Real-Time Traffic Monitoring
Utilizing Bluetooth Device Discovery

by
Rory Jacob
Marc Friesen
Tyler Mailey
Paul Grestoni

Final report submitted in partial satisfaction of the requirements for the degree of

Bachelor of Science
in
Computer Engineering
and
Electrical Engineering
in the
Faculty of Engineering
of the
University of Manitoba

Faculty Supervisor:
Dr. Robert McLeod

Industry Supervisor:
Doug Hurl

Copyright © 2013 by Rory Jacob, Marc Friesen, Tyler Mailey, Paul Grestoni
Abstract

The ubiquitous nature of Bluetooth equipped devices has made it opportunistic to scavenge information that can be repurposed for applications other than initially intended. One such opportunity is in vehicular traffic monitoring, whereby sampling of Bluetooth radios serve as proxies for vehicles and consequently density and flow. A wireless sensor network of probes was designed and implemented with the objective to collect Bluetooth device information for this purpose. To facilitate easy setup and a long battery life a solar powered probe was designed. Data from each probe is communicated to a master node through XBee communication, where it is stored on a Secure Digital (SD) memory card before being transmitted to a central server every five minutes over a Global System for Mobile Communications (GSM) cellular network. The server parses the data received and stores it in a database. Consumer and corporate websites may then access this database to display archived data or current data in real-time to various users.

The results achieved from testing conformed with expected traffic trends. Distinguishable peaks in the traffic data were observed, such as the rush hour morning commute, as well as a similar trend in the opposite direction during the evening rush hour times. This data confirmed the ability of the system to provide representative traffic data in a real-time scenario.
# Table of Contents

List of Figures ........................................................................................................... v
List of Tables ............................................................................................................. vi
List of Definitions and Acronyms ............................................................................. vii
Acknowledgements ................................................................................................. ix
Contributions ........................................................................................................... x

**Chapter 1 - Introduction** .................................................................................. 1

1.1 Motivation ........................................................................................................... 1
1.2 Proposal ................................................................................................................ 1
1.3 Problem Specifications ....................................................................................... 1

**Chapter 2 - Probe Design** ............................................................................... 3

2.1 Overview ............................................................................................................. 3
2.2 Hardware ............................................................................................................. 3
   2.2.1 Bluetooth Radio Selection .......................................................................... 4
   2.2.2 XBee Radio Selection .................................................................................. 4
   2.2.3 Radio Module Interfacing .......................................................................... 6
2.3 Power Management ......................................................................................... 6
2.4 Software ........................................................................................................... 7
   2.4.1 Packet Structure ......................................................................................... 8
   2.4.2 Initialization ............................................................................................... 9
   2.4.3 Synchronization ......................................................................................... 9
   2.4.4 Data Collection and Transmission ............................................................. 10

**Chapter 3 - Master Node Design** ................................................................. 12

3.1 Overview ........................................................................................................... 12
3.2 Platform ........................................................................................................... 12
3.3 Software Flow ................................................................................................. 13
3.3 Master Node - Server Communication ........................................................... 16
3.4 Memory Limitations ....................................................................................... 17

**Chapter 4 - Power Systems** ......................................................................... 19

4.1 Overview ........................................................................................................... 19
4.2 Initial Design ................................................................................................. 19
4.3 Intermediate Design ..................................................................................... 20
4.4 Final Design ................................................................................................... 21
Appendix B: ATmega328 Schematic ................................................................. 77
Appendix C: Database .................................................................................. 78
Appendix D: Budget ..................................................................................... 79
Appendix E: Accepted Paper for the CCECE 2013 .................................. 81
Vita .................................................................................................................. 88
List of Figures

Figure 2-1: Traffic Monitoring Network [5] .................................................................................. 3
Figure 2-2: Device Detection Distribution .................................................................................. 5
Figure 2-3: Custom Bluetooth Interface PCB ............................................................................. 6
Figure 2-4: Probe Software Flowchart ....................................................................................... 7
Figure 2-5: Packet Structure ....................................................................................................... 8
Figure 2-6: Device Scan Flowchart ............................................................................................ 10
Figure 2-7: Device Frame Structure ........................................................................................... 11
Figure 2-8: Marker Byte Example ............................................................................................... 11
Figure 3-1: GBoard .................................................................................................................... 13
Figure 3-2: Master Node Flowchart ........................................................................................... 14
Figure 3-3: Master Node Synchronization Flowchart ................................................................. 15
Figure 3-4: Master Node Data Flowchart .................................................................................. 16
Figure 3-5: SRAM Memory Map on ATmega88PA, ATmega168PA and ATmega328 [9] ........... 17
Figure 3-6: Program Memory Map for ATmega328 [9] ............................................................... 18
Figure 4-1: 9 V Rechargeable Design ....................................................................................... 20
Figure 4-2: Intermediate Design ............................................................................................... 20
Figure 4-3: Final Design Prototype Components ....................................................................... 22
Figure 4-4: Boost Converter Schematic [13] ............................................................................ 22
Figure 4-5: Step-up Hardware Prototype .................................................................................. 23
Figure 4-6: Multisim Model of the Step-up Converter ............................................................... 24
Figure 4-7: Step-up Converter ................................................................................................. 25
Figure 5-1: Received Data from Master Node Flowchart ........................................................... 28
Figure 5-2: Map with Probe Markers ....................................................................................... 30
Figure 5-3: Probe Options Panel ............................................................................................... 31
Figure 5-4: Route Information Panel ........................................................................................ 32
Figure 5-5: Route Setup Displayed on Map ............................................................................. 32
Figure 5-6: Line Chart of Traffic Count for a Probe vs. Time .................................................... 33
Figure 5-7: Traffic Status Display ............................................................................................. 34
Figure 6-1: a) Master Node b) Solar Powered Probe c) Basic Probe ........................................ 36
Figure 8-1: Traffic Counts at Probe 0 ....................................................................................... 38
Figure 8-2: Average Velocity vs. Time of Day ....................................................................... 39
Figure B-0-1: ATmega328 Architecture [13] ......................................................................... 77
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 5-1: traffic_data</td>
<td>Structure</td>
<td>27</td>
</tr>
<tr>
<td>Table 5-2: probe_pairs</td>
<td>Structure</td>
<td>28</td>
</tr>
<tr>
<td>Table C-1: Tables in Database</td>
<td></td>
<td>78</td>
</tr>
<tr>
<td>Table D-1: Project Budget</td>
<td></td>
<td>79</td>
</tr>
<tr>
<td>Acronym</td>
<td>Definition</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td></td>
</tr>
<tr>
<td>API</td>
<td>Application programming interface. A protocol that allows software components to communicate with one another</td>
<td></td>
</tr>
<tr>
<td>Bootloader</td>
<td>The program in non-volatile memory that loads the operating system</td>
<td></td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
<td></td>
</tr>
<tr>
<td>DC</td>
<td>Direct current</td>
<td></td>
</tr>
<tr>
<td>EOP</td>
<td>End Of Packet. Character representing the last of every transmission</td>
<td></td>
</tr>
<tr>
<td>GSM</td>
<td>Global System for Mobile. A cellular communication technology</td>
<td></td>
</tr>
<tr>
<td>GPRS</td>
<td>General Packet Radio Service. Packet oriented mobile data service</td>
<td></td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
<td></td>
</tr>
<tr>
<td>I/O</td>
<td>Input and Output</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>Identifier</td>
<td></td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
<td></td>
</tr>
<tr>
<td>Javascript</td>
<td>Client side computer programing language</td>
<td></td>
</tr>
<tr>
<td>LED</td>
<td>Light Emitting Diode</td>
<td></td>
</tr>
<tr>
<td>MAC</td>
<td>Media Access Control</td>
<td></td>
</tr>
<tr>
<td>mAh</td>
<td>The amount of current a power source can sustain for an hour</td>
<td></td>
</tr>
<tr>
<td>Multisim</td>
<td>Software used to create and simulate electrical circuitry</td>
<td></td>
</tr>
<tr>
<td>MySQL</td>
<td>Open source database management system</td>
<td></td>
</tr>
<tr>
<td>NiMH</td>
<td>Nickel-metal Hydrate</td>
<td></td>
</tr>
<tr>
<td>nMOS</td>
<td>N-channel MOSFET</td>
<td></td>
</tr>
<tr>
<td>PCB</td>
<td>Printed Circuit Board</td>
<td></td>
</tr>
</tbody>
</table>
PHP - Open source server side scripting language

Real-Time - For this project, real-time is considered to be updating the data on the server, from the master node, every five minutes

RTT - Round Trip Time

SD Card - Secure Digital Card. A non-volatile memory card used by various embedded devices

Shield - Hardware interface board for Arduino

SRAM - Static Random Access Memory

Stack - Memory architecture that functions in last-in first-out manner

String - A sequence of characters

Traffic Status - The traffic count, average delay, and average velocity through a given roadway

XBee - Radio module made by Digi International that uses the 802.15.4 protocol
Acknowledgements

There are several people we would like to acknowledge for providing us with the advice and equipment we needed to complete our project. First, we would like to thank our project advisor, Dr. Robert McLeod, who allowed us the use of his lab to work on our project as well as all the tools and components that were donated to us. Special thanks to Doug Hurl, Luis Escobar and the Public Works Department of the City of Winnipeg for their financial support. We would also like to thank Sinisa Janjic and Ken Biegun from the tech. shop for providing us with all the parts we needed, Zoran Trajkoski for constructing all of our PCBs, and to Cory Smit for all the tools and accessories he provided us.
Contributions

This design project aims to create a traffic monitoring system that can accurately acquire traffic data including velocity of vehicles and traffic congestion in a real-time fashion. While there are other systems currently available to monitor traffic, they are not real-time and are solely used for providing traffic counts. Our system will take advantage of the ubiquitous nature of Bluetooth devices to provide real-time traffic status. This project can be divided into several parts, which were completed by one or more group members.

Rory Jacob
- Design and implementation of the master node
- Design of communication protocol between master node and server
- Design of Bluetooth interface PCBs
- Assisted in server data collection

Marc Friesen
- Design and implementation of the probes
- Design of communication protocol between master node and probe
- Assisted in power system design and implementation
- Assisted in website design and implementation
- Assisted in casing construction
- Assembly of custom Bluetooth interfaces

Paul Grestoni
- Design and implementation of power system
- Constructed casings
- Assembly of custom Bluetooth interfaces

Tyler Mailey
- Design and implementation of server scripts
- Design and implementation of website

Group
- Testing and analysis of system
Chapter 1 - Introduction

1.1 Motivation

There are over 21 million drivers in Canada with a comparable number of cars [1, 2]. Every year the population is growing, and with this brings an increase in both drivers and cars on the road. With all of this comes an increase in congestion. At current levels, congestion costs Canadians over 15 billion dollars every year [3]. With the current state of technology, people want to be able to get to their destination without waiting in traffic for hours. As well, cities need to know which intersections are consistently problematic, and the congestion of nearby roadways so that they can effectively plan strategies to reduce congestion. Currently, there are a number of alternatives which provide metrics to cities or the public on current intersection state. However, these alternatives are either expensive or non-real-time. A project implemented by the City of Calgary used Bluetooth to detect and display average travel time on display boards located on the roadway; the cost of this installation was over $400,000 [4]. Smaller cities like Winnipeg have access to road strips, cameras, and magnetic puck sensors in the road. While these are able to provide traffic counts, they do not allow for the average velocity and travel time to be calculated in real-time and most of them are difficult or impossible to relocate to a different location.

1.2 Proposal

What we are proposing is a method to easily monitor the traffic status of any roadway or intersection, and display these results in real-time to municipal services and the public. Potentials for this application would be to create a vast probe network to monitor traffic flow in cities, or be integrated into a web based mapping site like Google Maps to help determine the fastest route between a source and destination.

1.3 Problem Specifications

The solution we are proposing has the following requirements. It should be inexpensive, easy to setup, provide real-time updates, and be able to reloacate and deploy quickly. Because of these requirements, the probes should be battery powered and solar rechargeable, lasting at least 24 hours. As well, since a probe network must be able to be relocated quickly, the size and weight of each probe should be kept under 1000 cm$^3$ and 5 kg respectively.
In order for the data collected to be useful, it must be presented to a user via a front end interface. Two interfaces will be required, a corporate website and a public website. The corporate website must be able to allow setup of probe networks and perform extended analysis on data collected. The public website must show the current traffic status to a user in real-time.
Chapter 2 - Probe Design

2.1 Overview

The objective of our probes is to detect and collect openly available device information when a vehicle with a Bluetooth device passes by. When a Bluetooth device is discovered, the time of discovery is recorded, and this information is sent to the master node using an XBee radio. Our network has several probes placed at known locations; using the distance between probes and the time of discovery, traffic status can be determined. An example network is shown in Figure 2-1.

![Figure 2-1: Traffic Monitoring Network [5]](image)

2.2 Hardware

For our probe design, we used the Arduino development platform. Arduinos are a widely used development platform which provided us with a large resource pool for troubleshooting and example code. The Arduino also provided us with a simple interface between the radio modules and the microcontroller as well as libraries to communicate with the radio modules, allowing us
to develop the code more quickly. To aid with testing and debugging, a multi-coloured LED was attached to the Arduino’s digital I/O pins which can be varied programmatically to produce different colours.

### 2.2.1 Bluetooth Radio Selection

To detect Bluetooth devices, a Bluetooth radio must enter into device discovery mode; this period is called the inquiry period. During the inquiry period, the radio is able to detect devices that have been set to auto-discoverable mode, which means that they are continually broadcasting information about the device. The information that is broadcast includes the MAC address of the device which is used for establishing a connection between two devices. When this MAC address is discovered, the Bluetooth radio sends this information to the microcontroller for processing. Initially, we had planned on using a Bluetooth module that was limited to only eight unique devices per inquiry. We did not believe this would be adequate during dense traffic conditions [6]; therefore we selected the Pro version of the Bluetooth module as it is able to discover 250 unique devices per inquiry period [7].

### 2.2.2 XBee Radio Selection

When detecting devices with the Bluetooth radio, devices can be detected from as far as 50 meters away. Because of this, a device can be detected 50 meters before the location of one probe, and 50 meters passed the next; this adds considerable error when probes are placed close together [7]. This error would be reduced by placing the probes as far from the master node as possible. The range of the standard Series 1 XBee module is about 90 meters, depending on the environment that it is in [8]. This would allow us to place two probes 180 meters apart, with the master node centered between them. The maximum percent error with this arrangement would then be

\[
\text{Error} = \frac{100 \text{ meters}}{180 \text{ meters}} \times 100\% = 56\%
\]  

(3.1)

The Series 1 XBee-Pro module allows us to place the probes up to 1600 meters away from the master, which would reduce our maximum error to

\[
\text{Error} = \frac{100 \text{ meters}}{3200 \text{ meters}} \times 100\% = 3.1\%
\]

(3.2)
However, even with the error present, it can be expected that the error associated with device detection will cancel out if successive probes have the same orientation. This is assuming a normal distribution of detection points with the mean at the probe’s location. If the distribution is not normal, all probes will still have the same distribution, and as long as they have the same orientation the mean of their distributions will have the same offset from the probes actual location. An example of normally distributed detection points is shown in Figure 2-2; with two probes placed at -400 m and +400 m relative to the master node, approximately equal numbers of devices are discovered before and after the probe, for each probe.

![Device Detection Distribution About Two Probes](image)

**Figure 2-2: Device Detection Distribution**

In addition to reducing the error for each device detected, the data collected is also more meaningful because there will be a larger difference in average time between probes during dense traffic states and sparse traffic states. These factors led to us choosing the XBee-Pro module.
2.2.3 Radio Module Interfacing

One of the problems we had with using two different radio modules on one Arduino platform was that they both used an interface board that used the hardware serial lines. Since there is only one set of hardware serial lines, using these interface boards as is for both of them was not an option. To circumvent this problem we designed a custom PCB, as shown in Figure 2-3, for the Bluetooth module to interface with the Arduino. This PCB connects the Bluetooth module to two digital I/O pins on the Arduino so that a software implementation of the serial communication could be used. The Arduino IDE already comes with a library to handle the software serial implementation, which allowed us to easily communicate with the Bluetooth module using the custom PCB.

![Custom Bluetooth Interface PCB](image)

Figure 2-3: Custom Bluetooth Interface PCB

2.3 Power Management

As with any wireless sensor network, managing power consumption is a primary concern. Because all of the probes are battery powered, we need to make sure we are using that power
efficiently. The XBee modules provide various alternatives to this end. With the XBee module configured for pin hibernation, we were able to programatically put the module in hibernation mode and wake it up by turning on or off one of the digital I/O pins. As the probes did not need to have the XBee modules active when there was no data to send, we put them in hibernation mode until there was data ready, at which point they were woken up. This reduced our average current consumption from approximately 160 mA to 100 mA.

2.4 Software

Each of the probes have the same software, with the exception of the probe identifier which is currently hardcoded. When a probe turns on or resets, the probe enters the initialization phase, followed by the synchronization phase, and finally the data collection and transmission phase. The probe will stay in the data collection and transmission phase continually with the exception that it re-synchronizes with the master once every 24 hours. The software flow is depicted in Figure 2-4 below.

![Probe Software Flowchart](image)

**Figure 2-4: Probe Software Flowchart**

During each phase, the LED is changed to a different colour to indicate which phase the probe is in. The LED is red when the probe is on, during initialization it is yellow, white during
synchronization, flashes blue when a device is detected, and finally green when the probe is sending the data the master node.

### 2.4.1 Packet Structure

In our system, the probe communicates with the master node using a packet structure that we developed. Each packet is between 3 and 92 bytes, excluding the first three of which are reserved. The reserved bytes are the control, length, and probe ID bytes, respectively. The packet structure is shown in Figure 2-5.

```
| Start-Of-Frame (5 bytes) | Control (1 byte) | Length (1 byte) | Probe ID (1 byte) | Payload (0 - 88 bytes) | EOP (1 byte) |
```

**Figure 2-5: Packet Structure**

Initially, the packet design did not include a start-of-frame. This was included to overcome a problem introduced when the XBee hibernation mode was added. When the XBee module wakes from hibernation mode, the receive buffer accumulates semi-random garbage data that is sent to the master node. The garbage data seemed to have consistent length, and the characters were often similar to previous series of garbage data. One possible source of this could be that the data in line is idle high. This means that on wake up the XBee module is likely charging the line capacitance until it reaches the high voltage of 5 V. Because the line capacitance is constant, the amount of time to reach the idle voltage would be constant as well, which could explain the consistent length of garbage data and the similar characters observed. The problem this creates at the master node is that it is unable to differentiate between real data and garbage data. By adding the start-of-frame the master node is able to differentiate between the two.

The control byte is used to indicate to the master what kind of packet it is receiving, an initialize synchronization packet, a synchronization packet, or a data packet. The length byte indicates how long the packet is, excluding the five bytes for the start-of-frame, and is used to make sure that all of the data was received correctly. The probe ID byte indicates from which probe the packet was sent. The packets are sent as character arrays that could be control characters, such as end-of-file, newline, and carriage return characters. To prevent control characters from affecting our packets, any byte that was less than 0x40 was marked, and 0x40 was added to it. Upon receipt of a packet, the master subtracts 0x40 from any byte that was marked as having been changed. Immediately
following the payload of any packet is a null character which is used to terminate character arrays in C.

2.4.2 Initialization

During the initialization phase, the Bluetooth module is reset and the probe waits for the appropriate response from the module. Next, the XBee module has to be configured with the parameters for our network. The XBee module is set to associate with a network coordinator, the correct network ID, and other parameters. The channel ID is not set on the probes because when the master node turns on, it scans the channels and selects the channel that has the lowest energy detected. The XBee module on the probe will then search all channels for the network coordinator, and set its channel ID accordingly.

2.4.3 Synchronization

When the probe records device information, it also adds its current timestamp. The timestamp on the probe is the amount of time in milliseconds that has passed since it was turned on or reset. Because each probe will have a different timestamp, there is no meaningful relation for the time elapsed between a device being detected at two different probes. In the synchronization phase, the master node determines an offset that relates each probes timestamp to its own timestamp. To start the synchronization process, the probe sends the master node an initialize synchronization packet. For the initialize packet, the control byte is set to 0x41, the length byte is set to 0x43, and the probe ID is the current probe sending the packet. After sending this packet, the probe will wait for an acknowledgement. If it does not receive an acknowledgement during the timeout period, it will resend the packet. It will try to initialize the synchronization process five times; if it does not synchronize with the master node, it is assumed that the master node’s XBee module is on a different channel. If this happens, the probe will reset and re-connect to the master node. When the probe successfully receives the acknowledgement, it sends five consecutive synchronization packets, each separated by an acknowledgement from the master node. The control byte for the synchronization packet is set to 0x42 and the payload of the packet contains the current timestamp. The master node uses the last timestamp received in the calculation for the probe offset which is explained further in section 3.3.
2.4.4 Data Collection and Transmission

During the data collection and transmission phase, the Bluetooth module is instructed to begin an inquiry. The inquiry period can be set to a maximum of 48 seconds. To avoid missing devices, the probes’ inquiry period is set to this maximum. The Arduino provided us with an SRAM constraint of 2 kB, which is where the real-time data is stored. Between the memory required for program variables and the memory required for real-time data, there was not enough memory to continually scan without offloading the real-time data. As the buffer size for the XBee module is 100 bytes, we stored the real-time data in 100 byte packets. If the packet fills during the inquiry period, the probe will send the packet to the master node, clear the packet, and then re-populate the packet with new data. The probe will stay in this phase continually until 24 hours has passed, at which time it will enter the synchronization phase to ensure that the offset is still accurate. After the probe re-synchronizes, it will again enter the data collection and transmission phase. The data collection and transmission phase is shown in Figure 2-6.

![Device Scan Flowchart]

Figure 2-6: Device Scan Flowchart

When a data packet is being generated, the control byte is set to 0x44. Each device frame is 8 bytes, starting with a marker byte, a 4 byte timestamp, and 3 bytes for the partial MAC address, shown below in Figure 2-7.
As indicated earlier, if any of these bytes or control characters are control characters, they can affect the data; specifically the null character. If any byte in the packet is the null character, only the data preceding it will be sent. As device information is packetized, each byte is evaluated to see if it is 0x00: the null character. If it is the null character, then it is changed to 0x40. When a byte is changed, the marker byte is logically OR’d with a bitmap that indicates the position of the byte being changed. For example, if the 6th byte in the device frame was changed, the 6th bit of the marker byte is set to 1. The marker byte is also self-marking, that is, if nothing was changed, the 0th bit in the marker byte is set to 1. An example of this process is shown below in Figure 2-8.
Chapter 3 - Master Node Design

3.1 Overview

The master node is the device that handles all of the incoming data from the probes and sends it via cellular GSM communication to the server. When choosing an appropriate device for this task, we had four main requirements for the device to meet. These requirements include: small size, inexpensive, able to communicate via GSM and XBee, and save data received from the probes to and SD card. The reason these requirements were chosen were threefold; to keep within our allotted budget of $1400, to allow for encapsulation in a small package for easy transportation and setup of the master node, and to be able to receive data from the probes, save it, and send it to our server periodically for analysis.

3.2 Platform

In choosing the hardware for the master node a number of options were considered. One option considered was the Arduino Uno as to be consistent with the probes. Since the standard Arduino Uno board does not contain XBee, SD, or GSM modules, each of those would have to be added using shields. This was deemed unfeasible due to the difficulty in interfacing all of these separate shields, as well as the cost of buying each shield individually. An alternative was found in the GBoard, as seen in Figure 3-1; it has a GSM/ GPRS module, SD card slot, and XBee port all built in to the board, and at a low cost it fit all of our requirements.
The only detractor for the GBoard was that it required a USB to Serial adapter to communicate with a computer, but this was inexpensive and was shipped with the GBoard. As well, since the GBoard had the same microcontroller as the Arduino Uno, the ATmega328, we were able to develop code on the Arduino Uno that could be transferred over to the GBoard. This proved useful later when the master node to probe communication and the other master node functions were being written independently of each other.

### 3.3 Software Flow

When the master node powers up, it initializes both the GSM module and the SD card. Once this is complete, it begins waiting for data from the XBee. When a packet is received by the master node, it is parsed for the start-of-packet frame to avoid garbage data sent from the XBee modules. There are three different packets that the master node can receive: the initialization packet, the synchronization packet, and the data packet. Each packet has different functions and the master node performs different tasks accordingly, as seen in Figure 3-2.
When the master node receives an initialization packet from a probe, it goes into synchronization mode. The master node waits for five consecutive synchronization packets from the probe. When the five packets are received, the average round-trip-time, RTT, between the master node and probe is calculated and used to calculate the time offset between the master node and the probe, according to the following equations:

\[ \overline{RTT} = \frac{Total \ RTT}{5} \]  

\[ Offset = timestamp_{master} - timestamp_{probe} + \overline{RTT} \]  

Figure 3-2: Master Node Flowchart
Where $\text{timestamp}_{\text{probe}}$ is the last received timestamp from the probe. Since the master node and probes have different time offsets, determining the offset for each probe allows us to calculate the discovery time of a Bluetooth device relative to the master node. Once the offset for the probe is calculated, it is sent to the server as seen in Figure 3-3.

![Master Node Synchronization Flowchart](image)

**Figure 3-3: Master Node Synchronization Flowchart**

The third packet is the data packet, which is structured as discussed in section 2.4.1. When a data packet is received, the packet is validated by checking to see if the length of the packet is equal to the length value located in the second byte of the packet. If it is determined to be valid, the packet
is saved to the SD card and the first three bytes of the data packet are sent back to the probe over XBee as an acknowledgment. This is seen below in Figure 3-4. Finally, every five minutes the master node takes all of the data saved to the SD card and sends it to the server via GSM. Further information on the sending of data over GSM is located in the next section.

![Diagram](image)

Figure 3-4: Master Node Data Flowchart

### 3.3 Master Node - Server Communication

In the initialization process, the GSM module on the GBoard receives a number of commands to initialize for HTTP communication. The initialization commands register the module with the network and assign the correct settings to allow an HTTP connection with our selected network. As well, the address for our server and the encoding type of the data to be sent to the server are set. Regardless of the contents of the data being sent to the server, the same steps are followed. First, a function called StartHttpDataSend takes as parameters the length of the data being sent and a timeout for how long the module should wait for the data to be added. These parameters are sent to the GSM module using the command AT+HTTPDATA="length","timeout". The length that is passed to the function is modified in two ways. Seven is added to the length to account for “UPLOAD=” being prepended to the data. This is used by the server to identify the data being received. The second way it is modified is by adding \((\text{length} / 100) * 2\) to the length. This is done because every time a data packet is added to the GSM module from the SD card, the module adds two extra bytes to the data. Next, packets are added to the GSM module using the function AddHttpData which has two parameters: the packet being sent and whether or not it is the last packet being added to the module. Multiple data packets are added to the GSM module when the master node is sending the data stored in the SD card to the server. When the master node is ready to send its data to the server, data is taken out from the SD card in 100 byte chunks, which the GSM module buffers in its internal memory. This is done to save memory, as the data taken out of the SD card must be saved in SRAM. When all of the data is added to the GSM module, the function SendHttpData is called which takes the data and sends it using the HTTP connection to
the server as an HTTP POST request. The POST request is used because it allows the storing of data in the request’s message body. This allows us to put all of our data in the body of the POST request and send it to the server where it can be received and analyzed.

3.4 Memory Limitations

As seen in Figure 3-5, the ATmega328 microcontroller that is used by the GBoard has 2048 bytes of SRAM.

![Data Memory Map](image)

Figure 3-5: SRAM Memory Map on ATmega88PA, ATmega168PA and ATmega328 [9]

This proved to be a problem when combining all of the code required for the master node. While the SD card, XBee communication, and GSM communication worked separately, once combined the master node produced strange behaviour and did not operate correctly. It was difficult to determine the cause of this behaviour as there are no built in debugging tools provided with the Arduino IDE, but by measuring the memory usage of each portion of code we determined the cause of this behaviour to be a lack of memory. Of the 2048 bytes of SRAM available, approximately 1800 were useable by us; however we required over 2500 bytes.

One main cause of our memory usage was the large use of Strings in the code for the master node. Using Strings was required because the GSM module utilizes String based commands. When the GBoard turns on, the Arduino bootloader on the ATmega328 microcontroller takes all Strings used in the program and puts them into SRAM. This is done by default as SRAM is much faster and easier to read from than program memory. When the String is called in code (ex: Serial.println(“Print this line”)), the CPU simply takes the String out of its location in SRAM and puts it onto the hardware serial lines. Since we were using more memory then we had access to our in code, this posed a problem. To circumvent this issue we tried a number of different ideas: we removed unused functions, used global variables instead of defining variables in functions, and reduced the number of function calls within functions. The latter was done because every time a function is called inside a function, all its variables are added to the stack, which is located
in SRAM. If function A calls B, which in turn calls C; A, B, and C’s variable are located on the stack in SRAM. This is not ideal because of our tight memory constraints. These ideas all worked to reduce our memory usage, but even after these revisions we were still having memory issues. Another solution was found in bypassing the SRAM completely by using the program memory to store Strings. Program memory on the ATmega328 has 32kB of available memory, as seen in Figure 3-6.

![Figure 3-6: Program Memory Map for ATmega328 [9]](image)

This is where the program code is stored so we needed to ensure that any data added to program memory stayed within the remaining memory bounds to prevent overwriting program code. Using this idea, we found a way to use the program memory as an intermediate between the serial lines and the CPU, bypassing the SRAM completely. This gave us enough space in memory to run the program successfully.
Chapter 4 - Power Systems

4.1 Overview

We wanted to design our probes to allow for easy deployment and monitoring of any roadway. The objective was to create small, compact, and portable power sources that were also solar rechargeable. The power sources also had to be as inexpensive as possible because we had to build seven of them; one for each probe and one for the master. Originally the desired run time for the power source was set at one week, but after running tests on a probe and determining its average current draw, the desired run time was reduced to 24 hours. For a probe to function properly it requires a minimum input voltage of 6 V, and a minimum current of 130 mA. We designed and implemented two power designs for this purpose. The first design was a solar rechargeable lithium ion power system. The second design used a 12 V lead acid battery to power the probes and master node. Both designs were used to limit costs.

4.2 Initial Design

During our search for a suitable battery for our power source we came across many options. The first one that we investigated was the use of 9 V NiMH rechargeable batteries. These batteries have a rating of approximately 200 mAh – 400 mAh [10]. As shown in Figure 4-1, our design would place several of these batteries in parallel hooked up to a solar panel that would charge them all as they powered a single probe. There were several problems with this design. To produce a runtime of 24 hours, a single power source would require seven 9 V NiMH batteries in parallel. This would cause any charge current produced by the solar panel to be divided by seven; reducing the speed of the charge cycle. The NiMH batteries also lose their voltage easily in cold weather [11] and would require specific protection circuitry to prevent overcharging and discharging. Finally, their cost per mAh is higher than comparable alternatives.
4.3 Intermediate Design

Our second design explored the use of lithium ion batteries as a power source. Lithium ion batteries are able to produce larger mAh ratings per battery compared to NiMH or Alkaline batteries [10, 11, 12]. We chose to purchase a 3.7 V lithium ion battery with a 1300 mAh rating because the lower voltage batteries tended to have higher mAh ratings. Lithium ion batteries require a special charge sequence so a lithium ion charger was purchased which was powered by a 1 W, 6 V solar panel. To step the voltage up to a usable voltage above 6 V, we purchased a DC/DC step-up converter that increased the voltage from 3.7 V to 12 V. The intermediate design is shown in Figure 4-2. After the design was finished a test was run to determine the battery life when connected to a probe. The battery drained after three hours. The cause of the short battery life was due to not accommodating for the extra current that the converter would draw from the battery. The probe draws a maximum of 130 mA, but the converter draws 480 mA from the battery pack. This was not acceptable for our design.
4.4 Final Design

For the final design we decided to stay with the idea of using a 3.7 V battery and step it up to a higher voltage. To increase the battery life we purchased two 3.7 V, 4400 mAh lithium ion batteries. We also purchased a larger 3.7 W, 6 V solar panel that could charge the batteries faster than the 1 W, 6 V solar panel. Finally we purchased two solar optimized lithium ion chargers; one for each battery pack. Solar optimized chargers are designed to be able to function with variable input voltages and currents that a solar panel would normally produce. All these new components increased the total cost of a single power source so a compromise was made where one probe would be solar powered and the other six probes would be powered by 12 V, 7000 mAh rated lead acid batteries donated to us by our advisor, Dr. Robert McLeod. This was done to show that if sufficient funds were available, a solar rechargeable power source would be a viable option. A single lead acid battery can power a probe for approximately 58 hours, after that the battery would need to be removed from the probe and recharged.

For the solar rechargeable system, two 4400 mAh batteries were connected in parallel creating one 8800 mAh rating power source. Each of the batteries were connected to a solar optimized lithium ion charger and then connected in parallel to the 3.7 W, 6 V solar panel. To step the voltage up to a value higher than the minimum of 6 V, we initially considered the use of our purchased step converters which would step-up the voltage to 12 V. However, we realized that if we did this, we would be supplying 6 V more than necessary to the probe, increasing the power supplied to the probe and lowering the total run time. To remedy this, we designed our own custom DC/DC step-up converter that stepped the voltage up to a value of 6.5 V, much closer to the minimum of 6 V required. This reduced the current draw of the step-up converter, which increased the battery life of our system. An image of all the components for our final design is shown in Figure 4-3.
4.5 Step-up Converter

For our step-up converter, we chose to build a boost converter as shown in Figure 4-4. The way a boost converter works is by using energy stored in an inductor. When the switch in boost converter is closed, it creates a loop with just the source and the inductor. During this period, current flows in the clockwise direction and the inductor stores this current; increasing the voltage drop across it. When the switch opens, the current is forced in the same direction by the diode but is reduced because of the increased impedance of the circuit. The inductor resists this change in current and flips its polarity, increasing the voltage seen by the load.

![Boost Converter Schematic](image)

The current draw of a probe was measured at different states by attaching a lead acid battery and measuring the current and voltage inputs. It was determined that a probe requires variable amounts of power based on whether or not the Bluetooth or XBee devices are transmitting. The
maximum current used by the probe was found to be 130 mA, therefore, the step-up converter must be able to provide at least this amount. This information was used to build a Multisim model for our solar rechargeable power source. To implement the switch in the step-up converter, a 555 timer powered by the 3.7 V battery, was configured to output a square waveform at 4.6 kHz. That output waveform was then connected to the gate of an nMOS transistor which acted like a switch. It was difficult to create a working step-up converter because the resistance of the probe is variable depending on which device is transmitting, 40 Ω when the XBee module is transmitting or receiving, and 80 Ω otherwise. These small impedances for the load also created some voltage division problems with our circuit. An image of our hardware implementation is shown in Figure 4-5.

![Figure 4-5: Step-up Hardware Prototype](image)

When trying to implement our Multisim model in hardware for the first time, we were unable to get it to function correctly; there were a few reasons for this. In our Multisim model we made some assumptions that were incorrect. We used a digital clock that had a square wave output of 5V and 0V to control the gate of the transistor. In our actual implementation we only had a peak voltage of 3.2V on our square waveform from the 555 timer. This caused our gate voltage to be lower during the turn on period of the transistor which was limiting the amount of current through the transistor. This in turn limited the amount of current charging the inductor and lowered our output voltage. To more accurately model our hardware implementation, we switched the digital clock in Multisim with a 555 timer. This was still incorrect because it was an ideal 555 timer where the output voltage was equal to the input voltage. To make our model as accurate as
possible we used a function generator that output a square waveform with an amplitude of 3.2 V. This produced the most accurate model of our hardware implementation. With the new gate voltage of 3.2 V in the Multisim model, we still needed to find a way to increase the current flowing through the transistors. To fix this problem, three transistors were placed in parallel allowing for an increase in current flow through the inductor. This is shown below in Figure 4-6.

![Multisim Model of the Step-up Converter](image)

Figure 4-6: Multisim Model of the Step-up Converter

Another problem we had was with blown components. The inductors we were using were not rated for the amount of current that we were trying to push through them. This resulted in us running tests and diagnostics with a broken inductor. This was fixed by choosing an inductor with a smaller mH value but a larger current rating. Once the hardware implementation was updated with these fixes it worked correctly. We did tests on the hardware implementation of the step-up converter to determine the average current that is drawn from the battery while the probe is running; it is approximately 230 mA. This was done to give us a rough estimate of how long our power source would last without charging. This was calculated by dividing the battery rating by the average current draw.

\[
Battery \ Life = \frac{8800 \ mA h}{230 \ mA} = 38.3 \ hours \quad (4.1)
\]

Upon completion of a working hardware implementation of the step-up converter, a single sided PCB design of the circuit was created using the program Fritzing. We then soldered on all of the
hardware parts and wired the board to the power sources, as seen in Figure 4-7. Headers were soldered to areas of the PCB that contained more fragile parts so that they could be replaced without further soldering.

Figure 4-7: Step-up Converter
Chapter 5 - Server

5.1 Overview

The server is responsible for receiving packets from the master node and converting it into data that can be handled by our database and later displayed on our front end systems. To provide this functionality we used a MySQL database running on our server as well as server side PHP code to analyze data and create our website functionality. To be consistent with the goals of our project, data stored in the database must be updated in real-time in order to keep the traffic information current.

Two separate front end displays will allow separation of information for corporate and public use. The corporate website allows for the probe network to be set up and modified. As well, the information and analysis that is displayed on this site is more detailed and specific to the needs of a corporate user providing information on average time, average velocity, and vehicle counts through a roadway. The public system displays the current traffic status with updates in real-time. This display can be used to help make decisions on which route to take when driving through a monitored area. Originally the public display was going to be implemented on a smartphone application that would display the current traffic status to the user. However, as the project progressed, we decided to have the website serve both desktop and mobile users. This simplified the development process while also allowing users with any type of device to access our site, not limiting access to a specific mobile operating system or web browser.

5.2 Data Handling

When data is received by the server it must be parsed due to the encoding on the data. The parsed data is then analyzed by a PHP script which places it in the appropriate tables in the database. As an example, the traffic_data table from our database is shown in Table 5-1. This data can be retrieved programmatically by querying the database.
We chose to use PHP because it is a server-side scripting language for website development that provides a simple interface to our MySQL database.

### 5.2.1 Data Modification

When data is received at the server from the master node a PHP script is initiated to handle the data. There are two different data types that can be received at the server from the master node. The first is the time offset for a specific probe. When this value is received the offset for the given probe is updated in the database. The second data type that can be received is the list of devices that were discovered at each probe since the last update from the master node. Due to the encoding that is done by the probes discussed in section 2.4.1, the probe ID, MAC address detected, and discovery time have to be decoded. Each device packet in the data received is decoded in a manner analogous to the encoding process. Once a MAC address has been decoded with its corresponding probe ID and discovery time, a function insertIntoTables is called with these three values passed to it as parameters. The first step in handling this entry is the script queries the probe_location table to get the correct time offset. This offset, as mentioned in section 3.4.4, is added to the discovery time. The script then continues by running a query of the traffic_data table to see if the MAC address was previously recorded in the table. As the flowchart in Figure 5-1 shows, if the MAC address is found, the script compares the probe ID of both detections of the MAC address. The start and end probe IDs are compared to the table probe_path to ensure they correspond to a valid path. If a valid path is found, a probe pair is created and stored in the probe_pairs table.
A probe pair contains the probe ID of both the start and end probes of a discovered device as well as the discovery time for each probe. This information is used to find the velocities of vehicles and detailed analysis for the corporate website. An example of the probe_pairs table is shown below in Table 5-2. This process is repeated for each device packet received from the master node.

**Table 5-2: probe_pairs Structure**

<table>
<thead>
<tr>
<th>Data</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device Mac</td>
<td>Last 3 bytes of MAC Address</td>
</tr>
<tr>
<td>Probe ID Start</td>
<td>ID of the first probe device was recorded</td>
</tr>
<tr>
<td>Probe ID End</td>
<td>ID of the first second device was recorded</td>
</tr>
<tr>
<td>Timestamp Start</td>
<td>Time when device was entering the intersection</td>
</tr>
<tr>
<td>Timestamp End</td>
<td>Time when device was leaving the intersection</td>
</tr>
</tbody>
</table>
5.2.2 Display Data Refresh

At the completion of the first PHP script a second PHP script is immediately initiated that is responsible for updating the current data and archiving previous data. To store the current data, the server’s current time is included with the current data, and inserted into the archive_data table. The same sequence is followed to archive pulses from the probe_pulse table to archive_pulse table. Once all the rows from current_data and probe_pulse have been copied to archive_data and archive_pulse they are cleared so they can be repopulated with data.

The script continues by retrieving all of the rows in the traffic_data table. Any entry left in this table means that a valid match was not found to create a probe pair. This is considered to be a probe pulse and used in conjunction with the data in probe_pairs to determine the traffic count at that probe. This can occur if the network failed to detect the device at two separate probes in a valid. The script goes through each row in the traffic_data table and for each instance of a probe pulse; the count in probe_pulse is incremented for the row corresponding to the same probe ID. The script queries the probe_pairs table for all entries and the difference in time between the two discoveries is added to the total_time column in the current_data table with the correct path ID for the probe pair. As each entry is added to the current_data table it is removed from probe_pairs.

By having these two PHP scripts run every time the server receives data from the master node, all data that is stored will be updated in real-time.

5.3 Corporate Website

The corporate site is our front end system that displays our current probe locations as well as our advance analysis of the most recent data that has been collected from the probes. It also allows our probe network to be set up, modified and saved on the database while it is being deployed at a roadway from a mobile device or at any other time from a computer.

5.3.1 Probe Setup/Modification

To create an interactive display that would show where probes had been placed along a roadway, we decided to use Google Maps. To add a Google Map display to our website we used the publically available Google Maps API Ver. 2 [14] using JavaScript. Utilizing the API from Google we are able to display streets and intersections at the correct latitude and longitudes, as well as markers, routes and other interactive features overlaid on a map as shown in Figure 5-2.
When the website is first opened it retrieves all of the probe information by querying the database. This query returns the probe ID as well as the latitude and longitude of each probe that was previously set up. This information is used by the Google Maps API to place a marker on the map at the correct location. To increase the ease of use, the marker clearly displays the probe ID on the marker so that they can be easily identified from each other. Each marker has two event listeners which wait for specific triggers. The first event is a mousedown event; this event happens when a user clicks the left mouse button down while the cursor is over the visible marker on the map. The mousedown event causes the probe ID to be displayed in the dropdown box below the map in the probe options panel, updates textboxes with the current latitude and longitude as shown in Figure 5-3. As well, the selected marker turns green while the unselected
markers return to a red state. The second event handler is a drag event. A drag event happens when a user holds the left mouse button down over a marker and proceeds to drag the marker across the map. When this event is triggered it continuously updates the latitude and longitude text fields in the probe options panel with the exact location that the probe has been moved to.

![Probe Options Panel](image)

**Figure 5-3: Probe Options Panel**

To add a new probe to the map, the user presses the New Probe button in the Probe options panel. If the button is pushed from a mobile device that has a valid GPS position the probe will be placed at that point. If a device with GPS is not available, probes can be added to the map by dragging and dropping markers at their corresponding GPS locations. Markers can alternatively be placed by manually entering the latitude and longitude into the text boxes and clicking the apply button. To save all the changes that have been made to the database, the Save Changes button is clicked and an HTTP POST request is sent back to the server. The website is automatically refreshed and the location data is sent in a POST request and stored in the database.

With the markers in the correct positions routes, can be created so that data can be stored when it is sent from the probes. To create a route there are two drop down boxes labeled for the start and end probe ID in the Route Information panel shown below in Figure 5-4. To calculate the routes we use Google’s Route Services, a function that is included with the Google Maps API. This function determines the shortest driving path between two GPS locations and calculates the distance of the path. This allows us to enter a latitude and longitude for the start and end points of the route that are specified by the markers that were selected in the drop down boxes.
Figure 5-4: Route Information Panel

The created routes are used later in the consumer statistics to display the traffic densities. When the save route button is pushed, we do not save the entire route. Instead, the distance of the calculated route and the start and end probe IDs are stored in the probe_path table. There is currently a problem with using the provided distance from the Route Service. As it can be seen in Figure 5-5 below, the route does not line up completely with the start and end probe.

Figure 5-5: Route Setup Displayed on Map
This is due to the way Google calculates routes. The Route Service finds a route between two addresses, so when we supply the function with a latitude and longitude as the start and end position of the route, the street address is approximated through a process called reverse geocoding and no longer is the exact point originally specified [15]. The route displayed is not important for the corporate website; however it is used for the consumer display to show the traffic status, and must be displayed properly on the map.

5.3.2 Analysis Display

Currently our website has a few different options for displaying the results of the analysis of our data. One display option we have provided is a line chart that represents the count at a selected probe during each update interval. The user is able to select which probes' data is presented. This can be seen below in Figure 5-6.

![Figure 5-6: Line Chart of Traffic Count for a Probe vs. Time](image)

By adjusting the sliders below the chart the timeframe being displayed will be modified accordingly. This chart displays the archived values for an individual probe. Depending on user selection, either the archive_data table or archive_pulse table is queried for all probes on website
load. The values retrieved are saved to variables corresponding to each probe so that each probe can be individually selected and displayed on the chart.

5.4 Public Website

Due to privacy concerns and the value of our data we decided that the only results that we would share with users of our public website would be a map showing current traffic status. This could be updated to use the archived values from our database to allow users to view the traffic status at a previous point in time. This display uses the same Google Map interface as the corporate site; however the location of the probes is not shown to the user. Instead, each valid route that is saved in the database is recalculated and displayed on the map using colors to indicate traffic status. As an example, a green path overlaid on the road would symbolize that traffic is flowing at a normal rate, while a yellow overlay would mean slow and a red overlay symbolizes that traffic is barely moving. This can be seen below in Figure 5-7.

![Traffic Status Display](image)

Figure 5-7: Traffic Status Display
To decide what color needs to be displayed on the map, we utilize the average time a vehicle takes to drive between the start and end probes in a path. This information was previously calculated during the data handling, seen in section 5.2.1, by the server and stored in the database. A query is done to find the average time, and this is compared to specific thresholds set for the states of traffic flow. As seen in the formula below, the time it would take to cover the distance between the probes going the speed limit is divided by the average time calculated from the database.

\[
\% \text{ traffic flow} = \frac{\left( \frac{\text{distance}_{path}}{\text{speedlimit}_{path}} - \frac{\text{avgtime}_{path}}{} \right) \times 100}{\frac{\text{distance}_{path}}{	ext{speedlimit}_{path}}}
\]  

(5.1)

This gives us the percent difference from the nominal time which is the expected time taken given the speed limit and distance. If the \% traffic flow is above the threshold for a red line, 75\%, the route will be displayed as red. If the \% traffic flow is between the threshold for a yellow line, 40\%, and the threshold for a red line, the route will be displayed as yellow. If the \% traffic flow is below these thresholds, the route will be displayed as a green line. These values were selected before testing and will be refined in the future.
Chapter 6 - Casing

A requirement of the probes was that it had to be easy to transport and deploy. This required the casing to be small and robust. We selected black, hard plastic cases of three different sizes. The solar rechargeable probe required a slightly larger case size of 120mm x 120mm x 60mm because of the extra components inside. The rest of the probes were able to fit in smaller cases since the battery would be outside the case; this led us to a case size of 150mm x 50mm x 80mm. The master node used a similar case, but had a distinct lid to differentiate between the probes and master node. The different casings can be seen below in Figure 6-1.

![Figure 6-1: a) Master Node b) Solar Powered Probe c) Basic Probe](image)

With these sizes the probes were able to fit snug inside the cases without allowing too much movement inside. Once the position of the probes inside the cases were determined, two holes needed to be drilled; one for the Bluetooth antenna and one for the XBee antenna. It was later decided that a “ON” and “OFF” switch should be added, along with a LED to represent the two states “ON” and “OFF”. Two more holes were drilled on the other side of the case to mount the LED and switch onto it. Then the LED and switch were soldered onto the probe. Eventually the regular LED was replaced with an RGB LED. With LED code added to the probe, unique colours were able to be outputted to the RGB LED signifying different states that the probe is in as specified in section 2.4. This was done to make debug during the testing of the probes easier. We were able to determine if probes were not properly synchronizing, as well as if probes were actually picking up Bluetooth devices. Once the probe casing was complete, the 12V batteries were attached to the bottom of the case where another hole was drilled to allow power inputs inside the case.
Chapter 7 - Testing

7.1 Bluetooth Discovery Test

To test whether the Bluetooth modules that we had selected could discover devices travelling at the speed limit of 80km/h, we placed one of our probes in multiple positions alongside the road and had someone monitoring the detections. To ensure the presence of Bluetooth devices, we took a vehicle containing multiple devices and drove past the sensors location on the street while doing the speed limit. This test showed that the Bluetooth shields were able to discover devices that passed by at typical speeds.

7.2 XBee Distance Test

To determine the distance that a connection could be achieved by XBee communication, we took one probe and the master node outside along a straight section of sidewalk and tested that they would connect while varying the distance between them. One problem that we encountered during this test was the difficulty of having a direct line of sight between the two devices. With this problem our distances were limited to 200 meters at which point the probe was still connecting to the master node.

7.3 Full Scale Deployment

As a complete test of our system we deployed four probes along a major thoroughfare to capture the traffic over a 24 hour timeframe. This test included using the master node to communicate with our server to store the data. This was then displayed in real-time on our front end systems. To deploy our system, we mounted the probes and master node to light standards along the thoroughfare using steel strapping.
Chapter 8 - Analysis and Results

During our full scale live test as discussed in section 8.3, we were able to collect data from both directions of a major thoroughfare in Winnipeg. In the pre-processed data there were several redundancies that had to be removed in order for the data to be representative of actual traffic counts. Additionally, we noticed probes were detecting vehicles traveling in both directions instead of the one direction we intended. This occurred because the probes’ Bluetooth range was larger than our initial test indicated; this had the effect of skewing our data. However, this did not seem to be occurring with Probe 0 which was set up in the south east section of the thoroughfare. The shape of the traffic data from Probe 0 represented that of normal flow including rush hour peaks and early morning valleys; as seen in Figure 8-1.

![Figure 8-1: Traffic Counts at Probe 0](image)

In addition to traffic counts, we were also able to process average velocities as shown in Figure 8-2. Using the device discovery timestamps embedded in the device packet, as seen in section 3.4.1, we were able to eliminate instances where a probe discovered a device on the unintended side of the thoroughfare. This is because if, for example, there is a path from probe 0 to probe 2, and a device is discovered by both probe 0 and probe 2, it will only be considered as a valid pairing if the discovery timestamp for probe 0 is less than the discovery timestamp for probe 2. During our first full scale test, we were not getting as many probe pairs as we expected to get
based on the traffic count data. Because of this, the average velocity calculated is not statistically meaningful at this point. Further testing and modification of the system is required to produce reliable velocity trends.

Figure 8-2: Average Velocity vs. Time of Day
Chapter 9 - Conclusion and Future Work

9.1 Conclusion

The design and implementation of a real-time Bluetooth detecting probe network has been presented. The goal of this project was to create a real-time traffic monitoring system that is easy to setup and deploy, provides real-time updates, and is inexpensive compared to similar alternatives. Three different parts were designed and prototyped: the probes, the master node, and the server based functions. A system was created that allowed the traffic density and average vehicle velocity to be detected and calculated using a probe network placed along a roadway. The master node coordinates the probe network using XBee communication and sends the data received from the probes periodically to a server over GSM. When testing the prototype system, promising results were received. During testing, traffic counts from the probes closely resembled expected counts for specific times of the day. As well, velocities calculated correlated with traffic counts and rush hour times.

9.2 Future Work

There are extensions that were intended or considered to be implemented in the final prototype. One such extension to further decrease the deployment time for the probe network would be to have a GPS chip installed in each probe. The probes would then automatically send their locations to the server so they would not have to be added manually through a website. As well, each probe would get their specific ID from the server, instead of having the IDs hardcoded on the device. This would allow for all the probes to have the same code installed and be updated dynamically on deployment.

Another extension would be to create PCB boards for all aspects of our project instead of using off the shelf parts. This would lower the overall cost of probes and the complete system.
References


Appendix A: Source Code

A-1 BT_Pro.cpp

A-1.1 Initialization

void BT_Pro::xBeeInit() {
  setLed(REDLED, LOW);
  setLed(GREENLED, LOW);
  xBeeWake();

  // Enter Command Mode
  delay( 1100 );
  Serial.print( "+++" );

  while( Serial.available() < 1 );
  while( Serial.available() ) {
    Serial.print( (char)Serial.read() );
    delay(10);
  }

  // Set probe PanID
  Serial.println( "ATID1337" );

  while( Serial.available() < 1 );
  while( Serial.available() ) {
    Serial.print( (char)Serial.read() );
    delay(10);
  }

  // Set probe address
  Serial.println( "ATMY0" );

  while( Serial.available() < 1 );
  while( Serial.available() ) {
    Serial.print( (char)Serial.read() );
    delay(10);
  }

  // Set association parameters
  Serial.println( "ATA16" );

  while( Serial.available() < 1 );
  while( Serial.available() ) {
    Serial.print( (char)Serial.read() );
    delay(10);
  }

  // Set hibernate parameters
  Serial.println( "ATSM1" );
while( Serial.available() < 1 );
while( Serial.available() ) {
  Serial.print( (char)Serial.read() );
  delay(10);
}

Serial.println( "ATD83" );

while( Serial.available() < 1 );
while( Serial.available() ) {
  Serial.print( (char)Serial.read() );
  delay(10);
}

Serial.println( "ATD70" );

while( Serial.available() < 1 );
while( Serial.available() ) {
  Serial.print( (char)Serial.read() );
  delay(10);
}

// Set as end device
Serial.println( "ATCE0" );

while( Serial.available() < 1 );
while( Serial.available() ) {
  Serial.print( (char)Serial.read() );
  delay(10);
}

// Write changes
Serial.println( "ATWR" );

while( Serial.available() < 1 );
while( Serial.available() ) {
  Serial.print( (char)Serial.read() );
  delay(10);
}

// Exit Command Mode
Serial.println( "ATCN" );

// delay(100);

delay(10000);
// setLed(REDLED, LOW);
setLed(GREENLED, HIGH);
}

A-1.2 Synchronization

bool BT_Pro::xBeeSync() {
Serial.flush();
// char initPacket[4];
// char syncPacket[9];
char response[10];
setLed( BLUELED, LOW );
setLed( GREENLED, LOW );

// Set code byte, length, and probe id for init packet
_packets[_packetIndex][0] = 'J';
_packets[_packetIndex][1] = 'e';
_packets[_packetIndex][2] = 'n';
_packets[_packetIndex][3] = 'n';
_packets[_packetIndex][PACKET_CONTROL] = 0x41;
// Control byte set as data packet
_packets[_packetIndex][PACKET_LENGTH] = 0x43;
_packets[_packetIndex][PACKET_PROBE_ID] = _probeId;
_packets[_packetIndex][PACKET_PROBE_ID+1] = 0x00;
Serial.println( _packets[_packetIndex] );

if( getResponse( response, (long)3000 ) ) {
    Serial.print( response );
}
else {
    Serial.println( "Bad response" );
}

// Init ack received
if( response[0] == 0x41 && response[1] == 0x43 &&
    response[2] == _probeId ) {
    // Serial.print( "Received init ack!" );
    for( _i = 0; _i < 5; _i++ ) {
        setLed( GREENLED, HIGH );
        int j = 0;

        _packets[_packetIndex][PACKET_CONTROL] = 0x42;
        _packets[_packetIndex][PACKET_LENGTH] = 0x4D;
        _packets[_packetIndex][PACKET_PROBE_ID] = _probeId;
        _packets[_packetIndex][PACKET_PROBE_ID+1] = 0xF0;
        // Set up the marker byte
        _packets[_packetIndex][PACKET_PROBE_ID+6] = 0x00;
        // Terminate the packet
        long current = millis();
        memcpy( &_packets[_packetIndex][PACKET_PROBE_ID+2], &current, sizeof( long ) );

        // Change any bytes in the character string that are 0x00 to 0x80 to prevent the string from terminating prematurely
        if( _packets[_packetIndex][PACKET_PROBE_ID+2] < 0x40 ) {
            _packets[_packetIndex][PACKET_PROBE_ID+1] |= 0x08;
        }
    }
}

}
46

```c
isEqual = _packets[_packetIndex][PACKET_PROBE_ID+2] += 0x40;
if ( _packets[_packetIndex][PACKET_PROBE_ID+3] < 0x40 ) {
    _packets[_packetIndex][PACKET_PROBE_ID+1] |= 0x04;
    _packets[_packetIndex][PACKET_PROBE_ID+3] += 0x40;
}
if ( _packets[_packetIndex][PACKET_PROBE_ID+4] < 0x40 ) {
    _packets[_packetIndex][PACKET_PROBE_ID+1] |= 0x02;
    _packets[_packetIndex][PACKET_PROBE_ID+4] += 0x40;
}
if ( _packets[_packetIndex][PACKET_PROBE_ID+5] < 0x40 ) {
    _packets[_packetIndex][PACKET_PROBE_ID+1] |= 0x01;
    _packets[_packetIndex][PACKET_PROBE_ID+5] += 0x40;
}
Serial.println( _packets[_packetIndex] );
setLed( GREENLED, LOW );
getResponse( response, (long)300 );
if( !( response[0] == 0x42 && response[2] == _probeId ) ) {
    setLed( BLUELED, HIGH );
    setLed( GREENLED, HIGH );
    return false;
}
else {
    setLed( BLUELED, HIGH );
    setLed( GREENLED, HIGH );
    return false;
}
setLed( BLUELED, HIGH );
setLed( GREENLED, HIGH );
return true;
```

A-1.3 Device Discovery

```c
void BT_Pro::inquiry() {
    boolean inquiryFinished = false;
    if( isOn() ) {
        // debugPrintln( "Starting device inquiry..." );
        // Clear packet buffer
        for(_i = 0; _i < PACKET_COUNT; _i++ )
```
clearPacket(_i);

// Initialize number of devices discovered on this inquiry
_numberOfDevices = 0;

// Send the inquiry command
sprintf(_theCommand, "%s %d", CMD_INQUIRY, INQUIRY_TIME);
sendCommand(_theCommand);

// Wait for the response from the inquiry
waitInquiryAnswer(INQUIRY_TIME * 1000);

// If debug is set, print out the results
// debugPrint("Found ");
// debugPrint(_numberOfDevices);
// debugPrintln("devices!");
else {
    delay(1000);
    // debugPrintln("The device is not on, cannot start inquiry");
}
}

void BT_Pro::waitInquiryAnswer(long inquiryTime) {
    char dummy[4];
    long previous = millis();
    char block[BLOCK_MAC_SIZE + 1];
    block[BLOCK_MAC_SIZE] = 0x00;

    while((millis() - previous < inquiryTime) &&
          _numberOfDevices < MAX_DEVICES)
    {
        // Look for keyword and save device.
        if (_stream->available()) {
            dummy[0] = _stream->read();

            if (dummy[0] == 'I') {
                while(_stream->available() < 2);
                dummy[1] = _stream->read();

                if (dummy[1] == 'A') {
                    dummy[2] = _stream->read();

                    if (dummy[2] == 'L') {
                        while(_stream->available() < BLOCK_MAC_SIZE);
                        _stream->read();

                        for(uint8_t x = 0; x < BLOCK_MAC_SIZE; x++) {
                            block[x] = _stream->read();
                        }
                    }
                    _numberOfDevices++;
                }
            }
        }
    }
}
parseDevice(block);
}
}
}

// Search inquiry end
if ( (dummy[0]=='Y') ){
    while(_stream->available() < 2 );
dummy[1]=_stream->read();
    if (dummy[1]==' '){
        _stream->flush();
        break;
    }
}
}

if ( _numberOfDevices > 0 )
    sendDeviceList();

// Condition to avoid an overflow (DO NOT REMOVE)
if( millis()-previous < 0 )
    previous=millis();
}

A-1.4 Transmission
void BT_Pro::sendDeviceList() {
    // debugPrintln( "Sending device list now ..." );

    setLed(REDLED, HIGH);
    setLed(GREENLED, LOW);
    bool acked = false;
    int nackCount = 0;
    long current = millis();
    char response[10];

    uint8_t length = 8;

    while ( !acked ) {
        while ( _packets[_packetIndex][length] != 0x00 )
            length++;

        _packets[_packetIndex][0] = 'J';
        _packets[_packetIndex][1] = 'e';
        _packets[_packetIndex][2] = 'n';
        _packets[_packetIndex][3] = 'n';
        _packets[_packetIndex][4] = 'y';
        _packets[_packetIndex][PACKET_CONTROL] = 0x44;
        // Control byte set as data packet
        _packets[_packetIndex][PACKET_LENGTH] = length + 0x40 - 5;
        _packets[_packetIndex][PACKET_PROBE_ID] = _probeId;

        Serial.flush();
        xBeeWake();
    }
// Serial.println( "Jenny" );
// delay(50);
Serial.println( _packets[_packetIndex] );

getResponse( response, 1000 );
Serial.print( "Response: ");
Serial.println( response );

char result[40];
sprintf( result, "%hi = %hi\n", response[1], (length + 0x40 - 5) );
Serial.print( result );

if( ((uint8_t)response[1]) == (length + 0x40 - 5) && response[2] == _probeId )
    acked = true;
else {
    if ( nackCount < 10 ) {
        delay( random(128, 1024) );
        nackCount++;
    } else
        acked = true;
}

// _i = 0;

// while( _packets[_packetIndex][i] != 0x00 ) {
//     Serial.print( (_packets[_packetIndex][i]) );
//     _i++;
// }

delay(50);
xBeeSleep();
clearPacket(_packetIndex);
_packetIndex = _packetIndex % PACKET_COUNT;
_numberOfDevices = 0;

setLed(REDLED, LOW);
setLed(GREENLED, HIGH);
A-2 SD_Xbee.ino

A-2.1 Packet Parsing

bool get_request( char* request, long timeout ) {
    long current = millis();
    bool goodRequest = false;
    while( millis() - current < timeout ) {
        j = 0;
        char* pos;

        // Read the serial line for any incoming requests
        while( Serial.available() > 0 && j < 100 ) {
            request[j] = (char)Serial.read();
            delay(2);
            j++;

            // Clear Jenny\n from the array and reset index; good request found
            // if( !goodRequest && ( pos = strstr( request, "Jenny\n" ) ) != 0x00 ) {
                j = 0;
                goodRequest = true;

                while( request[j] < pos + 8 * sizeof( char ) ) {
                    request[j] = 0x00;
                    j++;
                }
            }
            j = 0;
            current = millis();
            while( Serial.available() == 0 && millis() - current < timeout );
        }
        request[j] = 0x00;
        if( goodRequest )
            return goodRequest;
    }
    return goodRequest;
}

A-2.2 Master Node Main Program Loop

void loop() {
    bool packetFound = false;
    long offset = 0;
    // The currently synchronizing probes offset from real-time
    long rtt = 0;
    // Average round trip time
    of the sync packets

    //
long prevTime = 0;          // The time of the previous
ack; used for finding rtt
uint8_t probeTurn = 0;      // The probe that
currently has access to the master
uint8_t syncCount = 0;      // Number of
synchronization packets received, should be 5 when complete
long probeTimestamp;       // The timestamp of the probe
 sent in the packet

while( !packetFound && ( millis() - lastSend < 300000 ) )
{

    // Process packet if one is found
    if( getRequest( packet, 60000 ) ) {
        // Serialprint("It got it!\n");
        uint8_t control = packet[0];
        uint8_t length = packet[1] - 0x40;
        uint8_t probeId = packet[2];
        packet[length] = 0x00;
        packetFound = true;

        if( control == 0x41 && probeTurn == 0 ) {

            // Serialprintln( "Init ack" );
            Serialprint( "%s\n", packet );
            probeTurn = probeId;
            for( i = 0; i < 100; i++ ) {
                packet[i] = 0x00;
            }

            prevTime = millis();

            while( syncCount < 5 && millis() - prevTime < 500 ) {

                if( getRequest( packet, 500 ) ) {
                    control = packet[0];
                    length = packet[1] - 0x40;
                    probeId = packet[2];

                    if( control == 0x42 && probeId == probeTurn ) {
                        packet[length] = 0x00;
                        // Serialprintln( "Sync
ack" );
                        Serialprint( "%s\n", packet );
                    }
                }

            }

            // A bit confusing to look
            at, we want the final timestamp for the offset to be as accurate
            as possible
            // rtt should use the same
            offset = millis();
            rtt += offset - prevTime;

            // Serialprintln( "Sync
ack" );
prevTime = offset;
syncCount++;

if( syncCount == 5 ) {
    // Change any marked
    // bytes back to 0 and copy to probeTimestamp
    if( packet[3] & 0x08 ) packet[4] = 0x40;
    if( packet[3] & 0x04 ) packet[5] = 0x40;
    if( packet[3] & 0x02 ) packet[6] = 0x40;
    if( packet[3] & 0x01 ) packet[7] = 0x40;
    memcpy( &probeTimestamp, &packet[4], sizeof( long ) );

    // Average round trip
time for packets from a specific probe
    rtt = rtt / 5;

    // From now on, any
timestamps from this probeId have offset added to its timestamp
    to obtain real-time.
    // Store this value
    // in a table, indicating the probeId that this value is associated
    // with.
    // Also, any
    // systematic errors introduced by constant delays in the code will
    // be reproduced by all probes, so
    // offsets of all
    // probes should contain equal error and cancel out
    offset = offset - probeTimestamp - (rtt / 2);

    // Send Offset and
    // probe id to server plz
    sendOffsetForProbe(probeId, offset);
}
else if( control == 0x44 ) {
    // Serialprintln( packet );
    // delay(100);
    for( i = 0; i < length; i++ ) {
        if( packet[i] == 0x00 ) break;
    }
}
if( i == length ) {
    writePacketToSd( packet );
    packet[3] = 0x00;
    Serial.write( packet );
}
delay(100);
// printSdContents();
}

if( millis() - lastSend >= 300000 )
{
    delay(100);
    sendData();
    lastSend = millis();
}
A-3 GSM_Shield.cpp

A-3.1 StartHTTPDataSend

```cpp
byte GSM::StartHTTPDataSend( uint16_t dataSize, uint32_t timeout )
{
    char command[36];

    for( i = 0; i < 36; i++ )
        command[i] = 0x00;

    sprintf( command, "AT+HTTPDATA=%u,%u", dataSize + 7 + ( dataSize / 100 ) * 2, timeout );

    if( AT_RESP_OK == SendATCmdWaitResp( command, 1000, 50, "DOWNLOAD", 5 ) )
    {
        Streamprint(*_stream, "UPLOAD=");
        return 1;
    }
    return 0;
}
```

A-3.2 AddHTTPData

```cpp
//Add HTTP data to send via GSM
//dataToSend is the char packet being added to the data
//lastPacket is a simple true or false statement to say whether
//it is the last packet being added
//returns 1 if succesful, 0 if unsuccesful

byte GSM::AddHTTPData( char *dataToSend, byte lastPacket )
{
    Streamprint(*_stream, "%s", dataToSend);

    if( lastPacket == 1 )
    {
        if( RX_FINISHED_STR_RECV == WaitResp(30000, 10, "OK") )
        {
            Streamprint(*_stream, "UPLOAD=");
            return 1;
        }
        else
            return 0;
    }
    return 1;
}
```

A-3.3 SendHTTPData

```cpp
byte GSM::SendHTTPData()
```
{ byte aok = 0;
if( AT_RESP_OK == SendATCmdWaitResp("AT+HTTPACTION=1", 15000, 100, "OK", 4) )
{
aok = 1;
#ifdef DEBUG_PRINT
DebugPrint("Sent");
#endif
}
delay(7000);
sendCommand("AT+HTTPREAD=0,200");
return aok;
}

A-3.4 Initialization

/**********************************************************
Sends parameters for initialization of GSM module

group: 0 - parameters of group 0 - not necessary to be
registered in the GSM
       1 - parameters of group 1 - it is necessary to be
registered
**********************************************************/
void GSM::InitParam(byte group)
{
switch (group) {
case PARAM_SET_0:

    // Reset to the factory settings using AT&F
    SendATCmdWaitResp("AT&F", 1000, 50, "OK", 5);
    break;

case PARAM_SET_1:
    InitParam1();
    InitParam2();
    break;
}
}

void GSM::InitParam1()
{
    // Registers with network using AT+COPS=0
    SendATCmdWaitResp("AT+COPS=0", 10000, 50, "OK", 5);

    // set the SMS mode to text using AT+CMGF=1
    SendATCmdWaitResp("AT+CMGF=1", 2000, 50, "OK", 2);
//attaches the module to the GPRS Service using AT+CGATT=1
SendATCmdWaitResp("AT+CGATT=1", 10000, 30, "OK", 5);

//Defines PDP Context
SendATCmdWaitResp("AT+CGDCONT=1,"IP","internet.com"", 10000, 30, "OK", 5);

//Start Task & set APN, user ID, and password
SendATCmdWaitResp("AT+CSTT="internet.com", "Rogers", "wap", 10000, 30, "OK", 5);

void GSM::InitParam2()
{
    //Bring up wireless connection with GPRS
    SendATCmdWaitResp("AT+CIICR", 100000, 30, "OK", 5);

    //Get Local IP Address
    SendATCmdWaitResp("AT+CIFSR", 300, 30, "")
    //Gets Connection Status, should be IP STATUS before sending
    SendATCmdWaitResp("AT+CIPSTATUS", 10000, 30, "OK\r\nSTATE: IP STATUS", 5);

    //Sets GPRS parameters
    SendATCmdWaitResp("AT+SAPBR=3,1,"Contype","GPRS","", 10000, 50, "OK", 5);
    SendATCmdWaitResp("AT+SAPBR=3,1,"APN","internet.com"", 10000, 50, "OK", 5);
    SendATCmdWaitResp("AT+SAPBR=1,1", 10000, 50, "OK", 5);
}
A-4.1 InsertIntoTable

function insertIntoTables($myMac, $myProbe, $myTime) {
    // $myProbe = 1;
    $currTime = date( 'Y-m-d h:i:s', time() );
    // Get Offset for probe and add it to overall time
    $offsetQuery = "SELECT offset FROM probe_location WHERE probe_id='$myProbe'";
    $offsetResult = mysql_query( $offsetQuery ) or die (mysql_error());
    if( $data = mysql_fetch_array( $offsetResult ) )
    {
        $myTime += $data[0];
    }
    $testing = "";
    $searchForMacQuery="SELECT probe_id, discovery_timestamp FROM traffic_data WHERE device_mac='$myMac'";
    $result=mysql_query($searchForMacQuery) or die (mysql_error());
    $found = False;
    // If pair is found in database
    while( $row = mysql_fetch_assoc( $result ) )
    {
        // echo "Found match, removing from traffic_data"
        // echo $row['device_mac'] . " " . $row['discovery_timestamp'] . " " . $row['probe_id'] . "\n";
        $startProbe = 0;
        $startTime = 0;
        $startProbe = $row['probe_id'];
        $startTime = $row['discovery_timestamp'];
        echo 'Start: ' . $startProbe . '<br>';
        echo 'Curr: ' . $myProbe . '<br>';
        $testing .= "Start: '$startProbe' time: '$startTime' Finish: '$myProbe' time: '$myTime' =>";
        if( $startProbe != $myProbe )
        {
            $testing .= "Not Same,";
            // Check if valid path table
            $searchForPath = "SELECT path_id FROM probe_path WHERE probe_start='$startProbe' AND probe_end='$myProbe'";
            $result = mysql_query($searchForPath) or die (mysql_error());
            // Valid path found and startTime is less than the endtime, removing from traffic_data table, adding to probe_pairs table
if( $vPath == mysql_fetch_array( $result ) && $startTime < $myTime )
{
  $testing.= "Valid and Inserted";
  //Insert into probe_pairs database
  $insertPairsQuery = "INSERT INTO probe_pairs ( probe_id_start, probe_id_end, device_mac, timestamp_start, timestamp_end ) VALUES( '$startProbe', '$myProbe', '$myMac', '$startTime', '$myTime' )";
  $insert = mysql_query( $insertPairsQuery ) or die ( mysql_error() );

  //Remove from traffic_data database
  $removeMatchingQuery="DELETE FROM traffic_data WHERE device_mac='$myMac'";
  $remove = mysql_query( $removeMatchingQuery ) or die ( mysql_error() );

  echo "Inserted
";
  $found = True;
  break;
}
else
{
  $testing .= "Invalid Path";
}

if( $vPathReverse == mysql_fetch_array( $result ) && $startTime > $myTime )
{
  $testing .= "Reverse Path Valid and Inserted";
  //Insert into probe_pairs database
  $insertPairsQuery = "INSERT INTO probe_pairs ( probe_id_start, probe_id_end, device_mac, timestamp_start, timestamp_end ) VALUES( '$myProbe', '$startProbe', '$myMac', '$myTime', '$startTime' )";
  $insert = mysql_query( $insertPairsQuery ) or die ( mysql_error() );

  //Remove from traffic_data database
  $removeMatchingQuery="DELETE FROM traffic_data WHERE device_mac='$myMac'";
  $remove = mysql_query( $removeMatchingQuery ) or die ( mysql_error() );

  echo "Inserted
";
  $found = True;
  break;
}
else
{
  $testing .= "Invalid Path";
}
// else
// {
//   // Not a valid path, so add to traffic_table
//   $insertQuery="INSERT INTO traffic_data
(device_mac,discovery_timestamp,probe_id)
VALUES('$myMac','$myTime','$myProbe')";
//   $sql=mysql_query($insertQuery) or die( mysql_error() );
// }

else
{
    $testing .= "Mac & Probe already in Table";
    echo "Same mac with same probe already inserted\n";
    $found = True;
    break;
}

// Not found in database, therefore add
if( !$found )
{
    $testing .= "Adding New";
    echo "Adding to traffic_data\n";
    $insertQuery="INSERT INTO traffic_data
(device_mac,discovery_timestamp,probe_id)
VALUES('$myMac','$myTime','$myProbe')";
    $sql=mysql_query($insertQuery) or die( mysql_error() );
}

$testing .= "\r\n"

/* This section is for testing */
// $testSearchQuery = "SELECT * FROM test_table WHERE device_mac='$myMac' AND discovery_timestamp='$myTime'";
// $sqlTest = mysql_query($testSearchQuery) or die( mysql_error() );

// if( !$data = mysql_fetch_array( $sqlTest ) )
// {
//   $testInsertQuery = "INSERT INTO test_table
(device_mac,discovery_timestamp,probe_id)
VALUES(''$myMac',''$myTime',''$myProbe')";
//   $sql = mysql_query($testInsertQuery) or die( mysql_error() );
// }

return $testing;
}
A-5 pair_timeout_extended.php

A-5.1 Archive

function archive($currTime) {
    // Select all values of current data
    $archiveQuery = "SELECT * FROM current_data";
    $archiveResult = mysql_query($archiveQuery) or die ("Error getting all current values:
\n" . mysql_error() );

    while($row = mysql_fetch_array( $archiveResult ) )
    {
        $pathID = $row['path_id'];
        $timeTotal = $row['time'];
        $trafficCount = $row['trafficCount'];

        // Insert current data to archive
        $archiveInsert = "INSERT INTO archive_data (path_id, timestamp, time, trafficCount) VALUES ('$pathID', '$currTime', '$timeTotal', '$trafficCount')";
        $insert = mysql_query($archiveInsert) or die ("Error inserting to Archive:
\n" . mysql_error() );
    }

    $archivePulseQuery = "SELECT * FROM probe_pulse";
    $archivePulseResult = mysql_query($archivePulseQuery) or die ("Error getting all pulse values:
\n" . mysql_error() );

    while($archivePulseRow = mysql_fetch_array($archivePulseResult))
    {
        $pathID = $archivePulseRow['probe_id'];
        $pulseCount = $archivePulseRow['count'];

        // Insert current data to archive
        $archiveInsert = "INSERT INTO archive_pulse (probe_id, timestamp, count) VALUES ('$pathID', '$currTime', '$pulseCount')";
        $insert = mysql_query($archiveInsert) or die ("Error inserting to Archive:
\n" . mysql_error() );
    }

    // Clear the current data table after transfer to archive
    $myQuery = "UPDATE current_data SET time=0, trafficCount=0";
    $result = mysql_query($myQuery) or die ("Error clearing current_data:
\n" . mysql_error() );

    $myQuery = "UPDATE probe_pulse SET count=0";
    $result = mysql_query($myQuery) or die ("Error clearing current_data:
\n" . mysql_error() );

    Echo("Archived all current values successfully!
\n");
A-5.2 Pulses

function pulses($currTime) {
    // Finds all non-matched entries that have "expired" or timed out
    $pulseQuery = 'SELECT * FROM traffic_data WHERE time_inserted <= 
    ' . $currTime . ';
    $pulseResult = mysql_query($pulseQuery) or die ('"Error finding pulses older than five minutes:\n" . mysql_error());

    while($row = mysql_fetch_array( $pulseResult ) )
    {
        $myMac = $row['device_mac'];
        $probeID = $row['probe_id'];

        $dataQuery = 'SELECT * FROM probe_pulse WHERE probe_id=' . $probeID . ';
        $dataResult = mysql_query($dataQuery) or die ('"Error selecting current pulse data:\n" . mysql_error());

        $count = mysql_num_rows($dataResult);
        if($count == 1)
        {
            $dataRow = mysql_fetch_array( $dataResult );
            $newCount = $dataRow['count'];
            $updateCount = $newCount + 1;

            $updateLocations = 'UPDATE probe_pulse SET count=' . $updateCount . ' WHERE probe_id=' . $probeID . ';
            $result = mysql_query($updateLocations) or die ('"Error updating location tables:\n" . mysql_error());
        }
        else
        {
            $insertPulse = 'INSERT INTO probe_pulse (probe_id, count) VALUES (' . $probeID . ',1)';
            $result = mysql_query($insertPulse) or die ('"Error inserting new probe to database:\n" . mysql_error());
        }

        // Insert into traffic_pulse database
        // $updateQuery = 'UPDATE current_data SET time=0, trafficCount=' . $updateCount . ' WHERE path_id=' . $pathID . ';
        // $update = mysql_query($updateQuery) or die ('"Error updating current pulses:\n" . mysql_error());

        // Remove probe data from traffic_data database
        $removeMatchingQuery = 'DELETE FROM traffic_data WHERE device_mac=' . $myMac . ';
    }
}
$remove = mysql_query( $removeMatchingQuery ) or die
("Error deleting traffic pulse:\n" . mysql_error() );

Echo("Pulses handed successfully!\n<br>);

A-5.3 Pairs

function pairs($currTime)
{
    // Select every probe pair entry
    $selectPairs = "SELECT * FROM probe_pairs";
    $pairsResult=mysql_query($selectPairs) or die ("Error
selecting all probe pairs:\n" . mysql_error());
    $data = "";
    while($row = mysql_fetch_array( $pairsResult ) )
    {
        $probeStart = $row['probe_id_start'];
        $probeEnd = $row['probe_id_end'];
        $startTime = $row['timestamp_start'];
        $endTime = $row['timestamp_end'];
        $myID = $row['id'];

        $pathID = getPathID($probeStart, $probeEnd);
        if($pathID != -1)
        {
            // Find current corresponding path ID
            $dataQuery = "SELECT * FROM current_data WHERE
path_id='$pathID'";
            $dataResult=mysql_query($dataQuery) or die
("Error selecting current pairs data:\n" . mysql_error() );

            $dataRow = mysql_fetch_array( $dataResult );
            $newTime = $dataRow['time'];
            $newCount = $dataRow['trafficCount'];

            // Calculate difference in time in seconds
            $timeSec = $endTime - $startTime;

            $updateTime = $newTime + $timeSec;
            $pairsCount = $newCount + 1;

            // Update corresponding path ID
            $updateQuery = "UPDATE current_data SET
time='$updateTime', trafficCount='$pairsCount' WHERE
path_id='$pathID'";
            $update = mysql_query( $updateQuery ) or die
("Error updating current pairs:\n" . mysql_error() );
        }
        else
        {
            $startQuery = "SELECT * FROM probe_pulse WHERE
probe_id='$probeStart'";
        }
```
$startResult = mysql_query($startQuery) or die("Error selecting current pulse data:\r\n" . mysql_error());

$count = mysql_num_rows($startResult);

if($count == 1)
{
    $startRow = mysql_fetch_array( $startResult);

    $newCount = $startRow['count'];
    $updateCount = $newCount + 1;

    $updateLocations = "UPDATE probe_pulse SET count='$updateCount' WHERE probe_id='$probeStart'';
    $result=mysql_query($updateLocations) or die("Error updating location tables:\r\n" . mysql_error());
}
else
{
    $insertPulse = "INSERT INTO probe_pulse (probe_id, count) VALUES ('$probeStart',1)"
    $result = mysql_query($insertPulse) or die("Error inserting new probe to database:\r\n" . mysql_error());
}

$endQuery = "SELECT * FROM probe_pulse WHERE probe_id='$probeEnd'';
$endResult= mysql_query($endQuery) or die("Error selecting current pulse data:\r\n" . mysql_error());

$count = mysql_num_rows($endResult);

if($count == 1)
{
    $endRow = mysql_fetch_array( $endResult);

    $newCount = $endRow['count'];
    $updateCount = $newCount + 1;

    $updateLocations = "UPDATE probe_pulse SET count='$updateCount' WHERE probe_id='$probeEnd'';
    $result=mysql_query($updateLocations) or die("Error updating location tables:\r\n" . mysql_error());
}
else
{
    $insertPulse = "INSERT INTO probe_pulse (probe_id, count) VALUES ('$probeEnd',1)"
    $result = mysql_query($insertPulse) or die("Error inserting new probe to database:\r\n" . mysql_error());
}

//Remove probe data from traffic_data database
$removeMatchingQuery = "DELETE FROM probe_pairs WHERE id='$myID'";
$remove = mysql_query( $removeMatchingQuery ) or die ("Error removing probe pairs:\r\n" . mysql_error() );
}
file_put_contents('probepairs.txt', $data);
Echo("Probe pairs handeled successfully!\r\n <br>");
}
A-6 DynamicMap.php

```html
<meta name="viewport" content="initial-scale=1.0, user-scalable=no" />
<style type="text/css">
html { height: 100% }
body { height: 100%; margin: 0; padding: 0 }
#map_canvas { height: 60%; width: 60% }
</style>
<!--{mosmap mapnm='testmap'} -->

<script type="text/javascript">
  var map;
  var markers = [];
  var icons = [];
  var newTitle = 0;
  var data = [];
  var places = [];
  var directionsService = new google.maps.DirectionsService();
  var directionDisplay;

  function initialize()
  {
    directionsDisplay = new google.maps.DirectionsRenderer();
    var myLatlng = new google.maps.LatLng(49.823034, -97.13808);
    var myOptions = {
      zoom: 14,
      center: myLatlng,
      panControl: true,
      zoomControl: true,
      mapTypeId: google.maps.MapTypeById.ROADMAP,
      disableDefaultUI: true
    }

    map = new google.maps.Map(document.getElementById("map_canvas"), myOptions);
    directionsDisplay.setMap(map);
    loadMarkers(places);
  }

  function updateLocations()
  {
    var json = "[";
```
for(var i = 0; i < markers.length; i++) {
  pos = markers[i].getPosition();
  lat = pos.lat();
  lng = pos.lng();
  // data.push([markers[i].getTitle(), lat, lng]);
  json += '{ "probe": "' + markers[i].getTitle() + '", "lat": "' + lat + '", "lng": "' + lng + '"}';
  if (i < markers.length - 1) {
    json += ',';
  }
}
json += '];

var form = document.createElement("form");
form.setAttribute("method", "post");
form.setAttribute("action", "/DynamicMap.php");

form._submit_function_ = form.submit;

var hiddenField = document.createElement("input");
hiddenField.setAttribute("type", "hidden");
hiddenField.setAttribute("name", "LOCATION");
hiddenField.setAttribute("value", json);

form.appendChild(hiddenField);

function loadMarkers() {
  <?php
  $interLat = 49.823034;
  $interLon = -97.13808;
  $probeSearch = "SELECT * FROM probe_location";
  $probeData = mysql_query($probeSearch) or die ("Error retrieving probe locations: " . mysql_error());
  
  while($row = mysql_fetch_array($probeData)) {
    $probeID = $row['probe_id'];
    $probeLat = $row['probe_latitude'];
    $probeLon = $row['probe_longitude'];
    
    if($probeLat == 0)
      $probeLat = $interLat;
    if($probeLon == 0)
      $probeLon = $interLon;
  }
function placeMarker(spot_latlng, probeID) {
    icons.push(new StyledIcon(StyledIconTypes.MARKER, {color: "#ff0000", text: probeID.toString()}));

    var icon = icons[icons.length - 1];
    markers.push(new StyledMarker({styleIcon: icon, draggable: true, title: probeID.toString(), position: spot_latlng, map: map}));

    var marker = markers[markers.length - 1];
    addDropOption(probeID, probeID);
    google.maps.event.addListener(marker, 'drag', function() {
        var pos = marker.getPosition();
        var lat = pos.lat();
        var lon = pos.lng();
        resetMarkers()
    });
}
var lng = pos.lng();

document.getElementById("probedropdown").value=marker.getTitle();
document.getElementById("latbox").value=lat;
document.getElementById("lonbox").value=lng;
}

google.maps.event.addListener(marker, 'mousedown', function() {
    var pos = marker.getPosition();
    var lat = pos.lat();
    var lng = pos.lng();

document.getElementById("probedropdown").value=marker.getTitle();
document.getElementById("latbox").value=lat;
document.getElementById("lonbox").value=lng;

for(i=0;i<markers.length;i++)
{
    icons[i].set("color", "#FF0000");
    if(markers[i].getTitle() == marker.getTitle())
    {
        icons[i].set("color", "#00FF00");
    }
}
});

function newMarker()
{
    var spot_latlng;
    var location;

    if (navigator.geolocation)
    {
        navigator.geolocation.getCurrentPosition(markerAtLocation);
    }
    else
    {
        spot_latlng = new google.maps.LatLng(49.823034, -97.13808);
        placeMarker(spot_latlng, newTitle);
        newTitle = newTitle + 1;
    }
}

function markerAtLocation(position)
{
    var spot_latlng = new google.maps.LatLng(position.coords.latitude,
    position.coords.longitude);
    placeMarker(spot_latlng, newTitle);
function deleteProbe()
{
    var cont = false;
    var probeSelect = document.getElementById("probedropdown");
    var answer = confirm("Are you sure you want to delete probe: " + probeSelect.value + " and all related paths? Warning: This is permanent");

    if((probeSelect.value == "--" || probeSelect.value == 'NULL'))
    {
        alert("Please select a probe from the drop down menu.");
    }
    else
    {
        if(answer)
        {
            var json = "[{"probeID":" + probeSelect.value + "}]";
            var form = document.createElement("form");
            form.setAttribute("method", "post");
            form.setAttribute("action", "/DynamicMap.php");
            form._submit_function_ = form.submit;
            var hiddenField = document.createElement("input");
            hiddenField.setAttribute("type", "hidden");
            hiddenField.setAttribute("name", "DEL_PROBE");
            hiddenField.setAttribute("value", json);
            form.appendChild(hiddenField);
            document.body.appendChild(form);
            form._submit_function_();
        }
    }
}

function saveMarker()
{
    var id = document.getElementById("probedropdown").value;
    var lat = document.getElementById("latbox").value;
    var lng = document.getElementById("lonbox").value;

    var spot_latlng = new google.maps.LatLng(lat,lng);
    for(var i = 0; i < markers.length; i++)
    {
    }
if(markers[i].getTitle() == id){
    markers[i].setPosition(spot_latlng);
}
}

function saveRoute()
{
    var cont = false;

distanceInput = document.getElementById("distancebox");
startProbe = document.getElementById("startdropdown");
endProbe = document.getElementById("enddropdown");

if(distanceInput.value == ")
{
    calcRoute();
    if(distanceInput.value == "")
    {
        alert("Please select proper route.");
    }
    else
    {
        cont = true;
    }
}
else {cont = true;}
if(cont)
{
    var json = "[{"probeStart":" + startProbe.value + ","probeEnd":" + endProbe.value + ","distance":" + distanceInput.value + "}];"
var form = document.createElement("form");
form.setAttribute("method", "post");
form.setAttribute("action", "/DynamicMap.php");
form._submit_function_ = form.submit;
var hiddenField = document.createElement("input");
hiddenField.setAttribute("type", "hidden");
hiddenField.setAttribute("name", "PATH");
hiddenField.setAttribute("value", json);
form.appendChild(hiddenField);
document.body.appendChild(form);
form._submit_function_();
}

function calcRoute()
{
var start = document.getElementById("startdropdown").value;
var end = document.getElementById("enddropdown").value;
var distanceInput = document.getElementById("distancebox");
var pos;
var lat;
var lng;
var startText = "";
var endText = "";

for(var i = 0; i < markers.length; i++)
{
    if(markers[i].getTitle() == start){
        pos = markers[i].getPosition();
        lat = pos.lat();
        lng = pos.lng();
        startText = lat + "," + lng;
        // startText = markers[i].getPosition();
    } else if(markers[i].getTitle() == end){
        pos = markers[i].getPosition();
        lat = pos.lat();
        lng = pos.lng();
        endText = lat + "," + lng;
        // /endtext = markers[i].getPosition();
    }
}

if((startText != "")&&(endText != "")){
    var request = {
        origin:startText,
        destination:endText,
        travelMode: google.maps.DirectionsTravelMode.DRIVING
    };

    directionsService.route(request,
        function(response, status) {
            if (status == google.maps.DirectionsStatus.OK) {
                directionsDisplay.setDirections(response);
                distanceInput.value = response.routes[0].legs[0].distance.value;
            }
        })
    }
}

function addDropOption(OptText,Value)
{
    // Create an Option object
    var probeopt = document.createElement("option");
    var startopt = document.createElement("option");
    var endopt = document.createElement("option");
}
// Add an Option object to Drop Down/List Box

    document.getElementById("probedropdown").options.add(probeopt);
    // Assign text and value to Option object
    probeopt.text = OptText;
    probeopt.value = Value;

    // Add an Option object to Drop Down/List Box

    document.getElementById("startdropdown").options.add(startopt);
    // Assign text and value to Option object
    startopt.text = OptText;
    startopt.value = Value;

    // Add an Option object to Drop Down/List Box

    document.getElementById("enddropdown").options.add(endopt);
    // Assign text and value to Option object
    endopt.text = OptText;
    endopt.value = Value;
}

function fillTextBoxes()
{
    document.getElementById("latbox").value="--";
    document.getElementById("lonbox").value="--";
    for(i=0;i<markers.length;i++)
    {
        icons[i].set("color", ":00FF00");
        if(markers[i].getTitle() ==
            document.getElementById("probedropdown").value)
        {
            icons[i].set("color", ":FF0000");
            var pos = markers[i].getPosition();
            var lat = pos.lat();
            var lng = pos.lng();
            document.getElementById("latbox").value=lat;
            document.getElementById("lonbox").value=lng;
        }
    }
}

    google.maps.event.addDomListener(window, 'load', initialize);
</script>
A-7 RawData.php

```javascript
<script type="text/javascript" src="http://www.google.com/jsapi"></script>
<script type="text/javascript">
google.load('visualization', '1.1', {packages:
['corechart', 'controls']});
</script>

<script type="text/javascript">
function drawVisualization()
{
    var dashboard = new google.visualization.Dashboard(document.getElementById('dashboard'));

    var control = new google.visualization.ControlWrapper({
        'controlType': 'ChartRangeFilter',
        'containerId': 'control',
        'options': {
            // Filter by the date axis.
            'filterColumnIndex': 0,
            'ui': {
                'chartType': 'LineChart',
                'chartOptions': {
                    'chartArea': {'width': '90%'},
                    'hAxis': {'baselineColor': 'none'}
                },
                // Display a single series that shows the closing value of the stock.
                // Thus, this view has two columns: the date (axis) and the stock value (line series).
                'chartView': {
                    'columns': [0, 1]
                },
                // 1 day in milliseconds = 24 * 60 * 60 * 1000 = 86,400,000
                'minRangeSize': 3600000
            }
        },
        // Initial range: 2012-02-09 to 2012-03-20.
        // 'state': {'range': {'start': new Date(), 'end': new Date()}}
    });

    var chart = new google.visualization.ChartWrapper({
        'chartType': 'LineChart',
        'containerId': 'chart',
        'options': {
            // Use the same chart area width as the control for axis alignment.
        }
    });
```

73
'chartArea': {'height': '80%', 'width': '90%'},
'hAxis': {'slantedText': false},
'vAxis': {'viewWindow': {'min': 0}},
'legend': {'position': 'none'}
},

// Convert the first column from 'date' to 'string'.
'view': {
  'columns': [
    {//
      // 'calc': function(dataTable, rowIndex) {
      //   return
      //     dataTable.getFormattedValue(rowIndex, 0);
      //   },
      //   // 'type': 'string'
      // }, 1]
    0,1]
  }
}));
dashboard.bind(control, chart);

var dataTable = new google.visualization.DataTable();
dataTable.addColumn('datetime', 'Date');

var probe_id = document.getElementById( "probeView" ).value;

var probeResults = Array(5);
probeResults[0] = Array();
probeResults[1] = Array();
probeResults[2] = Array();
probeResults[3] = Array();
probeResults[4] = Array();

var timestamps = Array();
if ( probe_id == 0 )
  dataTable.addColumn('number', 'Probe 0 Count');
if ( probe_id == 1 )
  dataTable.addColumn('number', 'Probe 1 Count');
if ( probe_id == 2 )
  dataTable.addColumn('number', 'Probe 2 Count');
if ( probe_id == 4 )
  dataTable.addColumn('number', 'Probe 4 Count');
// dataTable.addColumn('number', 'Path ID');

// Create random stock values, just like it works in reality.
// var high;
// for (var day = 1; day < 121; ++day) {
//   high = day;
//   var date = new Date(2012, 0 ,day);
//   data.addRow([date,high]);
// }

74
<?php
$dataSearch = "SELECT * FROM test_table ORDER BY time_inserted ASC";
$archiveData = mysql_query($dataSearch) or die ("Error retrieving probe locations: ". mysql_error());
$time = "";

$probeCount = array( 0 => 0, 1 => 0, 2 => 0, 3 => 0, 4 => 0 );

while($row = mysql_fetch_array($archiveData))
{
    $probe_id = $row['probe_id'];
    $dataTime = $row['time_inserted'];
    if ( $time == $dataTime ) {
        $probeCount[$probe_id]++;
    }
    else if ( $time != "" ) {
        //echo "tvar date = new Date(" . strftime("%b %e, %Y %T", strtotime($dataTime)) . ");n";
        //echo "tvar date = new Date(" . strtotime($dataTime) . ");n";
        echo "tvar date = new Date(" . strftime("%Y, %m - 1, %e, %H, %M, %S", strtotime($dataTime)) . ");n";
        echo "timestamps.push( date );n"
        echo "probeResults[0].push( " . ($probeCount[0]) . ");n"
        echo "probeResults[1].push( " . ($probeCount[1]) . ");n"
        echo "probeResults[2].push( " . ($probeCount[2]) . ");n"
        echo "probeResults[3].push( " . ($probeCount[3]) . ");n"
        echo "probeResults[4].push( " . ($probeCount[4]) . ");n"
        $time = $dataTime;
        for( $i = 0; $i < 5; $i++ )
        {
            $probeCount[$i] = 0;
        }
    }
    else {
        $time = $dataTime;
        for( $i = 0; $i < 5; $i++ )
        {
            $probeCount[$i] = 0;
        }
    }
}

var length = timestamps.length;
for( var i = 0; i < length; i++ ) {
dataTable.addRow([timestamps[i], probeResults[probe_id][i]]);
}

dashboard.draw(dataTable);
dashboard.bind(control, chart);

google.setOnLoadCallback(drawVisualization);
</script>
Appendix B: ATmega328 Schematic

Figure B-0-1: ATmega328 Architecture [13]
## Appendix C: Database

Table C-1: Tables in Database

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>archive_data</td>
<td>Archive rows from current_data with time inserted</td>
</tr>
<tr>
<td>archive_pulse</td>
<td>Archive rows from probe_pulse with time inserted</td>
</tr>
<tr>
<td>current_data</td>
<td>The current traffic status of each path located in probe_path</td>
</tr>
<tr>
<td>probe_location</td>
<td>The GPS coordinates of each probe in the network as well as their respective offset</td>
</tr>
<tr>
<td>probe_pairs</td>
<td>The start and end probe ids and timestamps for each valid pair</td>
</tr>
<tr>
<td>probe_path</td>
<td>The start and end probe ids and the driving distance between them</td>
</tr>
<tr>
<td>probe_pulse</td>
<td>The current traffic count for invalid pairs found in traffic_data</td>
</tr>
<tr>
<td>traffic_data</td>
<td>Real-time raw data received from the master node</td>
</tr>
</tbody>
</table>
# Appendix D: Budget

Table D-1: Project Budget

<table>
<thead>
<tr>
<th>Unit Name</th>
<th>Number of Units</th>
<th>Cost per Unit</th>
<th>Total</th>
<th>Ordered</th>
<th>Delivered</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino Uno</td>
<td>1</td>
<td>$26.56 USD</td>
<td>$26.56 USD</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Xbee Shield</td>
<td>2</td>
<td>$18.70 CAD</td>
<td>$39.40 CAD</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Xbee Pro Module XBP24-AS1-001</td>
<td>7</td>
<td>$36.00 CAD</td>
<td>$252.00 CAD</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Bluetooth Module for Device Discovery</td>
<td>7</td>
<td>€75.00 EUR</td>
<td>€525.00 EUR</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>GBoard</td>
<td>1</td>
<td>$55.00 USD</td>
<td>$55.00 USD</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Foca USB Breakout Board</td>
<td>1</td>
<td>$7.50 USD</td>
<td>$7.50 USD</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Passive Components</td>
<td>N/A</td>
<td>N/A</td>
<td>$5.00 CAD</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Solar Panel</td>
<td>2</td>
<td>$25.00 USD</td>
<td>$50.00 USD</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Lithium-Ion Battery (4400 mAh)</td>
<td>2</td>
<td>$30.00 USD</td>
<td>$60.00 USD</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Lithium-Ion Battery (1300 mAh)</td>
<td>1</td>
<td>$12.00 USD</td>
<td>$12.00 USD</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Lithium-Ion Charger</td>
<td>2</td>
<td>$25.00 USD</td>
<td>$50.00 USD</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Lithium-Ion Charger</td>
<td>1</td>
<td>$12.50 USD</td>
<td>$12.50 USD</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Copper Clad Board</td>
<td>1</td>
<td>$13.50 CAD</td>
<td>$13.50 CAD</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ATmega328</td>
<td>1</td>
<td>$3.40 CAD</td>
<td>$3.40 CAD</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Casing</td>
<td>6</td>
<td>$8.00 CAD</td>
<td>$48.00 CAD</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------------</td>
<td>-------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subtotal</td>
<td>$1311.78 CAD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shipping costs</td>
<td>$125.00 CAD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>$1436.78 CAD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix E: Accepted Paper for the CCECE 2013

Vehicular Traffic Monitoring Using Bluetooth

M. Friesen, R. Jacob, P. Grestoni, T. Mailey and R.D. McLeod

Internet Innovation Center

Department of Electrical and Computer Engineering

University of Manitoba
ABSTRACT

The ubiquitous nature of Bluetooth equipped devices has made it opportunistic to scavenge information that can be repurposed for applications other than initially intended. One such opportunity is in vehicular traffic monitoring, whereby sampling of Bluetooth radios serve as proxies for vehicles and consequently density and flow. This paper discusses a complete system being developed at the University of Manitoba that utilizes a variety of wireless networking technologies and devices to collect inferred traffic data at an intersection along a major thoroughfare in Winnipeg.

Index Terms— Bluetooth Sensing, Traffic Monitoring, Congestion.

1. INTRODUCTION

Intelligent traffic systems promise to improve roadway congestion by capitalizing on information derived from traffic monitoring. In this paper a wireless sensor network is presented as a practical means of collecting a statistical representation of traffic density and flow.

The work reported here extends many of the uses initially suggested under the IntelliDrive initiative [1] that were primarily oriented towards improving mobility within surface transportation systems. However, it should be noted that the realization of advanced applications is not practical to deploy on scale without the momentum of the IntelliDrive like initiatives leveraging implementations based on web services. Web service realization for IntelliDrive applications is promising, since a uniform middleware can be achieved without extensive constraints on the underlying network and communication paradigms [2].

Early vehicular telematic applications were user-centric, whereas innovative applications that are now within reach concern statistics and crowdsourcing of auxiliary information. These newer class applications are cohort-centric rather than individual- or user-centric. These applications rely on inferencing from probes or floating cars, with the intent to capture the behaviours of a statistically significant portion of vehicles, such that meaningful inferences can be made and potentially generalized to the entire population of vehicles [3]. Applications include roadway infrastructure monitoring, as well as sampling empirical data input for traffic flow control. Being statistical in nature, in some cases, only a very small amount of floating car or probe data is required to infer significant events such as congestion build-up or dissolution [4]. Other probe based studies that suggest a stronger interplay with the network infrastructure include [5],[6] and [7].

The system described here is essentially a statistical crowdsourced monitoring of Bluetooth transceivers at or near a major intersection. Bluetooth was selected as the wireless sensing technology of choice as it Bluetooth is a telecommunications industry standard that defines how mobile phones, computers, personal digital assistants, car radios, GPS units, and other digital devices can be easily interconnected using short-range wireless communications. A particularly useful example of the use of this technology here is the interconnection of a mobile phone with a wireless earpiece for hands-free operation.

The basic system architecture presented here is that of a Bluetooth sensor network, interconnected with an XBee/802.15.4 middle tier, and a GSM backhaul tether to a web data collection and processing portal.

XBee Pro was selected as the middle tier wireless networking technology as it offers a low power solution with sufficient range for the interconnect between the Bluetooth sensors.

GSM was selected as the cellular tier as a means of aggregating and forwarding data collected to a web server for processing and display. The data sent to the central server is used to display the current traffic density and average velocity of vehicles at the intersection on both a Smartphone application and web front end. An illustration of the system deployed at a major intersection is presented in Figure 1.

The remainder of the paper is organized as follows: section 2 overviews the probe design, master node, power system and webserver. Section 3 provides a simple data collection demonstration and a motivational comparison with data collected mechanically. Section 4 concluded with a summary and acknowledgements.
2. MAJOR SYSTEM COMPONENTS

Major system components are presented next with particular emphasis on problems encountered and their resolution towards building a functioning prototype.

2.1. Probe Design

The objective of the probes is to collect vehicle trajectory information via Bluetooth device discovery and then transmit this information to the master node via the 802.15.4 protocol. It is desirable, but not necessary, to program the probes in such a way that the design is scalable. The probe design uses an Arduino Uno development board [8], which utilizes the ATMega328 microcontroller, an XBee module and a Bluetooth Pro module, both of which connect to the Arduino. An annotated probe device is illustrated in Figure 2.

![Figure 2: Probe Node](image)

Next the Bluetooth module had to be set for device discovery. The existing Bluetooth modules libraries would not meet our requirements as they could only detect 8 devices per inquiry. During traffic congestion periods it is desirable to discover all surrounding devices, as such it was decided to go with a Bluetooth module that could discover up to 250 unique devices per inquiry. This new module, however, was more expensive and required the development of a new library to control it. Initially there was some difficulty communicating with the module, but after inserting a short delay between successive character reads on the serial line, the problem was solved.

During the data collection process, the information is organized in a consistent manner to make it easy to decode; each device frame is 8 bytes. The device frame includes a marker byte, a 4 byte timestamp, and 3 bytes for the MAC address. To ensure privacy, only half of the MAC address is recorded which still provides 16 million unique combinations for one probe network. The first 3 bytes of the packet are reserved bytes; a control byte, a length byte, and a probe id byte. The maximum packet size that can be produced is 100 bytes, as this is the size of the receive buffer of the XBee module.

For our XBee network, the XBee module on the master was set as the network coordinator, and the XBee module on the probe was set to associate with a network coordinator. Probes communicate with the master using simple serial communication. In an attempt to save power, the XBee module on the probe was configured to hibernate when not in use. Putting the XBee into hibernation mode reduced the current draw from 160 mA to 100 mA, a reduction of 38%. This introduced a new problem: upon wake-up, the XBee module of the probe would accumulate random garbage characters in its receive buffer, which were automatically transmitted to the master node. As no solution to this was readily available, a work-around was implemented; prior to sending any proper packets, the probe would send “Jenny” to the master and anything the master received after “Jenny” would be processed. These types of workarounds are necessary when extending the capabilities of the probe and during the prototyping stage.

The last thing that was required for the probes is to synchronize their clocks with the master node. To do this, the probe was programmed to send the master 5 synchronization packets, including its current
timestamp, waiting for the master to acknowledge the packet before sending the next one. After all 5 synchronization packets have been received by the master node, the average round-trip-time, RTT, is calculated. An offset for that probe is created using the average RTT and the last timestamp received from the probe. This offset is added to any timestamp received from the probe when device discovery information is received.

The function of the probe is illustrated in the main flow diagram of Figure 3. The scan flow sub diagram is illustrated in Figure 4.

![Figure 3: Probe Flow Diagram](image)

![Figure 4: Bluetooth Scan Diagram](image)

Given that the prototype uses 6 probes, and to reduce cost, it was decided to design and build a custom PCB to interface the Bluetooth Pro modules. The reason for this is that the Bluetooth modules only require 4 pins to function, and they will be using the software serial; this will allow for the design modifications discussed earlier.

### 2.2. Master Node

The master node is the device that handles all of the incoming data from the probes and sends it via cellular GSM communication to the server. The requirements for this task are for the unit to be small, inexpensive, able to communicate via GSM and XBee, and have the ability to save data to an SD card. Using these requirements the GBoard was a good choice because it had everything required and it was Arduino based, making all nodes based on the same hardware family. An annotated master node is illustrated in Figure 5.

The next step was getting the individual components of the GBoard to work. For the GSM module, a preexisting library was used, but it needed to be modified to work correctly with the GBoard, as well as add specific functionality. For the former, the correct pin placements required were found by consulting the circuit schematic for the GBoard. The later involved rewriting most of the code in the library as well as reading the command sheet for the GSM module to find the appropriate commands for sending data to the server. This process required considerable time and testing. One main problem was connecting to the network using an MTS SIM card. After considerable testing, analyzing, and researching, it was initially thought that the SIM card did not work because the network did not recognize the GSM module. This was found to be incorrect as the SIM card was inserted in a phone and it connected to the network. After further research we found that the network that MTS uses is UMTS, which is different from the GSM technology. To test this, a Rogers SIM card was inserted into the GSM module and it was able to connect to the network as well as send text messages. These types of trials were unexpected and not well documented.

![Figure 5: Master Node](image)
software. By measuring the memory usage of each portion of code, we determined the cause of this behaviour to be a lack of memory. The GBoard only contains 2048 bytes of SRAM, 1800 of which are useable by us, while we were using over 2500 bytes of SRAM. Refactoring and lowering the memory usage of the code resulted in an integrated system that meets the stringent memory requirements.

As shown in the flow diagram of the Master node in Figure 6, data is sent to the central server every 5 minutes. This is done to ensure a continuous update of real-time information for the current state of the intersection.

![Figure 6: Master Node Flow Diagram](image)

### 2.3. Power Supply

The objective for the power supply was a small compact source that was solar rechargeable. Originally the desired run time for the power source was set at one week, but after running tests on the probe and determining its average current draw, the desired run time was reduced to 24 hours. Initially, a step-up converter was considered which caused a much higher current draw from the source than what was originally expected, requiring the purchase of larger batteries and a larger solar panel.

For developing the prototype the increased cost of a single power source led to the decision to have only one power source use a solar panel and the remaining units would run off of lead acid batteries that would need to be recharged after they ran out of power. This was done to demonstrate that a solar rechargeable source was a viable option.

For the rechargeable power source, there are two 4400 mAh, 3.7 V lithium ion batteries hooked up in parallel to a step-up converter that will step the voltage up to approximately 7 V. Both batteries are connected to lithium-ion chargers that will be powered by a 3.7 W, 6 V solar panel. A custom step-up converter was also designed to have lower current draw compared to the purchased step-up converter; thereby increasing the life expectancy.

The lead acid batteries were tested by connecting up a probe to them and measuring voltage and current. A probe requires variable amounts of power based on whether or not the Bluetooth or XBee devices are transmitting. This information was used to build a Multisim model for our solar rechargeable power source. For the switch in the step-up converter a 555 timer was used that output a square waveform to the gate of an nMOS transistor; turning it on and off. It was very difficult to create a working step-up converter because the resistance of the probe is variable depending on which devices are transmitting. After many hours of tweaking we were able to obtain a working model in Multisim and eventually a working prototype. We did tests on the step-up converter to determine how much current is drawn from the battery when the probe is transmitting; it is approximately 360 mA. This was done to give us a rough estimate of how long our power source would last without charging. The power supply unit is illustrated in Figure 4.

### 2.4. DataBase/Webserver

When the master node sends the information to the server, the data has to be stored in a way that it can be accessed and analyzed, as well as displayed to a user through the website or cell phone applications. The data must also be refreshed every five minutes such that the traffic information that is being viewed is always current; to provide this functionality a MySQL database was used.

Initial scripts were written where device MACs are compared to a table of existing MACs to find matching devices with a different probe location. When found, a probe-pair is created using the start probe, end probe and time; this can be used to find the velocity of the device. This code was updated and extended to handle the data that is sent from the master node. This was not straightforward due to the encoding done by the probes; individual bits had to be read to change bytes in the device information. Another PHP script was created that automatically runs every five minutes refreshing the traffic data that is displayed on the websites. This script goes through all of the data in the initial tables and creates the information required to be displayed by the websites. The data is stored as the total travel time for all vehicles and the total vehicle count through each probe-pair. With these two values the velocity can be found by a simple math function incorporating the distance between the probes. More detail is also available on the traffic count and average time spent at the intersection, without using space in the server to store these values.
Instead of creating different platform specific Smartphone apps, we decided to create a mobile version of the website, to act as the front end system that can be viewed on multiple platforms.

Figure 7: Google Map display of probe network

The website currently retrieves data from the database, which is used to create the information that is displayed to the user using PHP and HTML5. The website displays two different Google Maps that can be modified to display the traffic data. An example of displaying the probe network in a modifiable fashion is shown in Figure 7, where each moveable marker on the map symbolizes a probe at the intersection. These Markers also allow for actual driving distance between them to be calculated using Google’s software through their public maps API, simplifying the network setup.

3. VALIDATION

At this time unit testing is underway as well as basic integration of the major components. In terms of validating whether the methods will provide actual statistically meaningful traffic data a number of simple probe experiments were undertaken early on in the process. A stand alone probe was deployed at the Fort Garry Bridge under the following use case scenario.

Data Collection Process

Figure 7: Use case for Bluetooth Probe

Data by the probe device was collected over a 24 hour period and compared to data that was collected from a mechanical counter.

Figure 8: A simple Bluetooth probe and counter

Although there is considerable difference in the data collected compared to the mechanical counter the stylized overall shape is very encouraging. This simple probe experiment also prompted several design decisions for the more complete system. In particular, multiple probes were deemed necessary in order to differentiate traffic directionality as it is somewhat asymmetric during the various rush hours. Also
the need to scan and record more than 8 devices was evident from the volume of data anticipated. In addition to these experiments there are also other initiatives elsewhere with similar objectives in terms of monitoring traffic though capturing Bluetooth data [9]. The majority of these are currently being developed for estimating travel times as opposed to providing density and flow information. A complete test of the system its and deployment as illustrated in Figure 1 is planned for February 2013 and results will presented during the formal proceedings of CCECE 2113.

4. SUMMARY

This paper presented a functional prototype of a cost effective Bluetooth traffic monitoring system. The system combines several wireless networking technologies capitalizing on their respective capabilities. The lowest level includes the Bluetooth scanner to detect a statistical representation of proximate devices. The middle tier utilizes XBee/802.15.4 well suited for extended range coverage between probes and a master node. The master node aggregates data and forwards data to a web server/database for processing. A number of design trade-offs were discussed including the need for enhanced scan capabilities. The packetization and memory constraints also provided considerable challenges as well as the power consumption of the prototype system. Advantages of the power system design is that the system can be easily deployed for up to a week allowing a city’s transportation or public works department to evaluate the technology without incurring any significant costs to a trial.

5. REFERENCES


Acknowledgement

The authors would like to thank the City of Winnipeg Department of Public Works for financial support, and in particular Mr. Doug Hurl and Mr. Luis Escobar. In addition, the financial support of NSERC is gratefully acknowledged.