THE UNIVERSITY OF WESTERN AUSTRALIA

REVSki Data Transmission, Storage, and Visualisation

Dylan Leong, 21485566

Supervisor:

Thomas Bräunl

School of Electrical, Electronic, and Computer Engineering

Word Count: 7545 (7200 excluding appendix)

2019

Abstract

The Renewable Energy Vehicle Jet ski (REVSki), an experimental electric personal watercraft, has undergone many changes throughout several years and it is expected that the vehicle will continue being upgraded in the future. In order to assist with ensuring the normal operation of the REVSki continues even throughout these changes, this thesis is concerned with the real-time tracking and logging of the vehicle when it is in use.

To be more specific, the project is concerned with the wireless transmission of certain parameters such as location, motor speed, voltage, and current of the REVSki via a modem to a database, as well as providing a webpage to view these parameters.

As the vehicle is also being augmented with a water sensor, information about the water will be made available on the webpage, visualised in the form of a satellite-view map.

Acknowledgements

I would like to acknowledge Professor Thomas Bräunl, the supervisor of this project, as well as Marcus Pham, Ivan Neubronner and Kai Li Lim for their guidance throughout the past year.

I would also like to thank Marti Leven, Ze Lin, and Xiaoqing Ran, who also worked on the REVSki during the same timeframe, along with the SPAB team, Wesley Coleman, Morgan Trench, and Aaron Goldsworthy for sharing their knowledge.

Lastly, my family, as I would not have gotten far in life without them.

Contents

Table of Figures

Table of Tables

Table of Listings

Abbreviations and Definitions

Abbreviation	Definition
AT	ATtention
CAN	Controller Area Network
EVMS	Electric Vehicle Management System
EWIS	Electrical Watercraft Instrumentation System
GPS	Global Positioning System
HTTP	Hypertext Transfer Protocol
MySQL	My Structured Query Language
NMEA	National Marine Electronics Association
OSM	OpenStreetMap
PCB	Printed Circuit Board
PTDM	Pass Through Data Mode
RAM	Random-Access Memory
REV	Renewable Energy Vehicles
RS232	Recommended Standard 232
SDRAM	Synchronous Dynamic Random-Access Memory
SRAM	Static Random-Access Memory
TCP	Transmission Control Protocol
TTL	Transistor-Transistor Logic

1 Introduction

1.1 Background

The REV Project, run within UWA, involves the development and demonstration of a wide array of electric vehicles, with the aim of acting as a proof of concept for the use of renewable energy in transportation.

Under the REV project is the REVSki, a former petrol jet ski that has been retrofitted to become an electric battery-powered jet ski. As of 2019, the REVSki is able to reach a little over 40km/h (with a single passenger, as adding a second passenger lowers speed) and runs for about 20 minutes [1]. While it may not have reached the explosive power of its former petrol self, the REVSki is still being upgraded, and more importantly, currently functional, which is a step towards success in its conversion to electric.

1.2 Problem Statement

In 2018, work on the REVSki had a focus on improving the instrumentation of the vehicle, implementing a monitoring system, and installing some external sensors for collecting data about the water. While this thesis will primarily be covering the monitoring of the REVSki, it is important to note that some of the parameters required for monitoring come from the instrumentation (such as the EVMS) and external sensors.

To be more specific, there are two main reasons behind the implementation of a monitoring system:

1. To log some data during a drive and be able to look back on it.
2. To offer live tracking of the vehicle, which is useful for verifying the REVSki's capabilities while it is being driven.

In short, a monitoring system can be thought of as an auxiliary to all future changes to the REVSki. If an individual makes upgrades to the REVSki and wishes to check the effectiveness of said upgrades, they can make use of the monitoring system. Likewise, it can also be used for troubleshooting in case some changes go wrong, or as a tool to compare the state (and capabilities) of the jet ski to how it was in the past.

1.3 Scope

In terms of the data itself, it is desirable that the monitoring system can cover:

1. Tracking of the REVSki (latitude and longitude)
2. Status of the motor and batteries
3. Input from external sensors
4. The time of when data was taken

On the other hand, the monitoring system itself comes in three parts. Firstly, there needs to be a way to view and visualise data. Secondly, it is necessary to have some way to store the REVSki's data. Thirdly, some module needs to be installed onto the REVSki to transmit data to where it needs to be stored.

The first and second parts are done through the REV Project's web server, which hosts a MySQL database and website which handle the storage and visualisation of data respectively. The third part, transmission, is slightly more complicated.

Normally, it would be simple to attach a modem to the REVSki and establish a direct TCP connection to some server. The modem would transmit data periodically over the TCP connection, while the server would have a daemon running constantly to parse and store the data. Unfortunately, the REV Project's web server does not have any unused and open TCP ports. Likewise, UWA has servers but will not open ports due to security reasons. As such, the method of transmitting data via TCP to an **unused** open port is not viable for the REVSki, creating the need for an alternative solution.

2 Literature Review and Relevant Previous Work

2.1 REVSki Instrumentation

Maximillian Woloszyn, a student who previously worked on the REVSki, completed a thesis in mid-2018 involving the instrumentation of the vehicle [2]. The most relevant aspect of his project with respect to this thesis is the storage of data on an SD card. While this method fulfils the first purpose of monitoring, which is to be able to look back and reflect on a previous drive test, it is a few steps away from live tracking. Additionally, Woloszyn's logging lacked data on the battery cells, which would be added in the scope of this thesis.

That being said, Woloszyn's existing work provided a good framework for this project to build upon. Woloszyn had created an EWIS box for the REVSki, which contained an Arduino Mega, CAN bus shield, and GPS shield. The Arduino was used to update the dashboard and store parameters in an SD card. For this thesis, the EWIS box would be reused, upgraded to take more inputs from various sensors and instrumentation, and used to house an additional module that would be responsible for the transmission of data from the REVSki to a database.

2.2 Remote Control of Autonomous Surface Vessels

Aaron Goldsworthy did not previously work on the REVSki, but completed a thesis on a project that is also under the supervision of Thomas Bräunl called the Solar Powered Autonomous Boat (SPAB) [3]. One of his tasks was to find a way to transmit data from the boat to a database under the same restrictions outlined in this project. Namely, without the access of a server with an unused open TCP port. Goldsworthy's solution was to make use of how all web servers need to have port 80 open for the Hypertext Transfer Protocol (HTTP) [4], commonly known as the protocol in which messages are sent across the world wide web. Through port 80, the SPAB's modem would send messages conforming to the HTTP format, which are parsed on the web server side without the need for a daemon.

While Goldsworthy used different hardware for his project and did not go into the specifics of how successful his method was, his solution of using HTTP for data transfer is still applicable for this thesis and is the foundation behind the monitoring system's transmission.

3 Transmission

To reiterate, in order to transmit data from the REVSki to the REV Project web server's MySQL database, HTTP requests are used. Sending these requests require some sort of modem,

which in turn require some way of passing in the necessary data to be transmitted. There is also the question of exactly what data needs to be sent, and where said data can be sourced from.

3.1 Hardware Setup

3.1.1 Previous EWIS

Building on Woloszyn's previous work, the EWIS box was reused for the purpose of transmitting data. The Arduino Mega was kept and repurposed from saving to a SD card to instead sending data to a modem for transmission. The CAN Bus shield, previously used solely for updating the dashboard, was kept for the purpose of interfacing the Mega with any other modules in the REVSki via the CAN protocol to take data from.

On the other hand, the GPS shield was removed, due to the acquisition of an alternative.

3.1.2 Astra AT240

Figure 1: Astra AT240

The Astra AT240, shown above in Figure 1 is a module that functions as both a GPS and a 3G modem. Its addition to the EWIS box made the existing GPS shield obsolete due to redundancy. Due to its function as a modem, the Astra was initially the key hardware in making transmission possible.

The reason why a 3G modem was initially chosen over a 4G modem was because the Astra was a zero-cost option. It was formerly unused and owned by the REV Project before its installation into the REVSki. It also uses a sim card that was granted to the REV Project which has no running costs to keep it from expiring, thus saving the project the cost of purchasing both a 4G modem and a 4G sim card.

A characteristic of the Astra is that it has two RS232 ports [5] which are required to interface with it. The first port outputs GPS data and takes in configuration commands as an input. The second port is unused by default and needs to be assigned a purpose. For the sake of this project, the Astra is set into pass through data mode (PTDM) by passing config inputs into the first RS232 port, which then allows the use of the second port as a 'TCP forwarder'. To be more specific, once the Astra is assigned an IP address and port to connect to, it will attempt to make a TCP connection. Once the connection is successful, any messages received from the TCP connection will be readable from the second RS232 port as an output [6]. Additionally, any messages sent to the RS232 port as an input will be 'forwarded' to the modem and sent through the TCP connection.

In PTDM mode, the Astra allows a user to set several parameters that trigger transmission. These parameters are:

1. Packet max size, which triggers the modem to transmit when the Astra has been sent enough data to fill a packet of user-defined max size.
2. Packet timeout, which sets the modem to send even if the max size is not reached until some timeout has been reached.
3. Upon receiving a carriage return and newline.

The packet max size and timeout were set to be quite low in order to ensure that the Astra's modem would send packets as often as possible. The third parameter was unreliable due to the nature of HTTP requests, where messages could get quite long before the end of a line (indicated by a carriage return and newline, which would then try to trigger transmission of a very long message).

Due to the Astra having two RS232 ports, two RS232 to TTL converters were required. The solution to this is a PCB daughterboard that sits on the side of the Mega not occupied by the CAN Bus shield. The daughterboard contains a MAX3232 chip which covers the conversion for both RS232 ports. Note that the PCB has other functionality, such as a connection to the REVSki's external sensors. For more information, refer to the thesis of its designer, Ze Lin.

3.1.3 CAN network and EVMS

In order to obtain data about the batteries, a CAN network was established to the newly acquired EVMS, installed by Marti Leven [7]. The EVMS was purchased from a company named ZEVA, who defined its CAN protocol as a series of CAN IDs where various bits containing information about the battery voltage, amp hours, individual cells, etc, are broadcasted to. On the software side, this system is taken advantage of by defining a mask and filter using the CAN Bus shield library functions to limit incoming CAN messages to only the IDs with information that is useful for monitoring.

3.1.4 Assembled

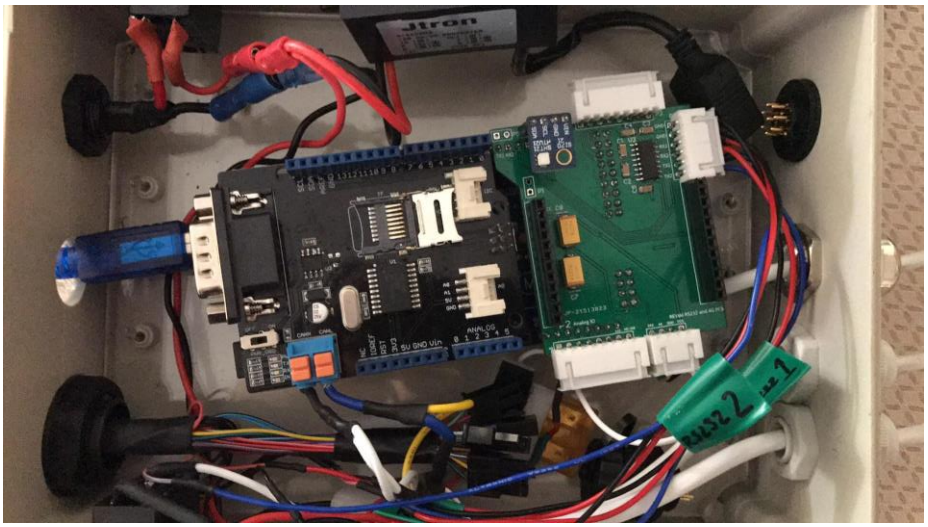

Figure 2: Assembled EWIS Box

After completing assembly, as shown above in Figure 2, a few more additions were added to the EWIS box. IP68 glands and panel mounts were installed where necessary to waterproof the box, save for one hole that would house the USB-C cable to flash to the Mega and would only be covered up with a gland when the REVSki was being taken to the water. A few blade fuses were added to protect the Mega and its auxiliaries against current shorting. Lastly, a DCDC with a 5V output was placed in the box to be used instead of the Arduino Mega's on-board voltage regulator, which would become hot when used. This is likely due to the Mega's recommended max input voltage being 12V [8], with the REVSki's auxiliary output voltage being closer to 14V. While there was no guarantee that a 14V input would damage the Mega, the DCDC was installed anyway as if anything went wrong, replacing it was much cheaper than the cost of replacing an entire Arduino Mega.

Table 1 below details the data taken for monitoring along with its source:

Table 1: Names and Sources of Data to be Stored and Transmitted

Data	Source
Timestamp	GPS
Latitude	GPS
Longitude	GPS
Speed	GPS
Battery Voltage	EVMS
Battery Current	EVMS
Water Temperature	External Temperature Sensor
Water Salinity	External Water Salinity Sensor

Individual Cell Voltages	EVMS

3.2 Software setup

3.2.1 Arduino

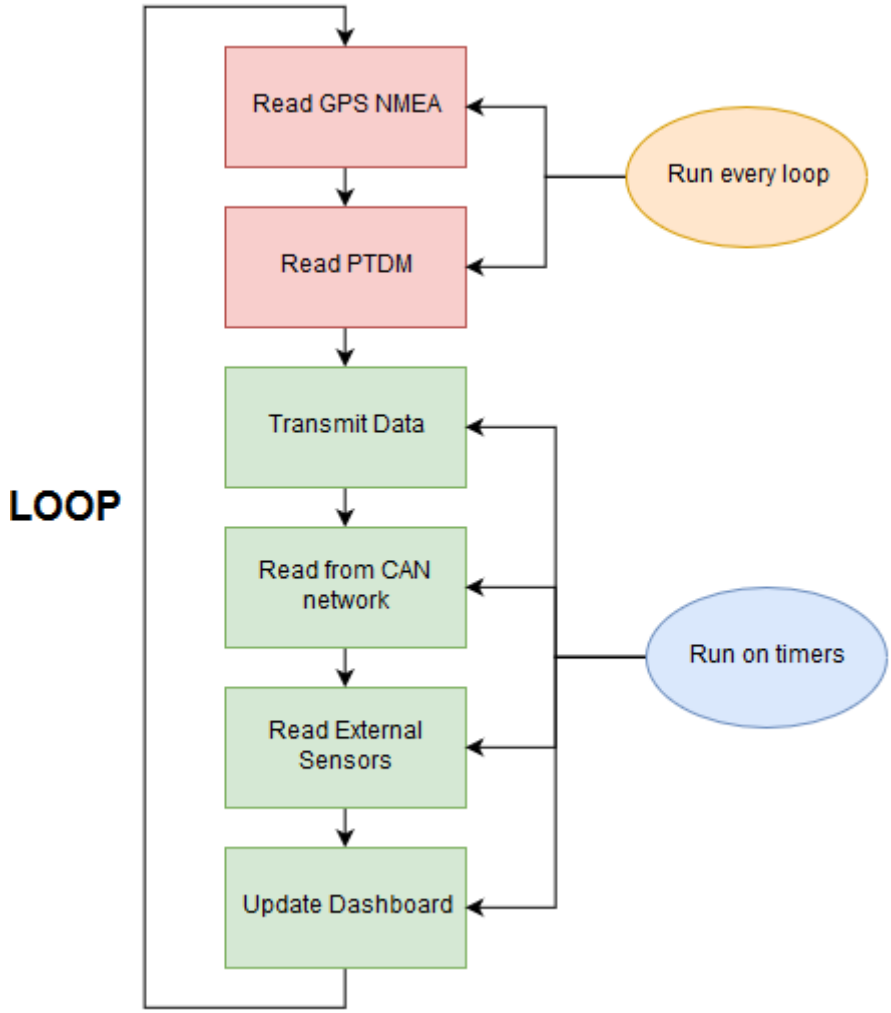

Figure 3: Diagram of 3G Software Loop

Alongside the hardware assembly, the Mega's software was upgraded to work in its new environment. Due to Arduino's lacking threading or any kind of multitasking outside external interrupts, the software was split up into different tasks or 'modules', as seen above in Figure 3, and given timers that would ensure that a task would only run in the Arduino's loop if a certain amount of time had passed since its last completion.

The modules reading from the GPS and PTDM RS232 outputs (both from the Astra) are notable in the sense that neither have a timer. This is due to the existence of the serial buffer, where messages sent via serial enter a buffer until they are read. If the buffer is full, it drops messages and they are lost forever. Incomplete or broken GPS messages can result in them being parsed incorrectly, leading to the wrong values for timestamp, latitude, longitude or

speed. PTDM messages (received from the other side of the TCP connection) are required for the Mega to know whether its HTTP request has been sent successfully or not, so losing either GPS or PTDM messages is highly undesirable. As such, the GPS and PTDM are checked and read from every loop. Additionally, it is important that at no point in the Arduino's sketch is the delay() function used, as it blocks the entire microcontroller from taking any other action which can lead to lost serial messages.

The module in charge of transmission first packs all the data inside a message that follows the HTTP format. Due to the size bloating caused by HTTP headers, the message can become very large. As packing involves a long series of concatenation, the Arduino code uses a character array over the string class. This is because of repeated string concatenation causing memory fragmentation, which is usually not a concern in modern day computers but is unfortunately relevant for an Arduino Mega which only has 8KB of SRAM.

It is also worth noting that if the GPS has yet to fix its location, the Astra returns invalid values for its GPS output such as '0-0-0000' for time stamp and values above 360 for latitude and longitude. As such, to prevent the Rev Project's web server from logging invalid data (and then trying to visualise it on a graph or map), an HTTP request is not sent if the Mega detects that the GPS is still fixing its location.

The last module handles all non-GPS data acquisition by checking and reading input from the external sensors and EVMS. For detailed information about the external sensors, refer to Xiaoqing Ran's thesis. The EVMS, as mentioned in the previous section, is connected to the Mega's CAN Bus shield via a CAN network and reports data in the form of an 8-byte package. For example, Table 2 below displays the format of one of the relevant CAN messages that contains information about the batteries and EVMS status. A full list of the CAN byte tables is available in ZEVA's EVMS manual [9]. The software takes and bit shifts the necessary bytes for conversion to integers, so data is ready to send for the next time the transmission module runs.

Table 2: Example of EVMS CAN Protocol

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	Error code					Status code		
Byte 1	Battery amp-hours remaining, 0.1Ah resolution, high byte							
Byte 2	Battery amp-hours remaining, 0.1Ah resolution, low byte							
Byte 3	Battery voltage, 0.1V resolution, high byte							
Byte 4	Battery voltage, 0.1V resolution, low byte							
Byte 5	Auxiliary voltage, 0.1V resolution							
Byte 6	Headlights	Isolation integrity (0-100%)						
Byte 7	Temperature (°C+40)							

3.2.2 Web Server

On the web server side, software is used to handle the receiving of data from the EWIS box which transmits HTTP requests in the POST format, allowing the message to contain a 'body',

or in the case of this project, all the data. An example request can be found below in Listing 1.

```
POST /revskitrack/http_recv.php HTTP/1.1
Host: therevproject.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 41

field1=value1&field2=value2&field3=value3
```

Listing 1: Example HTTP Post Request

A placeholder body has been used in the above example as the real system has many fields causing the length of the body to approach 500 characters. In order to parse this message, the web server hosts a PHP file which makes use of the $_POST super global, which is initialised automatically upon receiving an HTTP POST request. The software then pushes the data to a database for storage and logs its activity into a file for debugging, which is available for viewing on the REV Project website [10].

3.3 3G Methodology

Two methodologies were used with this setup. The first involved limiting transmission to only occur if the PTDM serial had indicated that the last message sent had been received by the webserver. The idea behind this was to stop the Mega from flooding the Astra with messages faster than it could send them. Using this method, the success rate of transmission was consistent and close to being guaranteed, but had worst case send times of **up to one minute**. Due to the scope and purpose of the monitoring system, this was unacceptable for a few reasons:

1. In the worst case, the time ‘gaps’ between transmissions was too large. One minute is a long time, especially for a vehicle, so having the system be this slow meant that a lot of data was missed out on.
2. It was a hindrance in live tracking, as needing to wait for a minute for data to arrive meant that the visualisation was not ‘live’ anymore.

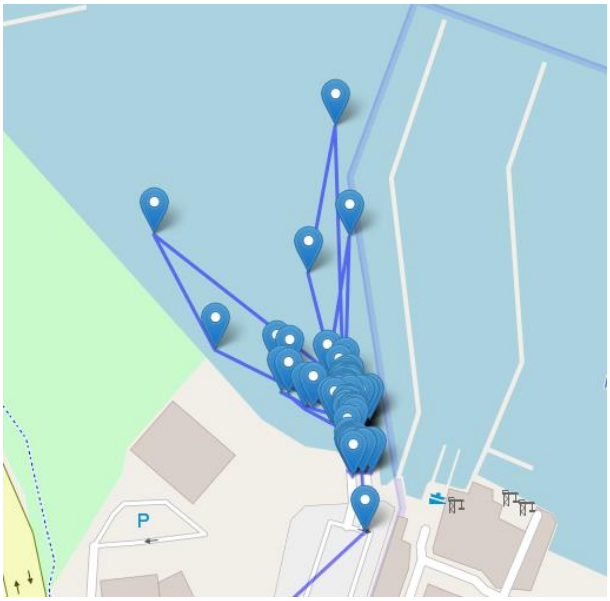

Figure 4: The Consequence of Long Transmission Times

To illustrate this point, the above Figure 4 displays a drive taken under this methodology. As seen, the markers are very far apart and more concentrated on the jetty rather than in the water, as the REVSki is traveling fast enough to make the Astra's slow transmissions very spread out. A lot of potential data was lost due to the speed of the Astra's modem, so a method that could send transmissions more frequently was desired.

The problems with the previous method led to an attempt being made at removing the restriction on transmission that required the software to wait for the PTDM to confirm that the last message sent had arrived at the web server. As such, the new method would attempt to send messages at regular intervals regardless of the status of previous message. The reasoning behind this was to queue up messages every few seconds (initially 20), so even if transmission was slower, there would still be enough data to look back on.

However, the method would only work if the Astra had a very large serial buffer, due to the possibility of the modem not being able to send messages quick enough. The result was a less consistent stream of data that had pockets of messages with 20 seconds between each other, followed by large gaps of (supposedly) failed transmissions. The success rate dropped to ~21%, and the logs of testing this can be viewed in Appendix A. With regards to the intent of the monitoring system, this method was not much better than the first, as it introduced the possible scenario of losing a lot of data transmissions. Additionally, it was not any better at live tracking than the previous method.

3.4 3G Evaluation

Ultimately, the Astra's two methodologies became a question of "consistent success rate with long transmission times" against "low success rate with occasional bursts of repeated successful short transmissions". Neither were very ideal due to being unreliable for both tracking and logging. The idea of saving multiple 'time slots' of data every 5-10 seconds onto the Mega before sending them all at once every minute or so was entertained but dismissed

due to the Mega's low amount of SRAM and very large message sizes which slowed the Astra's modem down even further.

As such, it was concluded that using the Astra would never meet the scope set out for the monitoring system, and that a better solution could only be reached by replacing the hardware.

3.5 4G Alternative

Due to the aforementioned problems with the Astra, it was theorized that the faster and more recent technology behind 4G could offer a speed advantage over the currently used 3G. As such, a cheap 4G modem was acquired in the hope of testing whether transmissions would be sped up enough to be more suitable for the monitoring system. The modem purchased was a Luat 4G 720H, shown below in Figure 5, and after some modifications to the existing PCB daughterboard, the modem was placed to sit comfortably on its surface, connecting to the Mega via TTL.

Figure 5: The Luat 4G 720H

3.5.1 Setup and Methodology Differences

In terms of software, the Luat is drastically different to the Astra. The Astra attempts to make a TCP connection when it starts up and will forward anything sent into its PTDM RS232 through the connection. There is no way to manually control the connection other than forcibly switching the Astra off.

The Luat is controlled by AT commands which give much more control over the state of the connection, making the software into something of a state machine. A diagram describing the state machine is available below at Figure 6. After start-up, the transmission process is carried out like such:

1. Attempt to start the TCP connection to a designated IP address and port.
2. Wait for the previous command to return an OK response.
3. Send a command to the Luat indicating the Mega is ready to send a message.
4. Wait for the Luat to respond that it is ready.
5. Pass in the HTTP request to the Luat.

6. Wait for a response indicating that the request has been received by the web server.
7. Forcibly shut the TCP connection and loop the process.

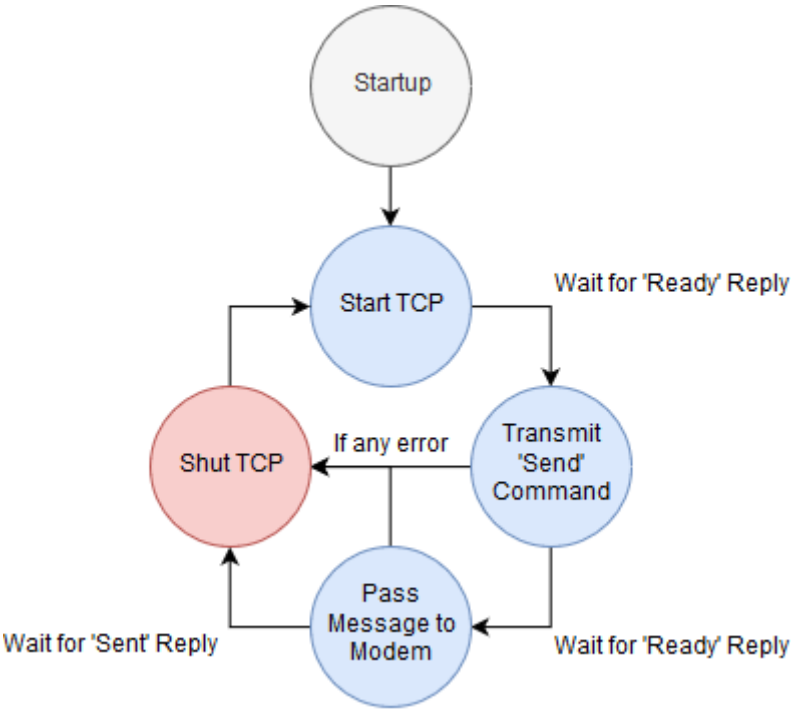

Figure 6: Diagram of the Luat's State Machine

Another difference is that the Luat needs to know the total length of the message it is sending which needs to be passed as in input in step 3. Previously, for the Astra, only the length of the body was required for the HTTP header 'Content-Length', requiring the creation of a character array to count its length. Now, another character array is required to check the length of the whole message, which further dips into the memory budget of the Mega.

3.6 4G Evaluation

Performance wise, the Luat can transmit data at a worst case of about 0.6 seconds. On average, transmission takes less than half a second. Testing results for the Luat are available for viewing in Appendix B. This is much more in line with the scope of the monitoring system, as data sent at this speed allows for both live tracking and more detailed logging.

3.7 Modem Comparison

Despite initially seeming to be much better than the previously used Astra, a comparison was necessary to ensure that the Luat could be used in the EWIS as a suitable replacement.

3.7.1 Control

Due to its extensive command list, the Luat offers much more control. At any point in its state machine, if something goes wrong or a command sent into the Luat returns a negative response, the TCP connection can be forcibly shut off, resetting the connection and allowing

for a fresh start for the next loop. The Astra, on the other hand, does not allow the Mega to control its TCP connection in any way.

3.7.2 Speed

In terms of speed, the Luat is drastically faster than the Astra, with their worst cases being ~0.6s and a minute respectively. Of the two, only the Luat is fast enough to meet the scope of the monitoring system, while the Astra is too slow.

3.7.3 Memory

Due to the Luat requiring the length of the entire HTTP request as an input during its send command, it forces the software to take up more memory on the Mega. While this is not an issue now, it may become a problem in future projects if it is decided that more data needs to be transmitted from the REVSki.

3.7.4 Cost

The Astra, which has a 3G modem, uses a sim card that was acquired by the REV Project that has no upkeep whatsoever. On the other hand, the Luat requires a 4G sim card, which tend to have an expiry date on their data usage if not renewed within a certain time. For example, the sim card currently in the Luat allows for 500MB of data usage but expires in 21 days. Cost-wise, unless telecommunications companies begin offering sim card options with a long expiry (or no expiry at all), using the Luat will be much more expensive in terms of upkeep cost.

3.7.5 Compatibility

Lastly, the Astra has no known compatibility issues with the rest of the EWIS box components. This is unfortunately not the case with the Luat, which has issues with the REVSki dashboard. Running software on the Mega that involves both modules will result in the Luat start-up stopping the dashboard from displaying anything, before crashing and restarting (which again stops the dashboard).

3.7.6 Comparison Conclusion

Sadly, despite all the advantages of the Luat, not having a dashboard is a safety issue that cannot be ignored. The Astra not having the same problem does not make it any more viable, as it is not usable either due to failing to have the performance for a monitoring system. For solutions to the transmission issues, refer to the recommendations for future work section.

4 Storage

The storage of data is handled by a MySQL database hosted by the REV Project's web server. The database splits data up into tables based on where each category of data is sourced from, as shown below in Figure 7. Each table uses a timestamp as its primary key to allow for MySQL queries to easily join tables together to view data from multiple sources at once.

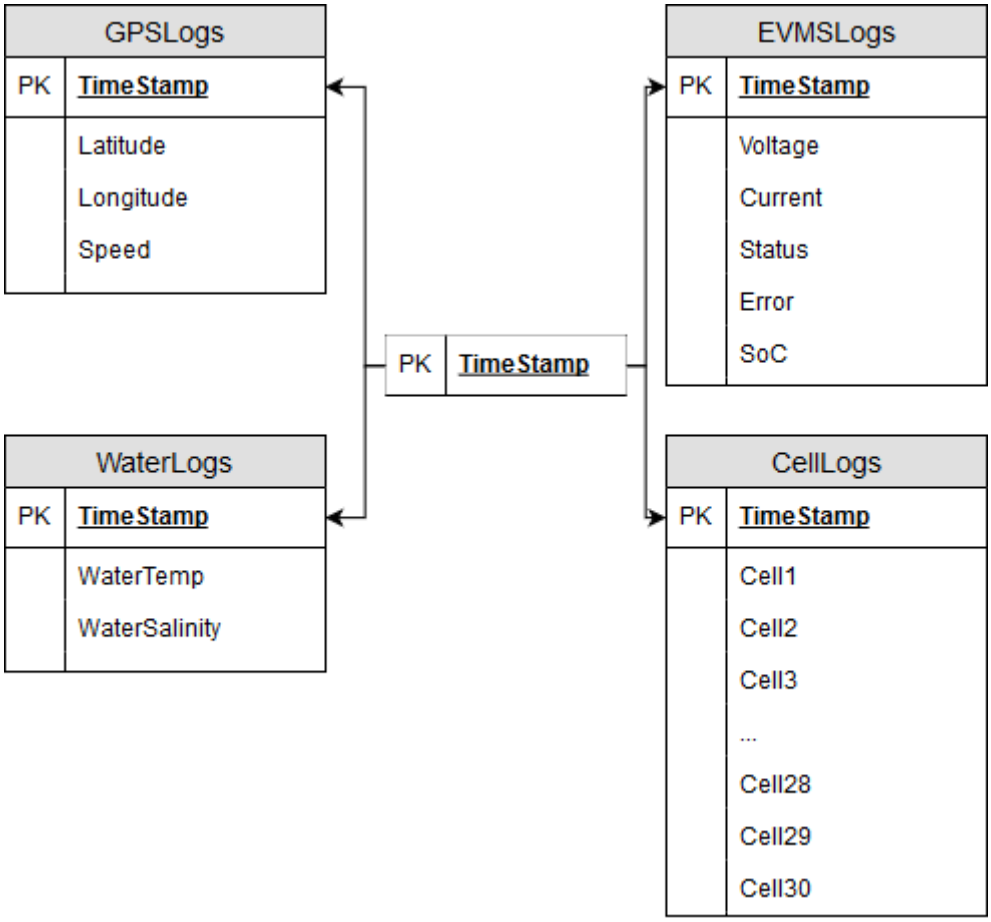

Figure 7: Diagram of Database Tables

Organizing data into tables by sources rather than just using a single table was done in order to ensure that future changes to the database would be as painless as possible. It is much easier to add a new table with extra data over adding columns to an existing table that already contains entries.

5 Visualisation

The visualisation aspect of the monitoring system takes the form of a website named REVSkiTrack hosted on the REV project's web server [11], written in PHP and JavaScript. jQuery and AJAX are also used to allow the website to update information on the web page without refreshing. The website contains a map that marks the location of where the REVSki has been while transmitting data, a graph that draws plots of the data against time, as well as a datetime selector that restricts the website to only visualising data between two dates.

Despite the completeness of Google Maps, it was decided that OpenStreetMap (OSM) would be used instead. This is because of Google's recent decision to change Google Maps to a paid service [12] which would introduce yet another upkeep cost into the REV Project's web hosting. OSM is an open-source and free alternative that may not have the detail and features of Google Maps, but is still functional enough for this iteration of REVSkiTrack. The main two disadvantages of OSM are the less frequent updates it receives to map changes, and its lack of ability to zoom in as much as Google Maps.

To assist with analysing logged data, a heatmap functionality has been added to REVSkiTrack's map, as shown below in Figure 8. A user can choose what parameter to have the heatmap 'focus' on (for example, speed, voltage, etc) and the routes between markers will be coloured accordingly. A legend is automatically generated at the bottom right that defines the number ranges of each colour. Markers can also be clicked on to display all the data about a point in time.

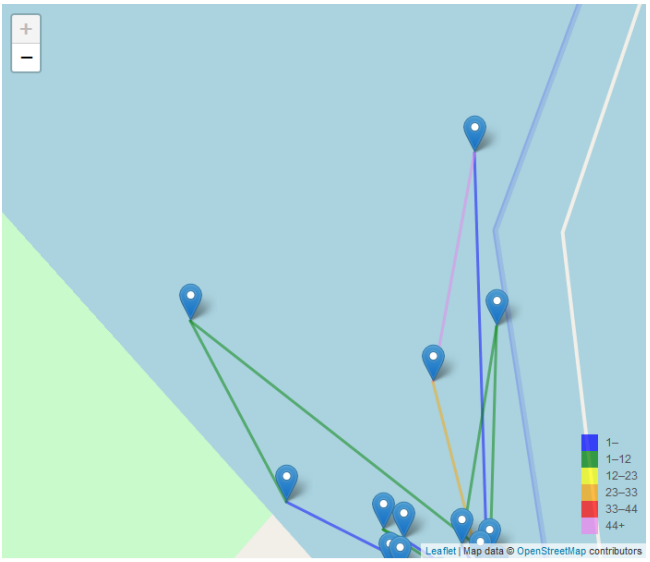

Figure 8: REVSkiTrack's Heatmap Functionality

Additionally, the graph is also usable as an analysis tool, implemented via the Dygraph JavaScript library. An example is shown below in Figure 9. REVSkiTrack includes a series of checkboxes that can be used to control whether the graph draws the plot of a set of data or not.

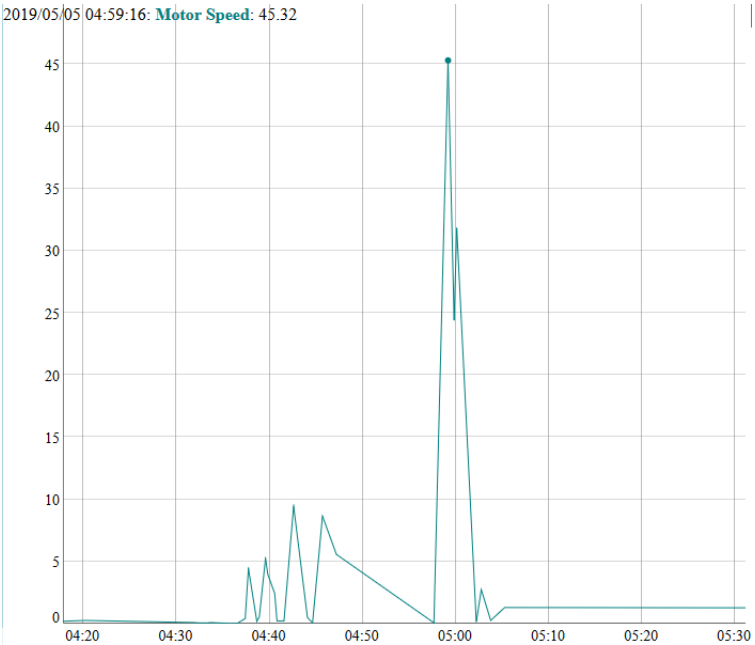

Figure 9: REVSkiTrack's Graph

6 Results

The two above figures, Figure 8 and Figure 9, are the results of a REVSki drive at Matilda Bay on the 5th of May, 2019 from about 4:30PM to 5:30PM, using the Astra's 3G modem. The data sent during the drive were successful in the sense that no transmission messages were corrupted or incorrect (so no wildly wrong values such as latitude or longitude above 360 degrees), proving the robustness of the transmission methodology. While the visualisation is shown to be working correctly, the problem of the 3G's infrequent data sending is still an issue, as evidenced by the large jumps in distances between data markers.

Unfortunately, due to time constraints and the lack of an available driver, the REVSki was not able to be tested with the 4G Luat during a drive on the water. However, the 4G modem was used on the 30th to the 31st of May 2019 to transmit REVSki data during a charging cycle. A sample of the results of this cycle can be viewed in Figure 10 below.

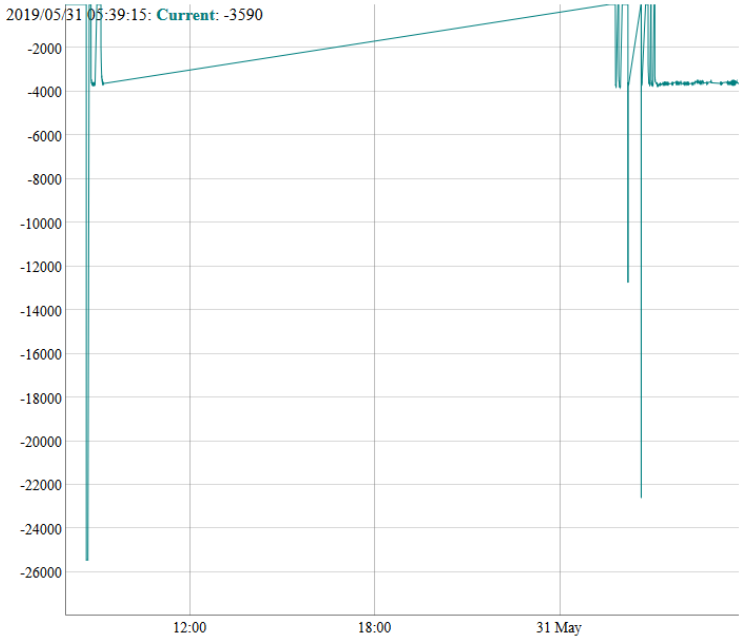

Figure 10: Graph of Current during Charging Cycle

Due to the availability of the visualisation web site, it is possible for the reader of this thesis to view the results in more detail. If this is desirable, feel free to have a look at https://therevproject.com/revskitrack.

7 Recommendations for Future Work

7.1 Improved Microcontroller

Due to the issues with transmission, it is recommended that future work involve the replacement of the Arduino Mega with a better microcontroller. This could have several benefits:

1. More memory. The Mega has 8KB of SRAM [8], half of which is currently being used. Should future work involve the addition of more data sets to save and transmit to the web server, it is very likely that the Mega's low memory will only serve to be a hindrance.
2. Higher clock speeds. The Mega has 16MHz of speed. 16MHz has no doubt been enough so far, but the number of tasks the Mega oversees has increased drastically from its previous iteration as a dashboard updater and SD card logger. As the REVSki receives more upgrades, it is plausible that the EWIS box will receive more responsibility, making a scenario where the Mega's clock speed is not good enough more and more likely.
3. Threading. No Arduinos have multi-threading, so users tend to resort to using the millis() function to run tasks on timers while rendering the delay() function unusable, much like in this project. A microprocessor with threads is much better suited for the EWIS box due to the number of tasks the Mega is in charge of carrying out (from updating the dashboard, reading the CAN network, reading the temp sensor, reading the water sensor, compiling transmission data in HTTP format, controlling the 4G Luat state machine, etc).

Some recommended microcontrollers with the above benefits are:

7.1.1 Raspberry Pi 3 B

A microcontroller with at least 512MB of SDRAM and more than 1GHz of clock speed [13] that also runs off its own operating system (based on Linux), which allows for threading. Cost wise, it can be expensive. A Raspberry Pi would still be able to interface with the rest of the EWIS modules through its four USB ports with the assistance of RS232 to USB and TTL to USB converters. Like Arduino CAN Bus shields, the Pi can gain CAN functionality via its own auxiliary boards such as the PiCAN.

7.1.2 Teensy 3.6

A small microcontroller with 256KB of RAM and 180MHz of clock speed [14] with libraries available for multithreading. Cost-wise, it is like Arduino Mega clone boards. A notable advantage of the Teensy 3.6 is that it has six serial ports, which are useful in case future development involves the installation of more modules that require interfacing via serial. Additionally, the Teensy 3.6 also has two independent CAN Bus ports and is thus able to communicate with the dashboard and EWIS CAN network without the help of an auxiliary shield.

7.2 Sim Card Solution

A low-cost solution to the sim card problem is recommended for future development, as the current 4G sim card in the Luat will have expired by mid-2019. The best scenario would involve having a sim card with either a very long expiry time or no expiry at all, but this is completely dependent on what Australia's telecommunications companies decides to offer.

There is the possibility that the current zero upkeep 3G sim card could be used as well, if one believes that the difference between 4G and 3G is not the sole reason as to why the Luat's worst case of 0.6 seconds is 100 times faster than the Astra's worst case of 60 seconds. It is

noteworthy that the 4G Luat has the sole function as a modem, while the Astra has a dual function as both a GPS and a modem (and handles some other functionality in the background as well). It is possible that due to the number of tasks that the Astra is doing, it cannot dedicate all its processing power towards being a modem which results in slower transmissions due to the Astra trying to complete another task instead. As such, it may be worth acquiring a cheap 3G modem (that works with Australia's 3G frequencies, which is something to watch out for if the hardware is sourced from overseas) to check if the Astra's other tasks are responsible for slowing down transmission or not.

7.3 The Dashboard

As mentioned in the section about the 4G Luat modem, the dashboard is for whatever reason not compatible with the Luat. This is especially bizarre due to neither the CAN Bus shield nor the PCB daughterboard share any pins nor resources. Due to a lack of time, an investigation into this issue has not been carried out, so it is recommended that this be done in future development to offer some insight into the problem.

7.4 Additional Data Transmission

With regards to additional data transmission, there is the recommendation to continue on with Wolozsyn's [2] and Hjariz's [15] safety instrumentation and store the data they were working on: battery temperatures and water depth sensors, which would make for good safety analysis tools when visualised on REVSkiTrack.

7.5 Additional Website Features

Lastly, as with all software, there are many additions and improvements that can be recommended for the website, so for the sake of being brief only the more pressing ones will be mentioned.

7.5.1 Optimization for Many Data Markers

Firstly, REVSkiTrack undergoes significant load when attempting to visualise many plots of data. For example, using the date selector to grab all data between the 1st May 2019 and 1st June 2019 results in a success, but navigating the map and zooming in or out is much slower than normal and prompts the browser console to print the following message below in Listing 2, indicating that the large number of markers is affecting the performance of the visualiser:

```
Will-change memory consumption is too high. Budget limit is the document
surface area multiplied by 3 (1271040 px). Occurrences of will-change
over the budget will be ignored.
```

Listing 2: Browser Warning about Memory Consumption when REVSkiTrack Loads Excessive Data

A possible solution to this problem is to optimize the map by eliminating data markers that are very close together, reducing the size of the data set that the map is required to display.

Alternatively, an attempt can be made to move data marker information out of the OSM map into another part of the website.

7.5.2 Date Selector Highlights

Another recommendation for REVSkiTrack is with regards to the date selector. Currently, while functional, there is no way for a user to know whether a date has stored data or not without trying it. It may be useful for viewers if the date selector is updated to highlight dates that contain data, which saves them time from needing to check multiple ranges of dates.

7.5.3 Mobile Cell-Displaying Webpage

Lastly, in order to explain this recommendation, it is first necessary to talk about the EVMS again. The newly installed ZEVA EVMS was delivered with a monitor [16] that displays information about the overall voltage and current as well as the voltage of each individual cell. It is particularly helpful when debugging and charging the REVSki. One issue with the monitor is that it is not waterproof, and cannot be taken on drives without risking damage, which is to be avoided to the monitor's cost being over a hundred dollars [16].

While it is still possible to check the REVSki battery cells by docking it by the jetty, removing the front covers and temporarily connecting the monitor, this is a hassle and could be made much more convenient by the development of a mobile-friendly web site that acts as an auxiliary to REVSkiTrack which serves the same functionality as the monitor, allowing future students to check the cell voltages on their phones instead. As of the time of writing, all the data displayed by the monitor is already being transmitted to the web server, which is half the work done already.

8 Conclusion

In conclusion, this project can be considered as a starting point for the REVSki monitoring system. It has successfully established basic visualisation in the form of web site and handled storage via a database. Transmission on the other hand, has been less successful. While the installation of the 4G Luat has technically made transmission quick and consistently robust enough to meet the project's scope of having live tracking and sufficient logging, it has also been set back due to the unexpected incompatibility between the Luat and the dashboard.

Once this issue has been solved, this project can be considered a framework for future upgrades to the monitoring system to be built upon, which in turn will provide a useful debugging, analysis, and comparison tool for future augments to the REVSki.

9 Resources

All source code written for this thesis has been made available on GitHub and is linked in Table 3 below.

Table 3: Table of Source Code

Resource	URL
Arduino Mega Sketch	https://github.com/dylanleong88/ECU_4G
REVSkiTrack PHP Source	https://github.com/dylanleong88/REVSkiTrack

10 References

[1] “REVSki,” [Online]. Available: http://therevproject.com/vehicles/jetski.php. [Accessed October 2018].

[2] M. J. Woloszyn, “Instrumentation for the REVSki; an Electrical Personal Watercraft,” University of Western Australia, 2018.

[3] A. Goldsworthy, “Remote Control of Autonomous Surface Vessels,” University of Western Australia, 2018.

[4] R. Stewart, “RFC4960: Stream Control Transmission Protocol,” September 2007. [Online]. Available: https://tools.ietf.org/html/rfc4960. [Accessed May 2019].

[5] “AT240 Datasheet,” [Online]. Available: https://gps-telematics.co.uk/wp-content/uploads/2017/09/AT240_datasheet.pdf. [Accessed February, March, April 2019].

[6] a. telematics, “Application Note: Pass Through Data Mode,” [Online]. Available: https://gps-telematics.co.uk/wp-content/uploads/2017/09/appnote_pass_thru_data_mode.pdf. [Accessed February, March, April 2019].

[7] M. J. Leven, “System Architecture & Instrumentation for Electric Jet Ski Project,” University of Western Australia, 2019.

[8] “Arduino Mega 2560 REV3 Specifications,” ARDUINO, [Online]. Available: https://store.arduino.cc/usa/mega-2560-r3. [Accessed April 2019].

[9] I. Hooper, “EVMS3 CAN Protocol,” May 2018. [Online]. Available: http://zeva.com.au/Products/datasheets/EVMS3_CAN_Protocol.pdf. [Accessed May, June 2019].

[10] “HTTP RECV WRITE,” [Online]. Available: http://therevproject.com/revskitrack/http_recv_write.txt. [Accessed April, May, June 2019].

[11] D. W. Leong, “REVSkiTrack,” The REV Project, [Online]. Available: http://therevproject.com/revskitrack. [Accessed February, March, April, May, June 2019].

[12] “Google Maps Pricing,” Google, [Online]. Available: https://cloud.google.com/maps-platform/pricing/. [Accessed March 2019].

[13] “Raspberry Pi 3 Model A Plus Specifications,” Raspberry Pi, [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-a-plus/#c-find-reseller. [Accessed May 2019].

[14] “Teensy Specifications,” PJRC, [Online]. Available: https://www.pjrc.com/teensy/techspecs.html. [Accessed May 2019].

[15] M. H. M. Jahis, “REVSki Electrical Safety System,” University of Western Australia, 2018.

[16] I. Hooper, “EVMS Monitor V3,” [Online]. Available: http://zeva.com.au/index.php?product=132. [Accessed May 2019].

11 Appendix

11.1 Appendix A

A test of the Astra 3G modem success rates without checking if the previous message has been successfully delivered to the web server. Taken over 5 trials.

Trial	Messages Sent	Messages Received	Success Rate (%)
1	87	19	21.84
2	90	20	22.22
3	89	19	21.35
4	90	20	22.22
5	62	13	20.97
		Average	21.72

11.2 Appendix B

A test of the 4G Luat's transmission speeds. Carried out by sending an HTTP request to the web server including a timestamp, then prompting the web server to record a second timestamp of when the message was received.

Sent Time	Response Time	Time Taken
15652	16186	534
23739	24079	340
26780	27303	523
30905	31420	515
38740	39222	482
41784	42310	526
45911	46417	506
53741	54093	352
56796	57320	524

60914	61426	512
68743	69084	341
71782	72307	525
75916	76435	519
80783	81293	510
88740	89083	343
91783	92280	497
95887	96398	511
103740	104099	359
106787	107288	501
110884	111383	499
115787	116303	516
123744	124084	340
126789	127296	507
130895	131391	496
138739	139096	357
141782	142273	491
145868	146403	535
153740	154091	351
156785	157285	500
160889	161418	529
165898	166419	521
173743	174076	333
176786	177296	510
180893	181392	499
188743	189085	342

191790	192296	506
195897	196401	504
203745	204083	338
206930	207443	513
211048	211554	506
218740	219210	470
221787	222292	505
225900	226407	507
230771	231289	518
238742	239100	358
241788	242283	495
245884	246387	503
253744	254074	330
256782	257296	514
260900	261417	517
268744	269080	336
271781	272303	522
275902	276413	511
283740	284092	352
286779	287290	511
290885	291404	519
298740	299073	333
301792	302326	534
305929	306448	519
313741	314086	345
316777	317335	558

320933	321444	511
328741	329066	325
331781	332288	507
335887	336419	532
343742	344090	348
346784	347291	507
350898	351411	513
358744	359092	348
361787	362295	508
365893	366400	507
373809	374137	328
376857	377378	521
381067	381557	490
388745	389097	352
391784	392314	530

Average	463.118421	in millis
Median	506	
Worst:	558	