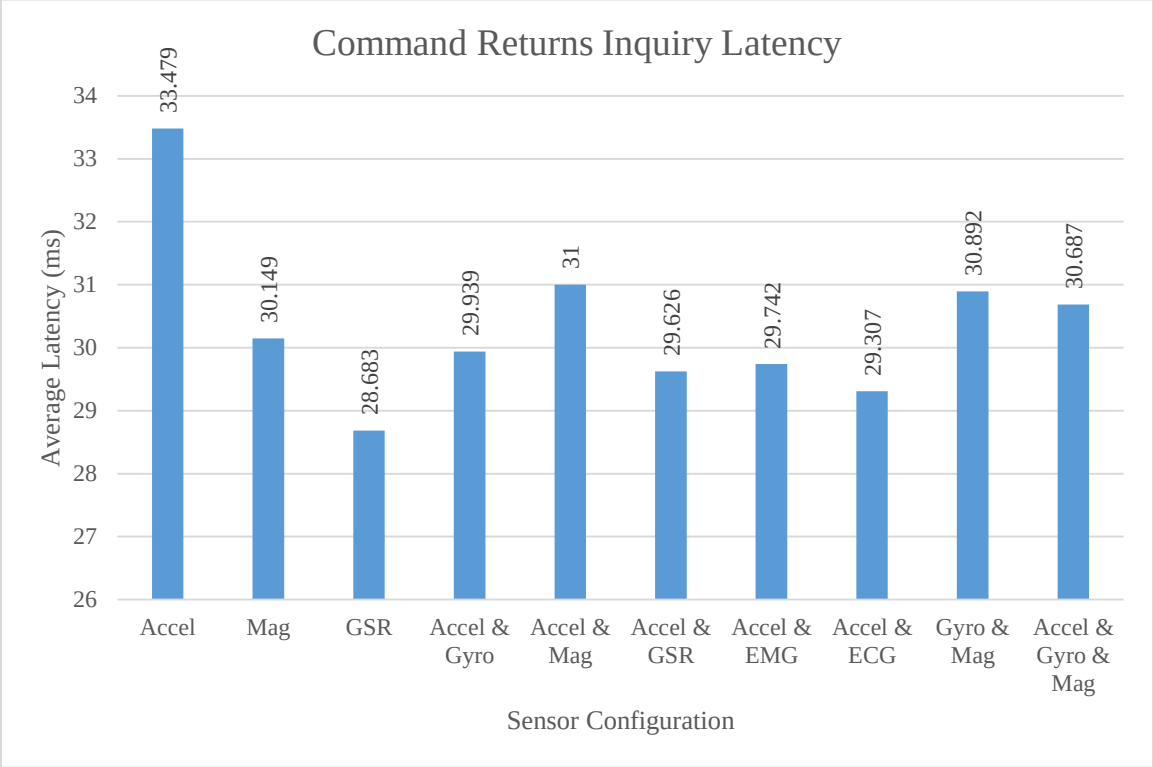


Figure 42: Command Returns Inquiry Latency Graph

The latency for the command returns inquiry was consistent across all test scenarios. The gyroscope, ECG and EMG had no custom commands implemented, and as such, they had no command returns to send. For details about the test scenario, see 5.1.3 Control Message Latency Test.

5.2.2.5 Data Inquiry Latency Test Results

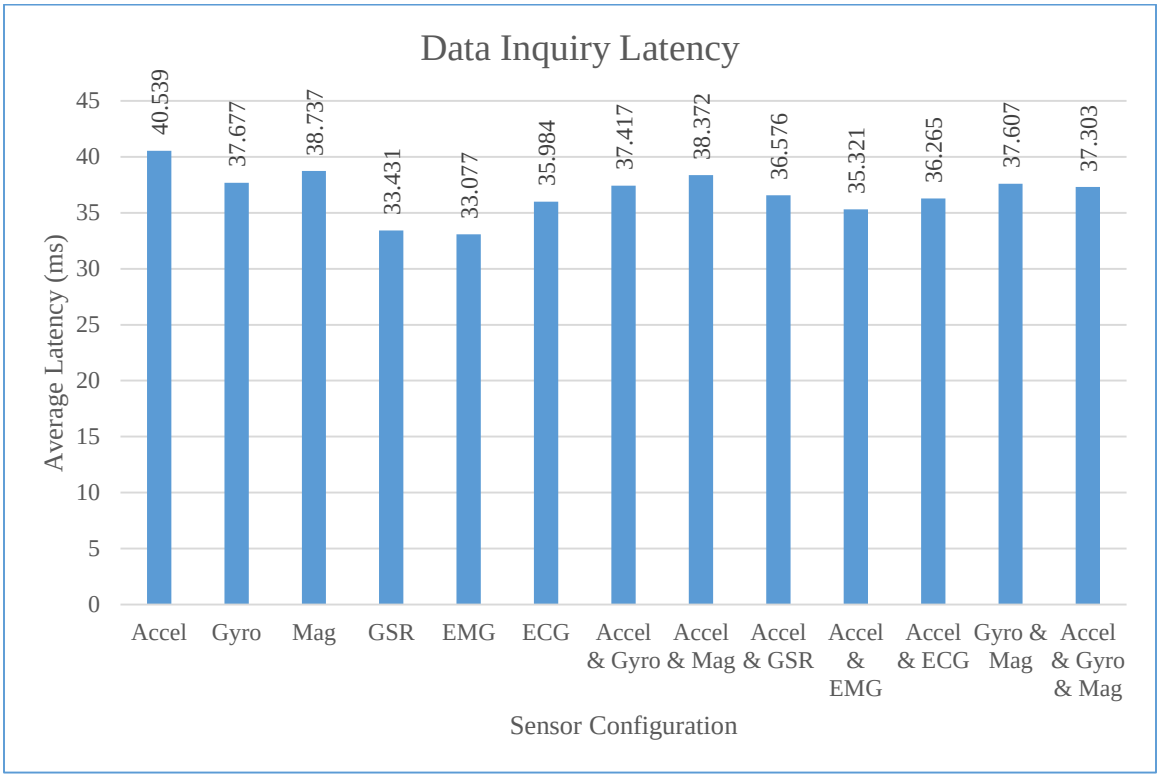


Figure 43: Data Inquiry Latency Graph

The latency between data inquiry requests and responses was highly consistent across all configurations. Each configuration sent at least one reply to the inquiry. Configurations with multiple sensors did not require additional time as each data inquiry packet queried for information about one sensor. For details about the test scenario, see 5.1.3 Control Message Latency Test.

5.2.3 Base Station Lifetime Test

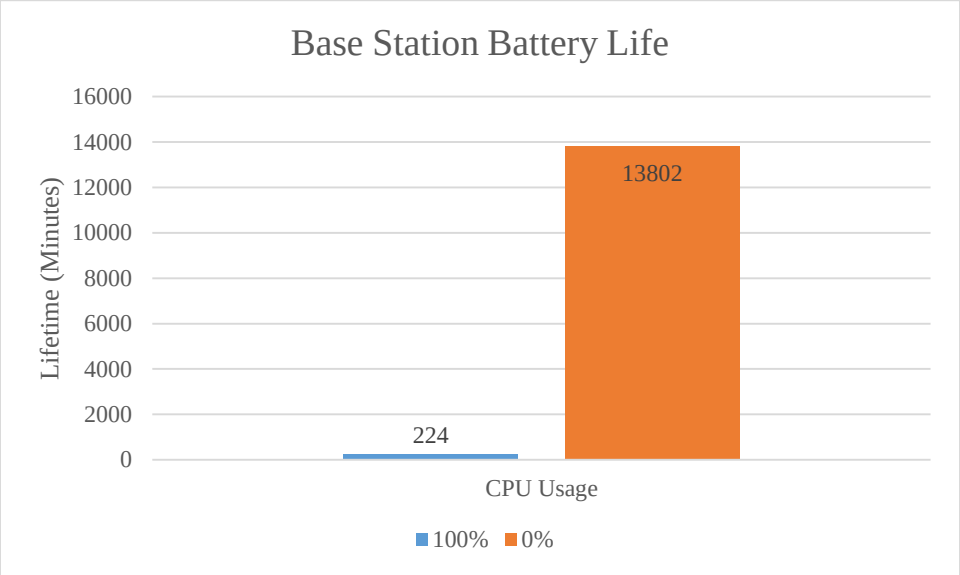


Figure 44: Base Station Lifetime Graph

This test was designed to evaluate the best and worst case lifetime of our base station. At idle, or 0% CPU load, we project the base station would last 13,802 minutes or about 9.5 days. However, at maximum capacity or 100% CPU load, the battery lasted only 224 minutes. These results are visualized above in Figure 44.

5.2.4 BAN Throughput Test Results

Sensor Configuration	Accel	Accel & EMG	Accel & ECG	Accel & Gyro	Accel & Gyro & Mag
Total Packet Size (bytes)	124	125	125	117	110
Header Size (bytes)	5	5	5	5	5
Total Sample Size (bytes)	7	10	12	14	21
Sample Overhead (bytes)	1	2	2	2	3
Buffer Size (number of samples)	17	12	10	8	5
Max Sampling Frequency (Hz)	512	320	256	256	128
Transmission Rate (Hz)	30.118	26.667	25.6	32.0	25.6
Throughput (KB/s)	3.735	3.333	3.200	3.744	2.816
Goodput (KB/s)	3.072	2.560	2.560	3.072	2.304

Table 5: Throughput Test Results

As expected, our protocol introduced overhead, which lowers the effective maximum sampling rate. As a direct result, the net throughput and goodput are lower than other protocols, such as BtStream. Our throughput was approximately one half of that achieved by Shimmer with BtStream [4] in all testing scenarios. The goodput we achieved was a little worse than half of the goodput in BtStream due to the additional overhead. Our investigation revealed that buffering samples on the motes before transmission significantly increased overall throughput and goodput. It is unclear from the BtStream documentation if they performed any buffering, however as BtStream has less overhead, it would be possible to buffer even more packets with BtStream than with our protocol. Finally, it may be possible to modify the architecture of our mote firmware to increase performance by limiting the amount of message passing between modules.

5.3 Untested Metrics

The following section outlines behavior or metrics that we did not observe or measure with the testing we carried out. In doing so, we highlight these variables and outline the potential effects of them within the context of our BAN. Depending on specific application of a BAN, these metrics may require testing before implementation.

5.3.1 Packet Loss

The loss of packets within the body area network is a potential adverse condition that would affect the operation or stability of the BAN. We were unable to actively test the effects of packet loss in our small-scale tests, due to reliability provided by the Bluetooth protocol. If the BAN were to suffer from packet loss, the two main issues that could arise are: the base station is unable to learn how to interact with a mote, sensor data from a mote either never makes it to the base station or is missing pieces. In the first situation, if the base station queries a mote for information and does not receive a response, then the base station can ask again after a pre-determined time out. This behavior helps to ensure that that base station can recover from small amounts of packet loss during the mote learning periods. In the second situation, if mote sensor data never makes it to the base station then the data displayed to a BAN user may be adversely affected. In a worst case scenario, where packet loss is 100% within the BAN, all communications between the base station and any connected mote would cease to work.

5.3.2 Transmission Distance

Within a wireless network, such as our BAN, the transmission distance between nodes in the network is a property that will likely affect performance and operation. Communication between devices in the BAN are limited by the physical limits imposed by the communication medium. In the case of Bluetooth, the communication medium implemented in our network, the transmission range of the devices vary by the class of Bluetooth NIC. The Shimmer motes operate class 2 Roving Networks RN-42 Bluetooth radios [40], which have a typical transmission range of approximately 10 meters. Under normal operations of the BAN, motes and the base station would likely be within that 10 meter distance. However, in situations where the devices stray outside that range, the quality of communications between nodes may degrade.

5.3.3 Maximum BAN Size

The maximum size of the BAN is currently limited largely by the communication protocol utilized, namely Bluetooth. A Bluetooth piconet consists of 1 master device and up to 7 slave devices under standard operation. In the case of the BAN, the base station acts the master while motes act as slaves. As a result, the maximum size of the BAN is limited to 8 devices. This limitation could be solved by the creation of a Bluetooth scatternet. The scalability of Bluetooth is discussed in C.3 Bluetooth Scalability.

5.3.4 BAN Goodput Over Time (Plug and Play Protocol Overhead)

The BAN goodput is defined as the throughput excluding any of the overhead associated with making the BAN plug and play e.g. the protocol overhead. The protocol overhead is necessary for the plug and play operation within our BAN. The amount of overhead is not static however, due to the variable properties of protocol packets. As a result, the effects of different amounts of protocol overhead could affect the performance of the BAN. Larger packets may take longer to process or require more CPU and battery usage to handle on both the mote and base station. In addition to the headers, there are control messages that need to be sent at the beginning of each connection to a mote. These messages allow the base station to learn what the mote is capable of. Due to this constraint, the goodput will start at 0% at the beginning of the connection, rapidly increasing after the control message discussion has been finished. In

most situations, the goodput will approach the throughput, as if the control messages were not needed. However, in extreme situations where motes are being changed constantly, or there is a high amount of packet loss, the control messages may still account of a large portion of BAN traffic.

5.3.5 CPU Usage

The CPU usage of both the mote firmware and the base station application was not a metric which we measured as part of the performance analysis of the BAN. While we was did not explore a CPU stress test of the base station or motes, such stress if high enough could affect the performance of the BAN. A mote that is overwhelmed in terms of CPU usage may be unable to process or response to requests or commands from the base station. A base station that is CPU overwhelmed may be able to process or display mote information or sensor data to a BAN user. Relative to Shimmer's BtStream firmware, the CPU usage is most likely higher due to the plug and play protocol overhead. The effect on CPU performance can be seen indirectly through the BAN lifetime results. A lower battery life would suggest that the firmware utilizes more resources than a firmware without plug and play overhead.

5.3.6 Data Streaming Latency

As opposed to control message latency, data streaming latency is the additional time it takes for a sample to be processed and sent to the base station. After the first packet, there is no explicit request sent for mote data. The mote simply collects another sample and sends it out without having to be instructed to do so. The mote can immediately act without waiting for request, so we expected the latency to be less than that of the observed control message latency. During our tests, it seemed the latency was almost non-existent. This is most likely due to timing imprecision either on the base station, the motes, or both. Currently, our results suggest that the data streaming latency is significantly less than control message latency, but the data is not precise enough to give a definitive answer. Any significant overhead in operating a BAN with our protocol will likely be introduced in the processing of control messages, the performance of which was evaluated in 5.2 Test Results

6. BAN Plug and Play Analysis

In order to evaluate the plug and play capabilities of previous Body Area Networks and our BAN, we generated the following rubric. Our understanding of the requirements for a BAN to be plug and play defined the criteria below. By our definition, a BAN is perfectly plug and play if the BAN configuration can change without any user effort. User effort includes new implementation, physical labor, or interactions with the base station application. For each criterion, we provide a range of possible amounts of user effort. This ranges from absolutely no effort, to the theoretical maximum work required. This allows for a more detailed evaluation beyond a series of pass or fails conditions. If a BAN meets the best case for each criterion, we consider it to be perfectly plug and play; there is no way to make it more plug and play according to our definition.

Criteria	**Grading**
Work required to add new mote to BAN	*Scale (1 = Best, 4 = Worst)* 1. Connect mote to Base Station 2. Flash mote with firmware before connecting to base station 3. Update base station software before connecting mote 4. Flash mote and update base station software before connecting mote
Work required to change sensor on mote	*Checklist (Less checks the better)* ➢ Physically change sensor ➢ Disconnect / reconnect mote ➢ Flash mote ➢ Update base station software
Work required to replace one mote with another in an operating BAN (assuming both motes are running firmware compatible with BAN)	*Scale (1 = Best, 2 = Worst)* 1. Disconnect old mote, connect new mote 2. Disconnect old mote, update base station software, connect new mote
Work required to add a completely new sensor to the BAN	*Checklist (Less checks the better)* ➢ Flash existing motes ➢ Update a small portion of the existing mote firmware (less than 25% of modules) ➢ Update a large portion of the existing mote firmware ➢ Update a small portion of the existing Base Station software (less than 25% of classes) ➢ Update a large portion of the existing Base Station software ➢ Updating the application layer protocol (new IDs for packet headers)
Work required to change the functionality of an existing sensor in the BAN	*Checklist (Less checks the better)* ➢ Flash existing motes ➢ Update a small portion of the existing mote firmware (less than 25% of

	modules) ➢ Update a large portion of the existing mote firmware ➢ Update a small portion of the existing Base Station software (less than 25% of classes) ➢ Update a large portion of the existing Base Station software ➢ Updating the application layer protocol (new IDs for packet headers)
Lines of code required to add a new command to a sensor	*Scale (1 = Best, 4 = Worst)* 1. Less than 100 lines 2. 100 - 500 lines 3. 500 - 1000 lines *More than 1000 lines*
Lines of code required to add a new sensor the BAN	*Scale (1 = Best, 4 = Worst)* 1. Less than 100 lines 2. 100 - 500 lines 3. 500 - 1000 lines *More than 1000 lines*

Table 6: Plug and Play Rubric

6.1 Plug and Play Evaluation: Shimmer BtStream vs. Our BAN Protocol

The table below displays rankings for both Shimmer's BtStream firmware and our plug and play protocol firmware using the plug and play rubric described above.

	Shimmer BtStream Firmware	**Our BAN Firmware**
Criteria	**Grading**	**Grading**
Work required to add new mote to BAN	*Scale (1 = Best, 4 = Worst)* 4. Flash mote and update base station software before connecting mote	*Scale (1 = Best, 4 = Worst)* 2. Flash mote with firmware before connecting to base station
Work required to change sensor on mote	*Checklist (Less checks the better)* ✓ Physically change sensor ✓ Disconnect / reconnect mote ○ Flash mote ✓ Update base station software	*Checklist (Less checks the better)* ✓ Physically change sensor ✓ Disconnect / reconnect mote ✓ Flash mote ○ Update base station software
Work required to replace one mote with another in an operating BAN (assuming both motes are running firmware compatible with BAN)	*Scale (1 = Best, 2 = Worst)* 1. Disconnect old mote, connect new mote	*Scale (1 = Best, 2 = Worst)* 1. Disconnect old mote, connect new mote

Work required to add a completely new sensor to the BAN	*Checklist (Less checks the better)* ○ Flash existing motes ✓ Update a small portion of the existing mote firmware (less than 25% of modules) ✓ Update a large portion of the existing mote firmware ✓ Update a small portion of the existing Base Station software (less than 25% of classes) ✓ Update a large portion of the existing Base Station software ✓ Updating the application layer protocol (new IDs for packet headers)	*Checklist (Less checks the better)* ○ Flash existing motes ✓ Update a small portion of the existing mote firmware (less than 25% of modules) ○ Update a large portion of the existing mote firmware ○ Update a small portion of the existing Base Station software (less than 25% of classes) ○ Update a large portion of the existing Base Station software ○ Updating the application layer protocol (new IDs for packet headers)
Work required to change the functionality of an existing sensor in the BAN	*Checklist (Less checks the better)* ○ Flash existing motes ✓ Update a small portion of the existing mote firmware (less than 25% of modules) ✓ Update a large portion of the existing mote firmware ✓ Update a small portion of the existing Base Station software (less than 25% of classes) ○ Update a large portion of the existing Base Station software ✓ Updating the application layer protocol (new IDs for packet headers)	*Checklist (Less checks the better)* ○ Flash existing motes ✓ Update a small portion of the existing mote firmware (less than 25% of modules) ○ Update a large portion of the existing mote firmware ○ Update a small portion of the existing Base Station software (less than 25% of classes) ○ Update a large portion of the existing Base Station software ○ Updating the application layer protocol (new IDs for packet headers)
Lines of code required to add a new command to a sensor	*Scale (1 = Best, 4 = Worst)* 2. 100 - 500 lines	*Scale (1 = Best, 4 = Worst)* 1. Less than 100 lines
Lines of code required to add a new sensor the BAN	*Scale (1 = Best, 4 = Worst)* 4. More than 1000 lines	*Scale (1 = Best, 4 = Worst)* 2. 100 - 500 lines

Table 7: Plug and Play Evaluation: BtStream vs. Our BAN

Although the Shimmer BtStream firmware allows for a functional BAN, it is not plug-and-play as can be seen in Table 7 above. Adding a new sensor requires writing a large amount of code, both for the mote and base station. The protocol relies heavily on hard-coded values, and has no ability to dynamically add or change functionality in the BAN. Finally, the firmware is designed to operate on a select group of devices, all of which are created by Shimmer; devices from other manufacturers could not operate in the same BAN.

Even though we attempted to develop a completely plug and play BAN, there are still areas that need improvement. Despite the fact that less work is needed to add a new mote to the BAN, the mote still needs to be flashed with the plug and play firmware. It may be possible to avoid this by having the mote automatically flashed somehow, possibly over air. Physically changing a sensor still requires a large amount of manual labor, mostly because the Shimmer hardware platform does not support automatic recognition of daughter board sensors. There is no way for the mote to tell which specific sensors are currently connected to the MSP430. Since the mote has to be flashed with the firmware for the proper sensors, the mote will also have to disconnect from the BAN. Luckily, no work has to be done to replace a mote in the BAN, since the base station can just relearn from the new mote when it connects. The greatest optimization seems to be in the amount of software development needed to extend the BAN. To add a new sensor to the ban, significantly less lines of code are needed, as well as fewer modules needing to be created or updated. Even less programming is required to just update the functionality of an existing sensor, because most of the labor is spent on designing the Inquiry command responses. Unlike Shimmer's firmware, our codebase is separated into many modules, with attempts to reuse modules and code wherever possible. We hope moving to other TinyOS platforms and further refactoring of our codebase will allow for greater flexibility and a better score on the rubric in the future.

7. Conclusion

The goal of this project was to design and carry out an initial implementation of a body area network protocol with a focus on plug and play. By incorporating a plug and play protocol into a body area network, the interactions between devices in the network can be simplified and streamlined. The protocol we designed succeeds at enabling a base station device to dynamically learn about mote capabilities and how to interact with them. No prior knowledge of the mote is required. The testing performed demonstrates that the protocol can be implemented and utilized with minimal impact on overall BAN performance. We believe that this protocol provides a solid foundation for any future commercial or research BAN development.

8. Future Work

The following details possible avenues of future work, and poses some questions for future researchers. In general, we thought that security would be a desirable next step in research. The combination of this project with a secure BAN would bring body area networks closer to practical, day to day use. We also described some concerns we had with our choices and left suggestions for future researchers to avoid the same minor setbacks.

8.1 Threat Modeling

If a future project was to focus on adding security to this BAN, it would be best to understand the security of the system in its current state. In order to do that, it would be best to use a systematic threat modeling and vulnerability identification approach. This will allow a concrete representation to "consider the design of the entire system, and not incorporate security technologies at random" [41]. We created a preliminary threat model for our BAN which can be found in appendix D.1 Threat Model We believe this to be the next step towards developing security for the current BAN platform.

8.2 Embedded System Cryptography

Security and cryptography were not within the scope of the project's implementation.. It is not hard to imagine that the security may be desired or in fact required within the context of a wireless sensor network, especially a body area network where the transmission of medical or other potentially sensitive information may occur. As a result, the implementation of application layer security on top of a body area network protocol is an area of interest. However, there are some issues that may need to be investigated. First, the sensor motes currently have severely limited computational and storage resources. The MSP430F1611 microcontrollers in the Shimmer motes have an 8 MHz processor, 48kB flash memory, and only 10kB RAM. Second, the firmware on the motes is implemented on the TinyOS platform using the nesC language. We have included some possible options for the Shimmer motes in appendix D.2 Embedded System Cryptography Libraries. However, if new mote platforms are integrated, these suggestions may not fit the new specifications. If future groups strive to implement cryptographically secure transmissions, they may want to focus on what to use for the motes.

8.3 Honeypot

A honeypot could be used to provide additional security to a BAN. Since a BAN usually consists of mobile, battery powered devices, a passive security model offers many benefits. A honeypot device would not require much resource utilization unless there is a present threat. Additionally, a honeypot would behave in a simpler manner than more active security measures, such as an intrusion detection system: there is no need for intensive packet sniffing and analysis with a honeypot. This means less computation, and therefore less battery power, is required to evaluate threats. We believe if a future group was to look into implementing security on top of this BAN protocol that the integration of a honeypot device should be considered. We recommend in-depth research on honeypots be carried out. In particular, some major aspects we believe could be addressed are: where a honeypot would be fit inside a BAN (e.g. what device a honeypot would operate on), how a honeypot would behave in a BAN, what information the honeypot would collect, as well as how that information would be used.

8.4 Alternative Communication

We used Bluetooth to connect our sensors with the base station. This decision was made because Bluetooth-enabled Android devices are readily available, and the Shimmer platform also supported Bluetooth. However, there are significant limitations which detract from the usefulness offered by the ubiquity of Bluetooth. A standard Bluetooth network is made up of individual connections between devices. As such, there is no way for a control device, such as a base station, to broadcast data to all connected devices. Similarly, it would be difficult to interconnect all the sensor platforms in our BAN, as Bluetooth pairings need at least a small amount of human interaction on one of the connecting devices, something not possible with our platform. Finally, the size of a Bluetooth network is limited to one Piconet, which can consist of one master, in our case the base station, and up to seven active slave devices. In order to increase the number of active devices in a Bluetooth network, some slave devices have to act as masters for a separate piconet, and relay all messages between them. This introduces additional overhead to those intermediary devices, which will reduce battery life and possibly impact performance.

One alternative that may scale better is IEEE 802.15.4 ZigBee [13]. By design, ZigBee is a mesh network with no central controller. ZigBee devices can transmit at a maximum of 250 kbps in the 2.4GHz band. The Shimmer platforms used for this BAN support ZigBee, however we were unable to locate a

ZigBee enabled Android device. In the future, there may be other mobile platforms with ZigBee available, or adapters to add ZigBee radios to mobile platforms.

The other alternative we found was Bluetooth Low Energy [42] (BLE). This is part of the Bluetooth 4.0 standard, but is not backwards compatible with previous Bluetooth 3.0 devices. BLE introduced the idea of device profiles, which generalize how the device performs. Many profiles already exist for healthcare related devices. The BLE protocol has specifications for different message types, including device state changes and data transmissions. This may make it easier to handle different types of messages arriving from multiple sources.

Future research projects using this platform may find it beneficial to investigate an alternative form of wireless communication. In addition, supporting multiple protocols enables multiple mote platforms to be integrated, regardless of their communication mechanism. While there are a few alternative options currently, the list may change in the future. Furthermore, the issues presented may be addressed, possibly making previous options more desirable. The choice of communication platform should be left up to future projects depending on their specific needs and limitations.

8.5 Alternative Sensing Platforms

While Shimmer provided a stable platform for development, there were some key features left to desire. The open source code and sensor documentation seemed less detailed than advertised. Many development steps for sensors involved reverse engineering the sparsely commented codebase because there were no specifications available. Additionally, any usage of existing Shimmer modules in the TinyOS platform led to trial and error because the modules were also underspecified. Having a platform with a plethora of schematics and input-output descriptions would have streamlined the development process. Having a TinyOS platform that supports simulation would have greatly reduced debugging time. Many TinyOS hardware platforms, such as the TelosB mote support simulation. This allows the mote firmware to be directly run on the development PC, with likely access to a debugger like GDB. Other platforms support debugging through connecting the mote to the development PC. The code can be debugged on the PC while it is executed on the native hardware. Since Shimmer did not support this, many problems had to be investigated through trial and error, alert statements, and logging. We recommend that future work should integrate a mote platform that better meets these needs.

8.6 Expanding the Plug and Play Protocol

We believe that there is significant potential for future extensions to this preliminary protocol and implementation. In particular, incorporating security on top of the protocol would increase the appeal for usage in a commercial body area network. Some starting points in regards to application security for TinyOS were outlined above in section 8.2. In addition, implementation and testing of the protocol on a multi-hop scatternet or larger-scale networks would serve to further validate the architecture. Inter-mote communication could be explored and would likely be required for an implementation of a multi-hop network or scatternet. The ability to update the plug and play protocol on-the-fly is a big area of potential expansion and experimentation. In particular, the ability to modify the grammars used to communicate information between the motes and the base station during BAN operation. Whatever future projects decide to approach, this body area network protocol should be flexible enough to use as a foundation.

9. References

[1] I. T. Sylla, "Body Area Networks: A way to improve remote patient monitoring," [Online]. Available: http://www.ecnmag.com/articles/2011/11/body-area-networks-way-improve-remote-patient-monitoring.

[2] "CodeBlue: Wireless Sensors for Medical Care | Harvard Sensor Networks Lab," [Online]. Available: http://fiji.eecs.harvard.edu/CodeBlue.

[3] V. Shnayder, B.-r. Chen, K. Lorincz, T. Fulford-Jones and M. Welsh, "Sensor Networks for Medical Care," [Online]. Available: http://www.eecs.harvard.edu/~mdw/papers/codeblue-techrept05.pdf.

[4] Shimmer, *BtStream Firmware User Manual.*

[5] Apple Inc., "Develop for IOS - Apple Developer.," [Online]. Available: https://developer.apple.com/technologies/ios/.

[6] Pebble, "Develop for Pebble," [Online]. Available: https://developer.getpebble.com/.

[7] "Android, the world's most popular mobile platform," [Online]. Available: http://developer.android.com/about/index.html.

[8] Emotiv, "Emotiv | EEG System | Electroencephalography.," [Online]. Available:

http://www.emotiv.com. [Accessed 19 October 2013].

[9] Shimmer, "Wearable Sensor Technology | Shimmer | Wearable Wireless Sensing Technology and Solutions.," [Online]. Available: http://www.shimmersensing.com/.

[10] Emotiv, "EPOC Features," [Online]. Available: http://emotiv.com/epoc/.

[11] Emotiv, "EEG Features," [Online]. Available: http://emotiv.com/eeg/.

[12] Emotiv, "Developer & Research Packages.," [Online]. Available: http://emotiv.com/store/dev.php.

[13] ZigBee, "ZigBee Specification Overview," [Online]. Available: http://www.zigbee.org/Specifications/ZigBee/Overview.aspx.

[14] Bluetooth SIG, Inc., "Bluetooth Technology Website," [Online]. Available: http://www.bluetooth.com/Pages/Bluetooth-Home.aspx.

[15] Texas Instruments Incorporated, "MSP430F1611 (ACTIVE) 16-bit Ultra-Low-Power MCU, 48kB Flash, 10240B RAM, 12-Bit ADC, Dual DAC, 2 USART, I2C, HW Mult, DMA.," [Online]. Available: http://www.ti.com/product/msp430f1611.

[16] Texas Instruments Incorporated, "Microcontrollers (MCU) Overview for MSP430 Ultra-Low Power 16-bit MCUs.," [Online]. Available: http://www.ti.com/lsds/ti/microcontroller/16-bit_msp430/overview.page.

[17] S. Evanczuk, "The most-popular MCUs ever," 20 August 2013. [Online]. Available: http://www.edn.com/electrical-engineer-community/industry-blog/4419922/9/Slideshow--The-most-popular-MCUs-ever. [Accessed 2 March 2014].

[18] Shimmer, "Shimmer Sensor Application Development Tools," [Online]. Available: http://www.shimmersensing.com/research-and-education/applications/body-sensor-network-applications-development-1/.

[19] Shimmer Research, "Shimmer is Honoured by Frost & Sullivan’s Product Leadership Award," 28 August 2013. [Online]. Available: http://www.shimmersensing.com/news/shimmer-is-honoured-by-frost-sullivans-product-leadership-award-for-its-hig. [Accessed 2 March 2014].

[20] TIK WSN Research Group, "The Sensor Network Museum," 2014. [Online]. Available: http://www.snm.ethz.ch/snmwiki/Main/HomePage. [Accessed 2 March 2014].

[21] TinyOS, "TinyOS Home Page," [Online]. Available: http://www.tinyos.net.

[22] Contiki, "Contiki: The Open Source OS for the Internet of Things," [Online]. Available: http://www.contiki-os.org/.

[23] RIOT, "The Friendly Operating System for the Internet of Things. Learn More," [Online]. Available: http://www.riot-os.org/.

[24] U.C. Berkeley EECS Department, "TinyOS FAQ," [Online]. Available: http://webs.cs.berkeley.edu/tos/faq.html#SEC-16.

[25] TinyOS, "TinyOS Community," [Online]. Available: http://www.tinyos.net/community.html.

[26] A. Dunkels, "Adam Dunkels," [Online]. Available: http://www.dunkels.com/adam/.

[27] Contiki, "The Contiki Community," [Online]. Available: http://www.contiki-os.org/community.html.

[28] O. H. M. W. M. G. T. C. S. E. Baccelli, "RIOT: One OS to Rule Them All in the IoT," INRIA, 2012.

[29] O. H. M. G. M. W. T. C. S. Emmanuel Baccelli, "RIOT OS: Towards an OS for the Internet of Things," *Proceedings of the 32nd IEEE International Conference on Computer Communications (INFOCOM), Poster Session*, 2013.

[30] TinyOS, "TinyOS Github - nesC," [Online]. Available: https://github.com/tinyos/nesc.

[31] The Eclipse Foundation, "Eclipse - The Eclipse Foundation Open Source Community Website.," [Online]. Available: https://www.eclipse.org/.

[32] Distributed Computing Group, "Yeti 2 - TinyOS 2 Plugin for Eclipse.," [Online]. Available: http://tos-ide.ethz.ch/wiki/index.php.

[33] FusionForge, "FusionForge: Welcome," [Online]. Available: http://fusion.wpi.edu/.

[34] "Android SDK," [Online]. Available: https://developer.android.com/sdk/index.html.

[35] "Exp4j," [Online]. Available: http://www.objecthunter.net/exp4j/.

[36] "achartengine - Charting library for Android," [Online]. Available: https://code.google.com/p/achartengine/.

[37] "chartdroid - native chart engine for android," [Online]. Available: https://code.google.com/p/chartdroid/.

[38] "GraphView," [Online]. Available: http://android.arnodenhond.com/components/graphview.

[39] "Androidplot," [Online]. Available: http://androidplot.com/.

[40] Shimmer, *Shimmer User Manual.*

[41] S. Myagmar, A. J. Lee and W. Yurcik, *Threat Modeling as a Basis for Security Requirements,* 2005.

[42] Bluetooth, "Low Energy | Bluetooth Technology Website," [Online]. Available: http://www.bluetooth.com/Pages/Bluetooth-Smart.aspx.

[43] Shimmer, "Shimmer Android Instrument Driver," [Online]. Available: http://www.shimmersensing.com/shop/shimmer-android-id. [Accessed 2013 October 19].

[44] "ShimmerResearch," [Online]. Available: https://github.com/ShimmerResearch/.

[45] "Manifest.permission," [Online]. Available: http://developer.android.com/reference/android/Manifest.permission.html.

[46] "Security Tips," [Online]. Available: http://developer.android.com/training/articles/security-tips.html.

[47] Bluetooth, "Simple Pairing Whitepaper," 3 August 2006. [Online]. Available: http://web.archive.org/web/20061018032605/http://www.bluetooth.com/NR/rdonlyres/0A0B3F36-D15F-4470-85A6-F2CCFA26F70F/0/SimplePairing_WP_V10r00.pdf.

[48] S. Radack, "Security of Bluetooth Systems and Devices: Updated Guide Issued by the National Institute of Standards and Technology (NIST).," *ITL BULLETIN FOR AUGUST 2012.*

[49] C. Petrioli, S. Basangi and I. Chlamtac, "Configuring Bluestars: Multihop Scatternet Formation For Bluetooth Networks.," *IEEE Transactions on Computers 52.6,* pp. 779-790.

[50] T.-Y. Lin, Y.-C. Tseng, K.-M. Chang and C.-L. Tu, "Formation, routing, and maintenance protocols for the BlueRing scatternet of Bluetooths," *System Sciences,* 2003.

[51] G. Zaruba, S. Basagni and I. Chlamtac, "Bluetrees-scatternet formation to enable Bluetooth-based ad hoc networks," *Communications,* pp. 273-277, 2001.

[52] M. Medidi and A. Daptardar, *A Distributed Algorithm for Mesh Scatternet Formation in Bluetooth Networks,* CSREA Press, 2004, pp. 295-301.

[53] S. Sunkavai and B. Rarnalmurthy, "MTSF: a fast mesh scatternet formation algorithm for Bluetooth networks," *Global Telecommunications Conference*, pp. 3594-3598, 2004.

[54] M. Rabbi, H. Chayon and T. Rahman, "Bluetooth's Compatibility for Large Scale Wireless Sensor Network," *Communication Technology*, pp. 1,5,27-30, 2006.

[55] TinyOS, "TinyOS Wiki - TinySec," [Online]. Available: http://tinyos.stanford.edu/tinyos-wiki/index.php/TinySec.

[56] C. Karlof, N. Sastry and D. Wagner, "TinySec: A Link Layer Security Architecture for Wireless Sensor Networks," in *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, 2004.

[57] D. Cvrcek, "Security of TinySec," 2008. [Online]. Available: http://www.cl.cam.ac.uk/research/security/sensornets/tinysec/. [Accessed 20 Dec 2013].

[58] D. Cvrcek, "Counters, Freshness, and Implementation," 7.

[59] M. Luk, G. Mezzour, A. Perrig and V. Gligor, "MiniSec: A Secure Sensor Network Communication Architecture.," *Information Processing in Sensor Networks*, pp. 479-88, 2007.

[60] G. M. A. P. a. V. G. Mark Luk, "MiniSec: A Secure Sensor Network Communication Architecture," [Online]. Available: https://sparrow.ece.cmu.edu/group/minisec.html.

[61] Willow Technologies, "TELOSB MOTE PLATFORM," [Online]. Available: http://www.willow.co.uk/html/telosb_mote_platform.html.

[62] P. Ning, "TinyECC: Elliptic Curve Cryptography on TinyOS (Version 2.0)," [Online]. Available: http://discovery.csc.ncsu.edu/software/TinyECC.

[63] A. Liu and P. Ning, "TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks," *Information Processing in Sensor Networks*, pp. 245-56, 2008.

[64] TIK WSN Research Group, "The Sensor Network Museum - MicaZ," [Online]. Available: http://www.snm.ethz.ch/snmwiki/Projects/MicaZ.

[65] Memsic Inc., "TelosB Datasheet," [Online]. Available: http://www.memsic.com/userfiles/files/Datasheets/WSN/telosb_datasheet.pdf.

[66] TIK WSN Research Group, "The Sensor Network Museum - Tmote Sky," [Online]. Available:

http://www.snm.ethz.ch/snmwiki/Projects/TmoteSky.

[67] TinyOS, "TinyOS Wiki - Imote2," [Online]. Available: http://tinyos.stanford.edu/tinyos-wiki/index.php/Imote2.

[68] A. Perrig, R. Szewczyk, J. Tygar, V. Wen and D. E. Culler, "SPINS: Security Protocols for Sensor Networks," *Wireless Networks 8.5*, pp. 521-34, 2002.

[69] S. Pelissier, "Cryptography algorithms for TinyOS," [Online]. Available: http://tinyos.cvs.sourceforge.net/viewvc/tinyos/tinyos-2.x-contrib/crypto/index.html.

[70] "NetSensorAes - A Simple NesC Implementation of AES," [Online]. Available: https://code.google.com/p/netsensoraes/.

[71] C. A. Hawkins, P. N. Lucia and G. Rubio, "The WPI Scheduler Project," 2007.

[72] "Wireless Sensor Research," [Online]. Available: http://www.shimmer-research.com/r-d.

[73] "Project Ubertooth," [Online]. Available: http://ubertooth.sourceforge.net/.

[74] "Java[TM] Charts for Android," [Online]. Available: http://www.java4less.com/charts/chart.php?info=android.

[75] "Android plot library - achartengine or Androidplot?," [Online]. Available: http://stackoverflow.com/questions/17268840/android-plot-library-achartengine-or-androidplot.

[76] "Android 4.3 APIs.," [Online]. Available: https://developer.android.com/about/versions/android-4.3.html.

Appendices

These appendices detail supplementary information that interested readers may find useful. These summaries may be useful to any future research teams looking to extend or our efforts. Section Appendix A: Shimmer includes our initial research into the Shimmer platform. Section Appendix B: Android

Research contains an overview of the Android device used as our base station, as well as a brief security analysis. Section Appendix C: Bluetooth has a brief overview of Bluetooth security, reliability, and scalability. Finally, Section Appendix D: BAN Security contains a security analysis of BANs in general, as well as some potential mitigation techniques.

Appendix A: Shimmer

Following is an inventory and analysis of the out-of-the-box capabilities of our mote platform.

A.1 Mote Baseboards and Expansions

The Shimmer sensor platform development kit includes 6 baseboards as well as six different sensor hardware expansion modules. Shimmer uses a modular system in that different expansion modules can be attached quickly and easily to a baseboard platform. This makes swapping modules and constructing a variety of mote configurations straightforward. The baseboard consists of a few major components, an MSP430 microcontroller, an expansion connector, and one built-in accelerometer sensor. On its own, a baseboard platform with no attached modules can be used as a basic sensing platform using the built-in accelerometer sensor. However, Shimmer includes a variety of expansion modules, which greatly increase the capabilities and applications of the sensor platforms. These expansion modules are separate boards that are simply attached to the baseboard and extend the sensing capabilities of the default device. The following is an overview of each of the six sensor expansion modules Shimmer provides and their applications:

1. Galvanic Skin Response (GSR) Module: the GSR sensor measures skin conductivity between two electrodes (not included) attached to two fingers on one hand.
2. Electrocardiogram (ECG) Module: the ECG sensor records the electrical activity through the heart muscles over a period of time using multiple electrodes (not included) attached to the skin.
3. Electromyogram (EMG) Module: the EMG sensor measures electrical activity associated with muscle contractions using electrodes (not included) attached to the skin.
4. Strain Gauge Module: the strain gauge sensor enables the measurement of strain and force utilizing an attached load cell or other instrument (not included).

5.9 Degrees of Freedom (9DoF) Module: the 9DoF expansion measures static and dynamic orientation data, inertial measurement, and complex 3D motion sensing. This expansion consists of both a gyroscope and magnetometer sensor.

6. GPS Module: the GPS sensor provides GPS data, barometric pressure, and temperature measurements.

For the expansion modules that require electrodes to record sensor data, such as the GSR, ECG, and EMG modules, we researched and invested in compatible electrodes to enable us to both test those modules during development as well as make demonstrations of the system possible. The strain gauge module requires the usage of an external instrument such as a load cell to measure force data. Research was done into compatible load cells that could be used in the BAN with limited success. The vast majority of available load cells and force gauges are built for and used in industrial, automotive, or highly specific mechanical applications such as inside machinery. With such a limited availability of both compatible and usable load cells it was decided to not utilize the strain gauge mote for its original purpose. This mote could be used as a replacement, or possibly a BAN level monitor.

A.2 Android Driver and SDK

Shimmer Sensing provides an Android Instrument Driver [43] free of charge to anyone with an account on shimmersensing.com. This driver is intended to simplify the development process when creating android applications that interact with Shimmer sensors. We downloaded the driver, which came with the source code for several android applications designed to test the Shimmer sensors. We were able to compile and deploy one of these applications to the base station phone to verify that it could communicate and interact with the sensors appropriately.

A.3 Shimmer BtStream Analysis

Originally named BoilerPlate, BtStream is the most robust firmware included with the Shimmer Sensing platform. Intended for commercial use, installing and using the firmware requires no technical knowledge beyond interacting with a USB hub on a pc, and operating a basic Bluetooth device. Once the mote is programmed, the user must also install and run one of Shimmer's stock applications on the device they intend to use as a base station. Anyone who can pair a Bluetooth headset with their smartphone or laptop, and download an application should have no issues getting the shimmer mote streaming sensor data to their chosen base station.

Shimmer has included a BtStream user guide [4] as well as a Shimmer Revision 2 user guide [40] to answer any questions about using the firmware, or get developers started on building their own firmware and base station application. Since the firmware was intended for commercial use, both guides do not go into intensive technical detail; those details are better found in their code documentation and BtStream GitHub repository [44]. Commands sent to and from the mote are split into five subsections: set commands, get commands, action commands, acknowledgements, and streaming data. Set commands simply set what is specified, followed by the value to set it to on the mote. Examples of set commands include sensor calibrations, buffer sizes, sampling rates, and sensor ranges. Action commands are merely a subset of set commands that do not require values to be set; the mote is still instructed to do something, but it does not require additional information following the command. These include toggling the mote's LED, as well as starting and stopping sensor data streaming. Get commands are simple inquires to the mote to retrieve information, such as the current sensor configuration or mote sampling rate. When a mote receives and processes a command, it will always send an acknowledgement back to the base station. If it was a get command, the mote will also include the data requested from the base station. Streaming data packets are simply the sensor data sent to the base station. The packet structure is agnostic to which sensor is sending data, and either sends 8-bit or 16-bit values.

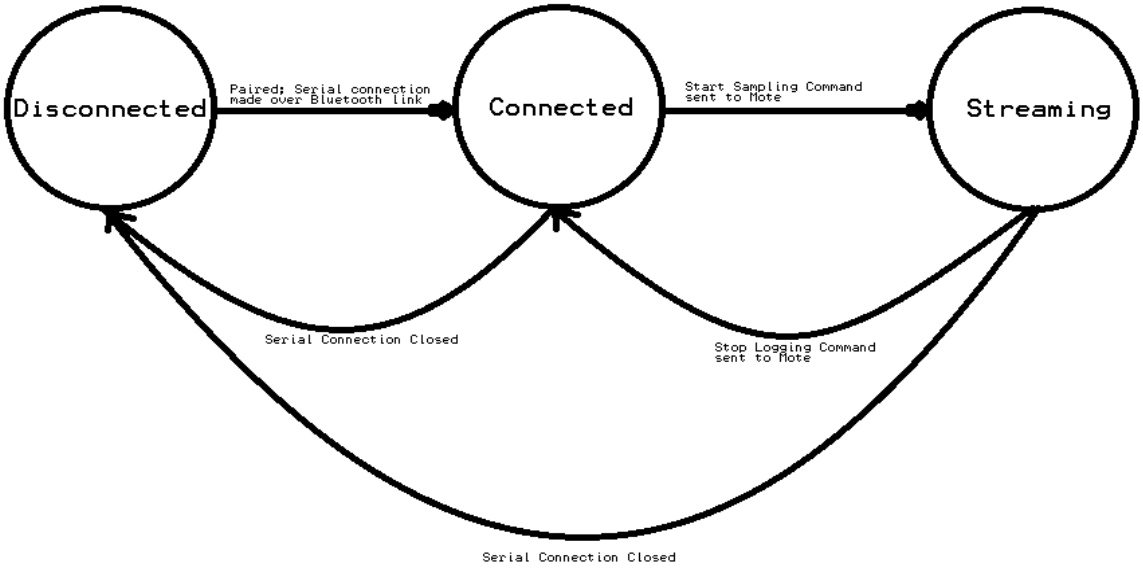


Figure 45: BtStream State Transition Table

The firmware separates the mote's behavior into three distinct states. In the disconnected state, the mote simply waits to be paired with. It will only enter discoverable mode to advertise itself in limited intervals to save battery. However, it will respond to any scanning requests received. Once the base station has paired and opened a serial socket connection with the mote, the mote will enter the connected state. In this state, the mote can process set, get, and action commands from the base station. The mote will return to the disconnected state if the socket is ever closed. If the mote receives a start streaming action command, it will enter the streaming state. In the streaming state, the mote will send streaming data to the base station; however, the mote can also still process set commands if on-the-fly configuration is necessary.

Overall, the BtStream firmware is designed well for commercial or basic academic purposes. Using the BtStream firmware and Multi-Shimmer Sync for Windows, Shimmer reports that most of their BAN tests could only reach a sampling frequency of 512 Hz without packet loss (the best case being 1024 Hz, worst case being 256 Hz). Similar lab tests show a streaming latency of 100±50ms. The battery life of each mote using BtStream ranged from 10.5 hours to almost 18 hours. The packet structure and behavior state abstractions worked nicely, making it easy to understand, and presumably easier to implement as well.

Notwithstanding, there are a number of potential downsides of the BtStream firmware. First off, security was not in the scope at all for designing this firmware. If this firmware was to be used for a Body Area Network outside an academic environment, transmissions could be easily intercepted and altered. Integrity is of utmost concern for a BAN consisting of medical devices. Secondly, the base station commands are overly agnostic: meaning, you can send a set command to enable a sensor that a mote doesn't have, or receive data and not necessarily know what sensor that data belongs to. Additionally, many of the packet structures have wasted space for future use. If we were to use the packet structures as a foundation for extensions, there would be much unnecessary overhead incurred.

Appendix B: Android Research

Below was some initial research done on our base station choice and security capabilities of the Android platform.

B.1 Nexus 4 / Android 4.3 Setup and Initial Testing

The base station of this BAN was a Google Nexus 4 Phone, running Android 4.2. This phone was selected because it has modern hardware, and runs the most recent version of the Android Operating System. Most cell phone carriers add additional software or features to phones that they sell. By purchasing the Nexus 4 directly from Google and without a cellular contract one can avoid limitations imposed by carriers.

The phone was updated to Android 4.3, which was the latest version available at the beginning of the project. This was done to add support for developing for Bluetooth Low Energy (BLE) devices, however we later discovered that the sensor platforms used were not BLE compliant. We were able to set up an Android development environment [34] on their computers, and deploy a test application to the base station to verify that the phone and development environments both behaved as expected. No further testing was performed before beginning development.

B.2 Application Security

We researched application security on Android devices. As the BAN is intended to collect medical data, the privacy and integrity of that data was to be kept paramount. Ideally, the application would run in an Android Virtual Machine on the base station, where it could be isolated from all other applications and processes on the phone. Unfortunately, there were no working implementations of Android VM's that ran on mobile devices available during the project.

Android applications often require the user to grant certain permissions when they are installed for the first time. As our application requires the use of the Bluetooth adapter, it must request the BLUETOOTH permission. Similarly, as the application is supposed to be able to enable and disable the blue tooth adapter, it requires the BLUETOOTH_ADMIN permission. A full list of these permissions and the associated access provided can be found on the Android Developer website [45]. When installing an application, a user must either accept all permissions requested by that application, or refuse and not let the application be installed. There is no mechanism to accept a partial permission list. By requiring the user to grant these permissions, the user is informed of the control the application will have on the host device.

Android does provide a reasonable level of security and isolation for individual applications. Each app runs in a unique Dalvik Virtual Machine. No communication between DVM's is possible unless desired by the application developers. Furthermore, any files created on the phone's internal storage will only be accessible to the application that created it. However, any data stored on external storage, such as

a Micro-SD card, could be read by any application with the appropriate permissions [46]. In order to increase the privacy of the data collected, we decided that any data stored should be kept on the internal storage of the phone. If any information must be stored on the phone's external storage, that information should be encrypted to prevent other applications from reading it.

Appendix C: Bluetooth

The following sections describe the benefits and disadvantages of Bluetooth from three different perspectives. Each of these criteria is paramount for the robustness of a communication protocol.

C.1 Bluetooth Security

The Bluetooth protocol, as of version 2.1, claims to provide Authentication and Confidentiality. The Bluetooth stack implemented on the Shimmer devices are 2.1 compliant, which allowed us to take advantage of the increased security provided over previous versions of the protocol.

Pairing two devices is the first step that must be taken before they can communicate with Bluetooth. Version 2.1 of the protocol introduced Secure Simple Pairing (SSP) [47]. SSP uses a Diffie-Hellman key exchange to negotiate an encryption key between the two devices that are being paired. Previous versions of Bluetooth required the user to enter a PIN of varying length on both devices. That PIN was then used as input to an algorithm which generated an encryption key. With SSP, the PIN displayed is only for the user to confirm the identity of the devices being paired, it is not used as part of the key generation.

There are four major association models that can be used to perform the pairing, each requiring varying levels of human interaction. As the shimmer devices have no display, and cannot accept input, the BAN relies on the Just Works association model, which requires no pairing confirmation from the user. This means that this BAN is vulnerable to passive eavesdropping and man-in-the-middle attacks during the pairing process, where other association models can mitigate those threats.

Bluetooth 2.1 requires Security Mode 4, which causes all communications to be encrypted. The only exception is the Discovery service, which is responsible for finding other active Bluetooth devices that can be paired. Previous versions of Bluetooth supported only up to Security Mode 3, which did not require all types of communication to be encrypted. Bluetooth 2.1 will fall back to a lower security mode

only when pairing with a legacy device. After the devices are paired, or associated, the encryption key created is known as the link key, and will be used to encrypt all further communication between the two devices.

Authentication is provided by a challenge-response scheme, where either member of a Bluetooth pair can challenge the identity of the other member. The challenger is referred to as a verifier, and the device being challenged is known as the claimant. The claimant proves its identity to the verifier by performing a variation of the SAFER+ algorithm known as the E1 algorithm to generate a response to the verifier using the claimant's Bluetooth address and the link key. The verifier performs the same calculation, and compares the claimant's response with the expected result. If they are different, the claimant fails verification, and no further communication from the claimant to the verifier will be accepted until the claimant can complete verification. As the link key is critical for performing authentication, it must be kept private in order to maintain a valid authentication routine. Unfortunately, as this BAN uses the Just Works association model, it is possible for a third party to obtain the link key by using a man-in-the-middle attack, and as such, the BAN cannot rely on this Bluetooth authentication method.

To provide confidentiality, Bluetooth 2.1 requires that all transmitted data payloads be encrypted. This is done with a stream cipher seeded by the link key, a pseudo-random number, and part of the response to an authentication challenge. However, as this BAN cannot guarantee the privacy of the link key due to the use of the Just Works association model, the BAN cannot rely on the encryption provided by Bluetooth. If a third party was able to obtain the link key through a man-in-the-middle attack, they could potentially decrypt all communications between the two hosts in the pairing. Furthermore, the length of the encryption key can vary from 8 to 128 bits depending on the manufacturer of the Bluetooth adapter. We were unable to determine the length of the key required either by the Shimmer devices or the base station. Finally, the encryption algorithm used, E0, which is again a variation of the SAFER+ algorithm, has not been FIPS approved. For these reasons we decided that encryption would have to be implemented at the application layer in order to maintain reasonable confidentiality. The data will still be further encrypted by Bluetooth as required by the 2.1 protocol.

Bluetooth 2.1 does not provide any methods for verifying data integrity, or non-repudiation. Without integrity verification, it is possible for an attacker to tamper with data being sent between devices. Although the attacker may not be able to decrypt the data being sent, sending false data in this BAN could cause the user to make incorrect medical decisions based on the false data [48].

C.2 Bluetooth Reliability

Bluetooth was selected as the method for wireless communication, so further research was required into what features were provided for in the specification. Among the features that were researched was the reliability of the communications sent over Bluetooth. Currently, reliability is not an issue, because the motes are sending data to the base station so quickly that a dropped packet here or there will not make a difference. However, not all of the future communications will be a stream of data. Some will be just a one-time message which could cause issues if it was lost.

Fortunately, Bluetooth includes a variety of features to make sure that the message you sent is received correctly. In order to help with interference, Bluetooth employs Adaptive Frequency Hopping (AFH). AFH allows the communication to move to the least congested 1 MHz channel in the 2402 MHz to 2480 MHz range. Of course, there is going to be some level of interference on every channel, so Bluetooth also includes error correction, and by extension detection. Specifically, Forward Error Correction (FEC) is used. FEC has been in use for decades, and as the name implies, gives the receiver of a transmission the ability to fix some errors without the need for retransmission. In addition, Bluetooth also uses ARQ retransmission. Every packet that is sent has an ACK/NAK bit which makes it possible to determine if a packet needs to be resent. Even if a packet was completely lost, you would still know, and would be able to resend the data.

C.3 Bluetooth Scalability

The scalability of Bluetooth networks, although not the main focus of the project, was considered and researched. While we were not concerned with developing or implementing a scaled Bluetooth network for the project, the ability for such networks to be formed was still relevant and of interest to the project. In the future if the size of the body area network needed or wanted to be increase, we wanted to ascertain if doing so would be possible. The Bluetooth protocol itself was first looked at to establish if it specified a scaled Bluetooth network or facilitated the development of one.

In the Bluetooth protocol the smallest network of devices is called a piconet. A piconet is an ad-hoc single-hop computer network that consists of a total of eight devices. Within a piconet one active device is the master and up to seven other active devices are slaves. Slave devices only communicate with the master device and do not as result directly communicate with other slaves. While the Bluetooth specification does not define or describe how piconets should be scaled, it does not prevent the formation of larger Bluetooth networks. In addition, the Bluetooth protocol does provide features that help facilitate the creation of bigger Bluetooth networks. The idea behind scaling Bluetooth networks is to connect

piconets together to form what is called a scatternet. The formation of scatternets is possible due to a number of properties of the Bluetooth protocol. A Bluetooth device is capable of being a master of one piconet but a slave in multiple piconets at a time. This enables connections between piconets to be made by utilizing devices that are members of multiple piconets. In addition, up to 255 slave devices can be set in what is called a parked state within a piconet. When in a parked state the device is not active within the piconet but is not unpaired from the master of the piconet. This enables devices to overcome the protocol limitation that they may only communicate in one piconet at a time. A device can switch between a parked and active state in multiple piconets in order to communicate within both.

After examining the Bluetooth protocol, research into the scalability of Bluetooth networks was carried out to see if any research or work existed on the subject. Research found focused on proposed scatternet formation, routing, and management protocols, associated algorithms, and network simulations utilizing different network topologies. For example, Bluetooth scatternets with star topologies called "BlueStars" are described in Petrioli et al.'s paper "Configuring BlueStars: Multi-hop Scatternet Formation for Bluetooth Networks" [49] Two other papers, by Lin et al. [50] and Záruba et al. [51] detailed protocols and experimentation for creating scatternets with ring and tree topologies respectively. Other research focused on subjects such as: specific formation algorithms for usage in scatternets [52], mesh scatternets [53], and the general suitability of Bluetooth for large-scale networks [54]. Through this research it was determined that while the creation and management of Bluetooth scatternets is certainly possible, and significant efforts in this area has been carried out, that currently there is no definitive protocol or framework that exists for doing so.

Appendix D: BAN Security

Our team performed a security analysis to identify potential security risks in a BAN. We also researched tools that may be used to mitigate some of those risks. Due to time constraints, we were unable to deploy any of these countermeasures in our BAN implementation.

D.1 Threat Model

We created a preliminary threat model, used to identify possible threats to the system and the technologies. An effort was made not to restrict analysis to just those threats that conceivably could be exploited by an adversary but to any and all potential threats to the system. At the same time, because of

the unpredictable nature of a potential attacker, we attempted to avoid predicting or assuming an adversary's behavior. The threat modeling consisted of the following stages:

1. Define, identify, and document system assets.

The first stage centered on establishing what resources have value within the context of the system and which therefore would need to be protected from threats and attacks. This step enabled us to accrue a concrete set of aspects within the system that needed to be analyzed further in terms of security. The following is the list of assets pinpointed within the system:

- Mote Local Data (SD Card Storage)
- Mote System Resources
- Mote Bluetooth Transmissions (Packet Data)
- Base Station Local Data (Internal / SD Card External Storage)
- Base Station System Resources
- Base Station Transmissions (Packet Data)

2. Pinpoint who and what is trusted within the system.

After documenting the assets, we then moved on to determining what entities within the body area network could or could not be trusted. This was done in order to help decide what each entity in the system should and should not be able to do or have access to, as well as how interactions between different entities would need to be handled. It was concluded that the only entity within the BAN that would be trusted was the base station, called the "central point of trust". Within the context of the network, the base station is the master device of the single Bluetooth piconet and interfaces with each slave mote. No other entities within the system are trusted.

3. Determine the relevant security goals.

This stage involved establishing the important properties of the system that must be addressed. These security goals, confidentiality, integrity, and availability, are relevant to almost any computer-related system and the body area network is no exception. In addition to those three goals, it was also determined that authentication and authorization would be integral properties of the system and should be addressed. Below each term is defined within the context of an asset or entity in a system.

➔ Confidentiality: For an asset to be confidential means that the asset is only accessed by authorized entities. An entity that should not be able to access an asset therefore should not be able to do so.

➔ Integrity: An asset can only be altered by authorized entities.

➔ Availability: For an asset to be available means that the asset can be accessed by authorized entities. An asset should not be prevented from being accessed by an entity that should be able to do so.

➔ Authentication: For an entity to be authenticated means that it has been determined that the entity is who or what it claims to be.

➔ Authorization: For an entity to be authorized means that the entity has been granted or denied access to an asset within the system.

After determining the security goals outlined above, the group analyzed different aspects of the body area network system and incorporated technologies in relation to each security goal. In doing so we pinpointed specific vulnerabilities associated with each security goal for each aspect of the system. All the information gathered was compiled into the following two tables. The first table details security goals, and whether particular aspects of our BAN met those goals. The second table describes security threats and potential mitigation techniques.

Security Goal ⇨ ⇩ Aspect	Confidentiality	Integrity	Availability	Authentication	Authorization
Shimmer Mote SD Card	None; only accessible by mote itself, unless a firmware implements a way to access the data via remote connection	None; only accessible by mote itself, unless a firmware implements a way to access the data via remote connection	An SD card has a limited storage capacity.	n/a	n/a
Shimmer Mote System Resources	n/a	n/a	The embedded system has a limited amount of CPU processing power and available memory.	n/a	n/a
Android Internal Storage	None; only accessible by application storing the data	None; only accessible by application storing the data	None; only accessible by application storing the data	n/a	n/a

Android External Storage	Android External storage is globally readable and writable	Android External storage is globally readable and writable	Android External storage is globally readable and writable	n/a	n/a
Android System Resources	n/a	n/a	The BAN application could potentially interfere with the other operations of the device	n/a	n/a
Bluetooth Discoverable Mode	The device and its BT address are exposed to any Bluetooth device in range while in discoverable mode	The contents of the advertisements and/or scanning requests could be modified mid-transmission	An discoverable device will continually respond to scans from other devices	A device can spoof its BT address and pretend to be another device in the area	n/a
Bluetooth Transmissions	Encryption enforced is only as good as the highest encryption supported by both devices; supported key length is between 8 and 128-bits.	Packet data can be modified mid-transmission; if keys are known, checksums can be modified; if keys are unknown, Bluetooth's CRCs and checksum headers may not catch modification	Bluetooth transmissions could be jammed or disrupted using RF interference	The identity of the device on the other end is guaranteed to be the device you paired with; see Pairing vulnerabilities	n/a

Table 8: BAN Security Goals Analysis Cross Table

Vulnerability	Threats	Countermeasures
Android devices' external storage is globally accessible.	1. Data stored by an application on external storage can be accessed or tampered with by any application on the device, or any entity with control over the device.	1. Do not use Android external storage to store sensitive data. 2. Store all data using Android internal storage. 3. Encrypt any data stored in internal storage to provide additional protection.
External storage media is physically removable from the device.	1. Any information stored on the external storage can be stolen or accessed.	See countermeasures above.
Bluetooth Secure Simple Pairing (SSP) does not guarantee authentication.	1. A device in a pairing procedure can falsify its identity or capabilities.	1. Prevent the ability to pair devices unless in a safe / private environment. 2. Enforce usage of more secure Bluetooth association model(s) if feasible.
Default Shimmer Mote Bluetooth Radio physical range extends far beyond the needs of a BAN. (Class 2, 10 meters)	1. Communications within the effective transmission range of devices in the BAN can be intercepted/sniffed/overheard.	1. Limit the Bluetooth transmission range of BAN devices. 2. Ensure communications are encrypted to thwart passive eavesdropping.
Bluetooth Discoverability Modes other than silent do not provide or guarantee privacy.	1. A device in discoverable mode can be seen and connected to by any other device in range. 2. A device in non-discoverable mode can still be found.	1. Avoid using discoverable mode unless pairing devices.
2.4GHz ISM band is a shared radio band.	1. Other devices operating in the shared 2.4GHz band can cause electromagnetic interference.	1. Bluetooth protocol utilizes a frequency-hopping spread spectrum mechanism to help mitigate RF interference.

Table 9: Vulnerabilities, Threats, and Countermeasures

D.2 Embedded System Cryptography Libraries

We briefly looked into initial cryptography system options for the motes in our BAN. The following were the most promising options. However, further research would be required to make a fully informed decision.

D.2.1 TinySec

The most well-known cryptographic library available for the TinyOS platform is TinySec [55]. It was introduced in 2003 and described as "the first fully-implemented link layer security architecture for wireless sensor networks" [56]. TinySec claims to address the common restrictions of sensors networks, specifically the severe resource constraints imposed by sensor hardware. The creators of TinySec argue that existing network security protocols such as SSL/TLS, SSH, and IPSec are far too heavy-weight for utilization in sensor networks. As a result, TinySec serves as the protocol to fill the niche requirements of wireless sensor networks. In addition, the authors of TinySec boast the library's portability across hardware and wireless radio platforms, making it an ideal solution for a wide variety of applications.

The TinySec library offers two forms of link layer security: authenticated encryption and authentication only. In the case of authenticated encryption, "TinySec encrypts the data payload and authenticates the packet with a MAC", whereas with authentication only, "TinySec authenticates the entire packet with a MAC, but the data payload is not encrypted" [56]. Both encryption and MAC mechanisms utilize a block cipher, specifically the cipher block chaining (CBC) mode of operation. By using block ciphers for both encryption and generating MACs, TinySec reduces space require for code. The TinySec implementation when initially finished required 728 bytes of RAM and 7146 bytes of program space, but was subsequently reduced to use just 256 bytes of RAM and 8152 bytes of ROM. Minimal RAM and ROM usage is important in sensor networks due to hardware restrictions. For encryption, the Skipjack block cipher is utilized by default while the RC5 block cipher can be used as well. For message integrity, a CBC-MAC implementation is used. An in-depth analysis and evaluation of the library's performance, such as cipher performance, throughput, latency benchmarks, energy consumption, and ease of use was carried out and detailed by the creators of TinySec. Testing was carried out on the Mica2 sensor platform, which has an 8 MHz processor, 128kB of instruction memory, 4kB of RAM, and 512kB of flash memory. The results of their evaluation showed promising results of minimal increases (~ 10% increase) in energy consumption as well as latency / transmit times over the normal operation metrics of the sensors.

The TinySec security library may fill the security needs of a wireless sensor network such as a body area network. That being said, as it stands it is unlikely that the TinySec library would be compatible with this project's protocol and firmware out of the box. Potential issues include:

1. TinySec was implemented exclusively for TinyOS version 1.x and may not be compatible with the body area network protocol which is implemented on TinyOS 2.x.

    1. To the best of our knowledge, no port of TinySec to TinyOS 2.x exists.
2. TinySec utilizes a modified version of the default TinyOS 1.1.2 radio stack and by default does not support common sensor network protocol/wireless radios in usage such as ZigBee (802.15.3) or Bluetooth.
    1. The project's protocol currently utilizes the Bluetooth protocol and it is likely that significant modifications would be required in order to get TinySec working with the group's firmware and protocol.
    2. A modified version of TinySec with ZigBee support is available [57].
3. TinySec was evaluated on sensors with different computational resources than those currently utilized in the BAN and as a result may not perform the same on the Shimmer hardware or other hardware in the future.
4. Issues with TinySec have been identified and documented, such as potential replay attack vulnerabilities [58], pitfalls of the usage of a single network-wide key system, and poor energy consumption [59].

The issues highlighted above are areas that would benefit from more in-depth research, testing, and analysis of the TinySec library.

D.2.2 MiniSec

Another secure sensor network communication architecture, MiniSec [60], aims to resolve issues with existing architectures and protocols, specifically TinySec and ZigBee. Like TinySec, MiniSec is implemented for the TinyOS platform and written in nesC. All cryptographic primitives and implementations utilized were ported to nesC for the architecture. The creators of MiniSec argue that TinySec and ZigBee fall short and "are unable to achieve low energy consumption while simultaneously providing the three important properties of secure communication: secrecy, authentication, and message replay protection" [59]. Within the context of wireless sensor networks, low energy consumption is very important as battery life on sensors is a limited resource. While TinySec is able to achieve low energy and memory usage as described in the previous section, the authors of MiniSec claim that it does so by reducing security provided. Ideally, a protocol that achieves both low resource usage and a high level of security would be best suited for usage in a BAN and as for that reason MiniSec is of interest to the project.

To achieve low energy consumption and better security, MiniSec offers two modes of operation: single-source unicast communication, and multi-source broadcast communication which are each optimized separately. Like TinySec, MiniSec utilizes a block cipher mode of operation, specifically an Offset CodeBook (OCB) block cipher to simultaneously achieve secrecy and authenticity of data. In contrast, TinySec requires two steps to achieve secrecy (CBC-encryption) and authenticity (CBC-MAC). As a result, TinySec requires more computation and energy usage. MiniSec does require more memory usage that TinySec, however the authors argue that "the design tradeoffs in MiniSec make it well-suited for current state-of-the-art sensor devices" [59]. The implementation of MiniSec uses 874 bytes of RAM and 16KB of code memory. In addition to secrecy and authenticity, MiniSec advertises weak freshness of messages, as well as replay protection mechanisms in both unicast and broadcast modes.

MiniSec was designed and developed for the popular Moteiv Telos mote platform [61]. A huge benefit of this fact is that the hardware of the Telos motes MiniSec used and the Shimmer motes currently utilized by the project are nearly identical. Like the Shimmer motes, the Telos motes feature an 8MHz TI MSP430 microcontroller, 48kB flash memory, and 10kB RAM. As a result, MiniSec performance evaluations and metrics can be easily used to give a good idea of how MiniSec would operate on Shimmer motes or similar hardware.

MiniSec's creators performed an evaluation of the architecture on Telos motes which focused on both communication and computational costs associated with transmitting and processing packets. MiniSec achieves a reduction in the security overhead of packets and requires only 3 bytes whereas TinySec requires 5 bytes. Any decrease in total packet size is beneficial in a sensor network environment where hardware resources are limited. It requires more energy and computational resources to process and communicate larger packets, and as such MiniSec ensures an overall decrease in those costs over TinySec. When comparing the total increase in communication overhead over the default TinyOS network stack, MiniSec only incurs an 8.3% increase while TinySec incurs a 13.9% increase [59]. Less overhead translates to less energy and computational resource consumption and thus increase the resources available to a mote.

Compared to TinySec, MiniSec boasts better energy consumption as well as better security provided at the cost of minimal increase in memory usage. MiniSec provides solutions to issues with both the TinySec and ZigBee protocols and as a result makes it an ideal potential candidate for usage in a body area network environment. As well, the MiniSec codebase is publicly available should it need to be ported or adapted to another platform. However, further work needs to be carried out in order to determine if MiniSec is a completely viable solution for application level security for a sensor network. Specifically, some areas of interest remain unanswered by this preliminary research:

1. MiniSec would likely need to be adapted for usage with this project's body area network protocol and it is not known how difficult that process would be.
2. MiniSec does not address the issues of key establishment or management, which are necessary for secure network communication. Research into compatible and secure keying solutions would be required if MiniSec were to be used.
3. In a multi-hop sensor network environment, MiniSec does not address the issue of secure routing. Research into secure routing solutions would also be needed should MiniSec be utilized in a multi-hop network.

D.2.3 TinyECC

TinyECC is library designed for configurable usage of elliptic curve cryptography (ECC) operations in a wireless sensor network [62]. The goal of TinyECC, as described by the creators is to "provide a ready-to-use, publicly available software package for ECC-based PKC operations that can be flexibly configured and integrated into sensor network applications" [63]. TinyECC advertises an array of optimization options in the library that enable tweaking of performance based on the needs of the developer. Generally speaking, public-key cryptography has not been heavily utilized in sensor network protocols or applications due to the large overhead associated with it. TinyECC serves to provide a solution to that problem specifically tailored for usage in wireless sensor networks.

Like TinySec and MiniSec, TinyECC was designed and developed for the TinyOS platform. It was written entirely in nesC with the exception of some in-line assembly code sections for sensor platform specific optimizations. Such sensor platforms include MicaZ [64], TelosB [65], Tmote Sky [66], and Imote2 [67]. TinyECC was tested on all the aforementioned sensor platforms. The library implements nearly every known optimization available for ECC operations and each optimization is controllable by a software switch. These optimization switches increase the configurability and flexibility of the library and enable the developer to make optimization decisions based on their specific application's requirements.

TinyECC offers three well-known ECC methods that have been proven to be secure: the Elliptic Curve Diffie-Hellman (ECDH) key agreement, the Elliptic Curve Digital Signature Algorithm (ECDSA), and the Elliptic Curve Integrated Encryption Scheme (ECIES). The library supports 128-bit, 160-bit, and 190-bit elliptic curve domain parameters. An example given by the authors is that 160-bit ECC provides the same security as 1024-bit RSA. The benefit of these variants of existing ECC schemes is that they

enable smaller key sizes while still providing a similar level of security to what they were based on (i.e. DH and DSA).

An evaluation of TinyECC was designed and executed by the library developers on the four sensor platforms mentioned earlier. In doing so, information such as performance, resource consumption, and optimization impacts were accrued. Their experiments consisted of measuring performance (execution time) and resource consumption (ROM, RAM, energy) metrics in two distinct cases for each optimization switch provided by the library. In the first case, with all other switches disabled, metrics were recorded with each switch enabled and disabled. In the second case, with all other switches enabled, the same metrics were recorded with each switch on and off. Detailed results such as individual optimization impacts, as well as the most computationally efficient, and most storage efficient configurations are provided.

As it stands, it is not known whether TinyECC would be compatible or feasible for usage with the body area network protocol this project is developing. More research into TinyECC and how it performs on hardware used by Shimmer motes is necessary to determine if TinyECC is a viable solution for key exchange or encryption.

Three major TinyOS-based and wireless sensor network oriented security protocols have been identified and briefly described. Other research on the subject such as the SPINS protocol [68] as well as some other TinyOS/nesC cryptographic libraries [69] [70] were looked at in addition. Areas of future work and research have been outlined in regards to each major protocol highlighted. While the focus of this project is not on implementing security via cryptography, we believe that future projects may focus on that problem. As such, this section serves to provide as a starting point for continued and more in-depth research of existing protocols and solutions. Within the context of a body area network security is an unavoidable necessity and the libraries outlined may serve as solutions or starting points to achieve security. For example, it may be possible to utilize TinyECC in conjunction with another protocol such as TinySec or MiniSec in order to implement both secure key agreement as well as encryption and authentication within a wireless sensor network. Ultimately, the greatest limitations will likely not be incurred by the protocol itself, but by the hardware and resource limitations of the sensor platforms used. As sensor platform hardware technology continues to improve, the limitations of motes will likely decrease and enable more varied security solutions.