School of Electronic and Electrical Engineering
FACULTY OF ENGINEERING

ELEC3885

Project Report

Design of a Web-Based Drone Management Platform using WebSocket

Ali Al Badra

	Group No. 1
Supervisor: Craig Evans	**Assessor:** Andy Kemp

School of Electronic and Electrical Engineering
FACULTY OF ENGINEERING

UNIVERSITY OF LEEDS

ELEC3885 Group Design Project

Declaration of Academic Integrity

Plagiarism in University Assessments and the Presentation of Fraudulent or Fabricated Coursework

Plagiarism is defined as presenting someone else's work as your own. Work means any intellectual output, and typically includes text, data, images, sound or performance.

Fraudulent or fabricated coursework is defined as work, particularly reports of laboratory or practical work that is untrue and/or made up, submitted to satisfy the requirements of a University assessment, in whole or in part.

Declaration:

- I have read the University Regulations on Plagiarism [1] and state that the work covered by this declaration is my own and does not contain any unacknowledged work from other sources.
- I confirm my consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to monitor breaches of regulations, to verify whether my work contains plagiarised material, and for quality assurance purposes.
- I confirm that details of any mitigating circumstances or other matters which might have affected my performance and which I wish to bring to the attention of the examiners, have been submitted to the Student Support Office.

[1] Available on the School Student Intranet

Student Name: *Ali Al Badra* Group No. *1*

Signed: Ali Al Badra [fy16aab] Date: *21/05.2020*

Abstract

Drone autonomy has been an active research area in recent years due to the large potential anticipated from large scale adaptation of drone technologies. Advances in the field of robotics have expanded the applications of autonomous drones; expediting their employment in defence sectors, for example, to help in disaster scenarios. However, from limited range to scalability restrictions, managing a network of autonomous drones remains an obstacle slowing down the vast deployment of drones in various applications. The purpose of this piece of work is to design a platform that facilitates autonomous drone deployment into a network of fire-fighting drones; managed and monitored through a web-based interface. Cloud services are used to manage such network, while WebSocket protocol is adopted for real-time, full-duplex communication. The platform offers features aiding in fire prevention and control in line with the overall project's goals. The system has been successfully demonstrated using simulated drones while evaluating its performance against responsiveness and network resource consumption under different usage scenarios.

Acknowledgment

I would like to thank my supervisor, Dr. Craig A. Evans, for his continued support and guidance throughout this project. I would also like to thank my group members: Rafeh Ishtiaq, Adrian Ozoemena, and Kamil Zabraniak for their cooperation. Finally, I would like to thank Andrew Bilbrough, the Teaching Support Technician, for his help in providing us with components and continued advice.

Table of Contents

Table of Figures

List of Abbreviations

GCS: Ground Control Station
IoT: Internet of Things
QoS: Quality of Service
WSN: Wireless Sensor Network
M2M: Machine to Machine Communication
DDP: Drone Discovery Protocol
WAN: Wide Area Network
LAN: Local Area Network
FMU: Flight Management Unit
ESC: Electronic Speed Controller
Rpm: Rotation per Minute
PDB: Power Distribution Board
RPI: Raspberry Pi
LiPo: Lithium Polymer Battery
PWM: Pulse Width Modulated
AWS: Amazon Web Services
RTC: Real Time Communication
HTTP: Hypertext Transfer Protocol
ACK: Acknowledgment Packet
SYN: Synchronization Packet
TCP: Transmission Control Protocol
WS: Websocket Protocol
DBMS: Database Management System
UAV: Unmanned Aerial Vehicle
Bps : bits per second

1. Introduction

Drones, much like airplanes, must communicate with a ground control station (GCS) that monitors their location, status, and surrounding air-traffic. However, unlike airplanes where pilots can monitor internal components mid-flight, drone pilots become detached from internal components once their drone takes off. Therefore, the need for an off-board device capable of monitoring and possibly controlling drones arises.

The importance of a GCS becomes apparent when designing an autonomous drone; as the absence of a pilot means users have no control over the drone in-flight. That sense of control needs to be maintained; either through manual control whenever required, or through closely monitoring the drone as it completes user-defined missions. This presents an issue for autonomous drones as they might jeopardize their radio connection with the GCS if they move further away from it.

In line with the project's objective of designing an autonomous firefighting drone capable of completing long-haul missions, this centralized approach of having a single device act as a GCS presents a barrier to true drone autonomy. Furthermore, commercial GCS applications, such as QGroundControl, introduce a range of limitations; from the inability to scale to more than a single drone, to the limited range of pre-defined generic features; making them impractical for this projects' needs.

In an effort to overcome the limitations mentioned earlier, this chapter presents the design and implementation of a user-friendly platform capable of performing necessary GCS tasks while allowing for scalability and providing custom features in line with the project's goals. Finally, the proposed system's performance is analysed in a testing environment and the relative issues are discussed.

The proposed solution is a web-based drone management and monitoring platform capable of managing multiple drones, while serving an arbitrary number of users via the internet. The proposed system was designed around firefighting drones; featuring thermal image processing, mission planning, and manual control enabled through a web-based interface. Moreover, it takes a decentralized approach to

generic GCS implementations by hosting its GCS application on servers across the internet; virtually distributing access to the GCS for both users and drones.

This approach is made possible through the use of cloud computing [1]; subscription-based services that enable the delivery of computational resources, such as networking, storage and servers, over the internet. Internet connectivity is enabled on drones through the use of 4G modules; turning each drone into an Internet of Things (IoT) device within a large and scalable network of drones.

The two end-points of the system, drones and users, will communicate over the internet using WebSocket; a newly emerging network protocol that enables bi-directional data flow between endpoints. A website is then used to provide a human-friendly web-based interface between users and their drones, regardless of their respective location, network, or hardware.

2. System Design

To design such system, one must take into account the different components and how they interact with each other to work as a coherent system capable of sustaining as many users and drones, while maintaining a steady quality of service (QoS).
The system is designed around abstraction; a concept that deals with complexity by hiding unnecessary details from users, allowing them to focus on the overall conceptual ideas of the system.

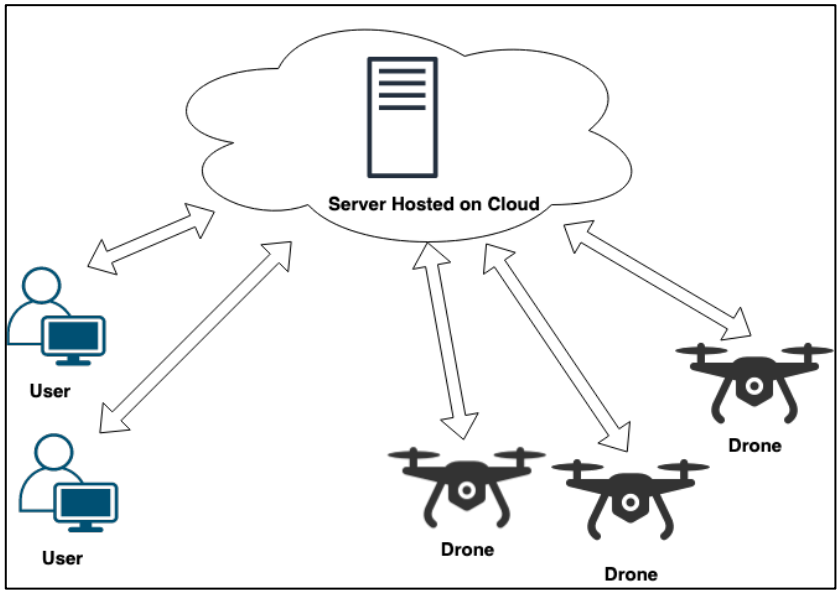

Figure 1 – Overall System Design

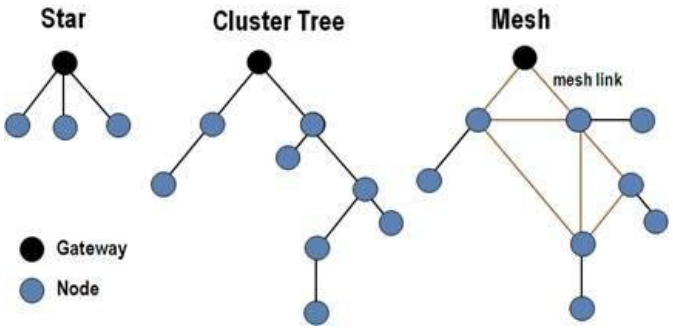

Figure 2 - WSN topologies. Source: http://researchgate.net

Fig. 1 shows a simplified view of the overall system design where a hosted server is responsible for routing messages between endpoints. The cloud acts as a middleman; its main purpose is to provide an interface between the endpoints, while also allowing users to manage and monitor multiple drones.
This design has been derived from wireless sensor networks (WSN) [2][3]; a model that enables the deployment of a large number of small, self-operating sensors capable of sensing, processing, and routing data. Those sensors are then thought of as nodes in a network and could be configured to either communicate with each other in a machine to machine (M2M) manner, or directly with the gateway node..
However, although drones can be thought of as sensors in a WSN, they do not share typical sensor limitations that WSN try to overcome; such as low power and limited computational capabilities. This system adopts WSN's Star Topology to enable the deployment of multiple drones [4], as shown in Fig 2, where the cloud acts as a gateway node.

2.1 Abstraction

Abstraction [5] is the method with which software developers deal with complexity. Behind every coherent system lies a web of sophisticated, interconnected components and services that, once linked and configured together, work seamlessly as a single system. However, their specific implementations are usually hidden from their users to ensure focus is on the conceptual ideas of the whole system, rather than the implementations of the integrated components.
Google Maps, for example, illustrates abstraction by hiding unnecessary geographical details from the user. When zooming in, users observe a detailed view of the geographical landscape; with individual streets and buildings. As users zoom

out, Google hides those unnecessary details in that context; shifting user's attention to countries, oceans, and continents.

One way with which abstraction has been achieved in this system was by hiding drone-related low level details, such as drone IP, from the endpoints of this system.

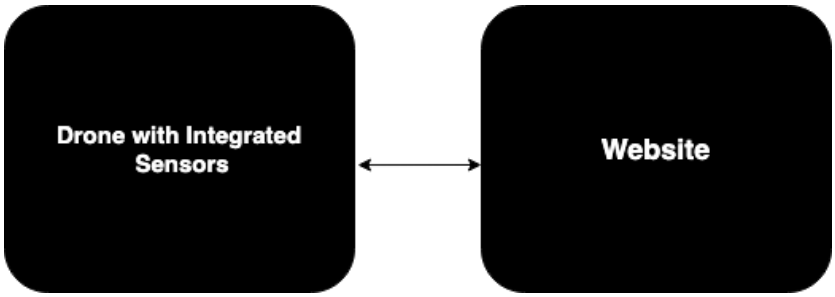

Figure 3 – Blackbox Topology

Here, the drone is thought of as merely a black box; relaying its sensor data to a website regardless of its location, hardware, operating system, or sensor-specific implementations.

The main driver of this system is data; as long as data sent across the system follows a previously agreed upon format, components will be able to communicate and serve end-users.

A good example of how abstraction has been achieved on this system can be demonstrated through the Drone Discovery Protocol (DDP). But first, general routing concepts must be made clear.

Routing is the process with which messages are transmitted across the internet, this is done using devices known as routers; with the goal of routing messages either within one network, or across different networks.

Each network has a designated edge-router responsible of connecting it to wide area networks (WAN), as well as internal routers and switches responsible for routing messages between the different devices within a local network (LAN).

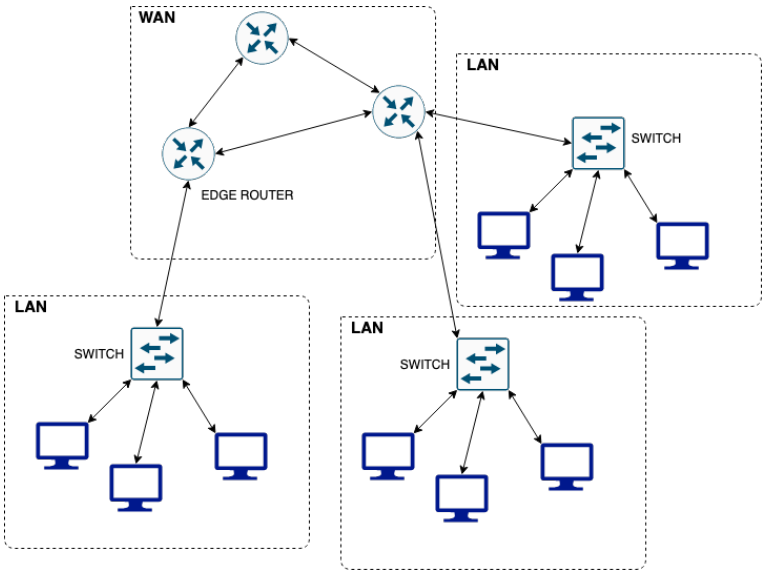

Figure 4 – Simplified Network Topology

Generally, devices across different networks can only communicate if one or both of their IP addresses are known, this is to allow edge routers to route their messages to the wanted destination. Without a known and discoverable IP address, messages simply will not reach their destination. In the context of this system, drones are internet connected devices constantly moving across different networks; this poses the first challenge: connectivity across the internet.

Traditionally, for a drone to exchange data with the server, it's IP address must be known. This is because the drone is constantly moving across networks and therefore it's IP address changes accordingly. To tackle such low-level problem, the system ensures the server is publicly accessible by hosting it on the cloud; this gives the server a hostname through which drones can initiate a connection. This raises the level of abstraction adopted; as it hides away the complexities of networking and IP addresses; while also allowing drones to be discovered regardless of their location. DDP is discussed in greater details in section 3.1.1.

The integration of cloud computing services in this system has enabled such levels of abstraction, in addition to the added capabilities of scaling; making it an integral component of the system.

2. Drone Assembly

Drone assembly, unlike software development, is a labour-intensive task that requires vigorous testing and debugging, which justifies the five-months period spent

on it. Most of the issues faced throughout this task, aside from missing or faulty components, were related to aeronautics [6]; the study of the science of flight.
The frame, as seen in figure 4, consists of a carbon fibre landing gear arms, as well as mounting plates for the Flight Management Unit (FMU) and the various other components. It is lightweight at only 475 g yet strong enough to withstand collisions.

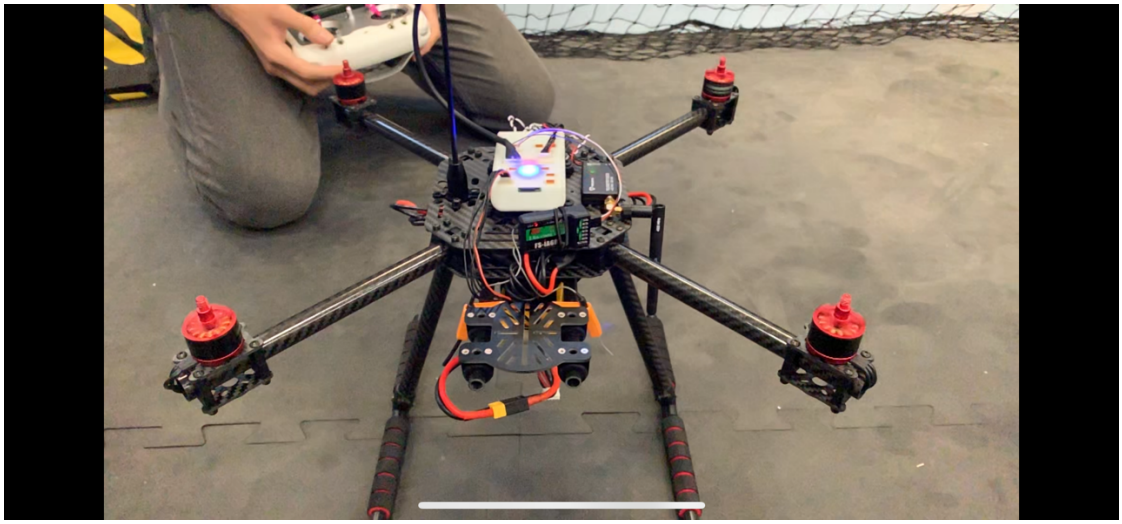

Figure 5 - The drone

At the end of each arm is a ReadytoSky RS2212-920 Kv brushless motor capable of providing up to 624 g of thrust while drawing 30 A. Generally, each motor is expected to lift a maximum weight equal to half its thrust rating. Each motor is also connected to a 40 A electronic speed controller (ESC). The Kv value used when describing a brushless motor refers to its speed constant [7]; a ratio of the motor's unloaded rotations per minute (rpm) to its peak voltage. ESCs control the speed at which each motor runs and ensures all motors run at the same speed. In case of a mismatch in the Kv rating between the four motors, ESCs will work to reduce the speed of each motor to that with the lowest Kv value; as a mismatch would cause the drone to exhibit unexpected behaviour in-flight. The ampere rating represents the amount of safe current that can pass through the ESC. The ESCs are finally connected to a power distribution board (PDB) which in turn supplies each ESC with 12 V.
The drone weighs 0.9-1 kg without the cameras, Raspberry Pi (RPI), and sonar sensors. It is estimated to have a maximum total weight of 1.1-1.2 kg as other components and connectors are added. This total weight, when evenly distributed across the four motors, yields the weight each motor needs to lift. As mentioned

earlier, this weight is less than half the motor's 624 g thrust rating and therefore the drone is capable of lifting its own weight under the chosen specifications.
The drone is powered by a 5000 mAh 3S 50C Lithium Polymer (LiPo) battery connected to the PDB. The 5000 mAh capacity was chosen to provide long flight times for large missions while the 50C continuous discharge rate draws sufficient current to the ESC. According to [8] the maximum safe current draw from a LiPo battery can be calculated as follows

$$Current\ Draw\ (A) = Capacity\ (Ah) \times\ Continuous\ Discharge\ Rate \quad (1)$$

According to (1), the maximum safe current that can be drawn from the battery is 250 A, while the maximum safe current that can pass through each ESC is 40 A, delivering the required 30 A to each motor and thus ensuring compatibility.

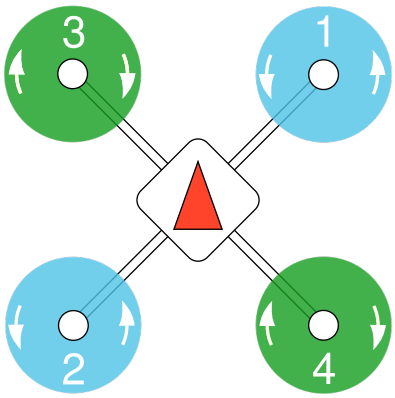

Figure 6 - Quadrotor-X airframe

Source: http://nxp.gitbook.io

Figure 7 - pinout of FMU servo rails

Source: http://nxp.gitbook.io

As for the aerodynamics of the drone, it belongs to the Quadrotor-X airframe category. As shown in Fig. 6, each motor is faced with an oppositely-rotating motor to produce enough thrust to take-off. Each motor connects to its specific PWM pin on the FMU as shown in Fig. 7; this is to allow the FMU to control the motors once in-air.
Finally, embedded in the FMU is an accelerometer and a gyroscope to stabilize and control the drone in-flight. Those must be calibrated regularly to maintain a controlled flying experience. Unexpected behaviour is expected if the FMU is not calibrated, this includes pitching or rolling at an angle; making it harder to control the drone.

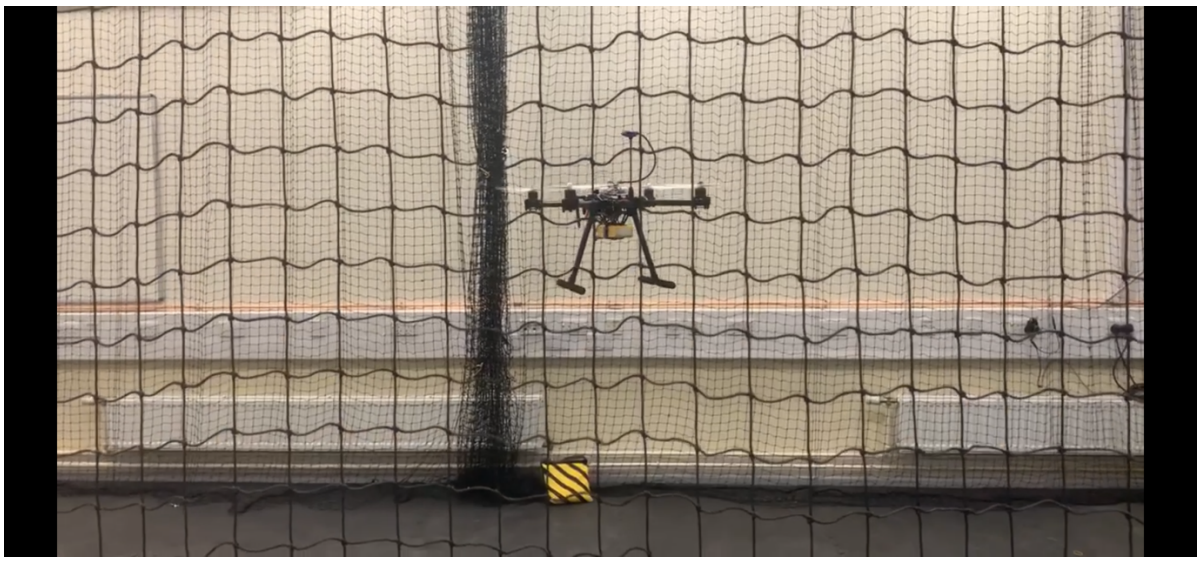

Figure 8 - Drone Flight Test

3. Cloud Integration

As mentioned earlier, the cloud has enabled ,and eased, the integration of a number of services within the system; which made it an irreplaceable component of the system.

Cloud Computing, as defined by Amazon, one of the largest providers in the field, is [9] “the on-demand delivery of IT resources over the internet”, a simple use case for cloud computing is by utilizing storage resource made available by a cloud service provider, such as Amazon Web Services (AWS). One would request a certain amount of storage from AWS who in turn allocate this storage on their computer clusters; giving customers fast and reliable access to the data stored, and offloading the storage limitations they might be facing in their respective systems.

Offloading computational power, on the other hand, might be an even more powerful use of cloud services in regards to IoT projects; it provides the small and perhaps simple IoT devices and sensors with an arbitrary computational power that’s only limited by the amount of money the client is willing to spend.

In the context of this system, cloud computing solutions have been utilized to host a dedicated server application controlling both backend and frontend operations, allowing for great flexibility and ability to scale. In backend operations, the server is mainly responsible for routing messages between users and their drones, as well as processing and storing drone data in real-time. Regarding frontend operations, the server hosts a drone monitoring and management website, named “DroneIO”, in which users can register new accounts, deploy drones, and manage them.

To establish a connection to the server, a 4G module will be attached to the RPI; providing continuous internet access and aiding in the pursuit of drone autonomy. The proposed module is a Quectel EC25 Mini PCIe 4G/LTE Module which attaches to a sixfab 3G-4G base-shield. The base shield connects to RPI through the 3.3 V UART port and receives power directly from the USB port.

Figure 9 - sixfab 4g shield integrated with RPI
Source: http://cnx-software.com

3.1 Backend Operations

The main purpose of a server applications is to execute backend operations; computations done in background without knowledge from the users of the system. Examples of backend operations include firing requests to fetch data from other servers, retrieving records from a database, and checking against a list of emails when a new user tries to register.

Every GCS needs to maintain a secure and stable connection with its drone(s) to ensure bi-directional data flow between users and drones regardless of the drones status; whether it's carrying out a mission or staying idle at a known location. Moreover, users need drone data delivered to them in real-time with minimal latency to facilitate smooth and reliable control through the website, making the GCS a Real Time Communication (RTC) application.

3.1.1 HTTP

In traditional web application design, server and client communication follows the HTTP [10] protocol in which a client initiates a connection with a server via a number of acknowledgment (ACK) and synchronization (SYN) messages exchanged between the two devices, known as a TCP handshake [11]. Once a connection is established, the client may request data from the server, which then serves available

data back to the client before immediately closing the connection. Such communication protocol is synchronous [12]; since a client request is always followed by a server response, and a response cannot be issues without a request. As a result of this synchronous nature, HTTP communication is said to be half-duplex [13], meaning data could only be delivered from server to client, upon client request, and not the other way around. While such protocol serves the needs of traditional websites, where clients requests webpages through URLs, it poses as a challenge for RTC applications requiring instant data transfer, such as chat and gaming applications, since clients need data delivered as soon as it becomes available on the server; something HTTP was not designed for.

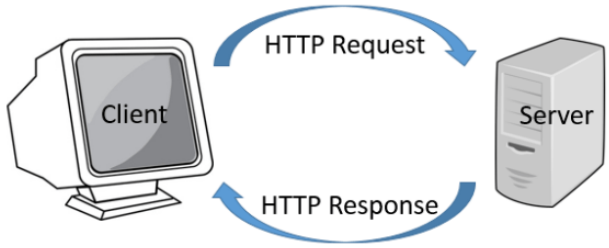

Figure 10 Traditional HTTP Communication. Source: http:i0.wp.com

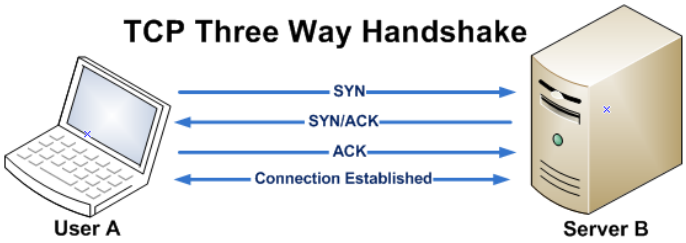

Figure 11 – TCP Handshake. Source: http://freecodecamp.org

3.1.2 Polling and Long-Polling

To overcome the half-duplex nature of HTTP communication, RTC applications would overwhelm the server with requests sent at a high frequency, a technique known as polling [14]. While polling gives HTTP the functionality of a bi-directional channel; it incurs unnecessary overheads in the form of bandwidth and server responsiveness[5], caused by the continuous request-response process happening at a high frequency. Therefore, a variation of polling was developed, known as long polling [14]; a form of polling in which the server keeps a connection open for a short period of time following a client request if data is not yet available. If data becomes available at the server, during that time window, it will be delivered to the client

before closing the connection, otherwise the connection is closed and the client needs to initiate a new connection.

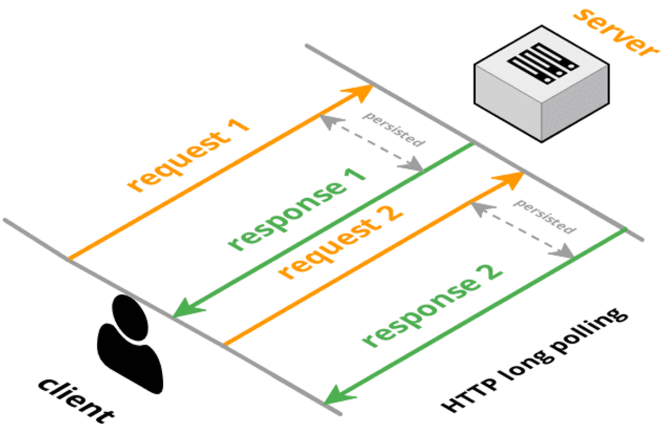

Figure 12 HTTP Long-Polling.
Source: http://ably.io

Although bandwidth usage is significantly decreased in long polling as opposed to polling, long polling tries to facilitate full-duplex communication over a traditional HTTP connection which is naturally half-duplex, this makes it more intensive on the server, as it adds the need to keep a connection open after a client request, which HTTP servers do not perform natively.

3.1.3 WebSocket

To enable the development of RTC web applications, a new protocol called WebSocket (WS) was defined. WS is a natively asynchronous, full-duplex communication protocol that provides flexible bi-directional data flow between clients and servers to ease the development of RTC applications [13][15], and provide fast and reliable data transfer. WS, when compared with its closest competitor long polling, offers lower bandwidth resource consumption [16] and average latency, making it the most efficient communication protocol for this drone management platform.

To establish a new WS connection [17], the client first initiates a handshake over an HTTP connection, much similar to a TCP handshake in HTTP. When the server receives the handshake request, it identifies it as an connection upgrade request; upgrading the connection from HTTP to WS. Once connection has been upgraded, a WS connection is established, allowing bi-directional data flow between client and server.

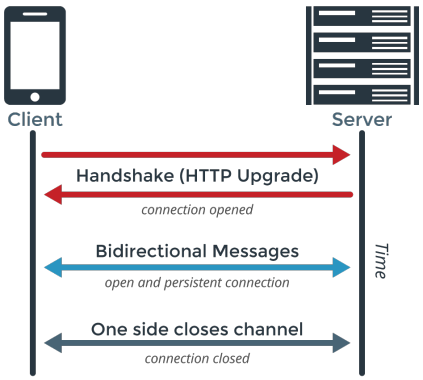

Figure 13 – WebSocket Protocol.

Source: http://pubnub.com

A network architecture for the drone management platform therefore needed to be designed and developed around WS to enable a number of features through the web, such as drone deployment and discovery, one of the main backend operations carried out by the server application. The architecture was built using the Laravel-WebSockets [18] PHP package; a PHP library providing an API for developers to build their RTC applications around WS. Using Laravel WebSockets, developers may create dedicated channels and event. Channels are WS communication channels created by the server application. Once a channel is created, multiple clients may subscribe to a single channel in which they listen to events. An example of a channel would be a user notifications channel in a social media web application, with clients being registered users, and events being likes or tweets. Channels on Laravel-WebSockets are very versatile; allowing developers a range of configuration options, including custom hosts, ports and encryption standards. Moreover, WS channels could also be configured as public or private, with private having stricter security measures. In the context of this platform, each registered drone will be given a dedicated channel based on the its unique ID as shown in Fig. 14. Once a drone is subscribed to its channel, it starts listening to server events ,such as “Send-Status”, to send the drone’s current status.

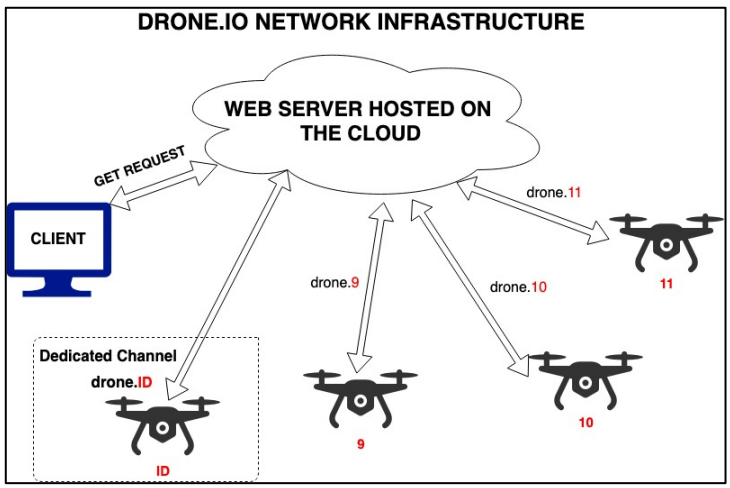

Figure 14 - droneIO Network Infrastructure

This architecture automates DDP; once a new drone comes online it connects to the server application which can be reached via its public URL as its hosted on the cloud. Upon successful connection, the communication protocol is upgraded from HTTP to WS, as described earlier, and a “register” request is then sent by the drone. The register request is a WS message sent via the public WS channel “home” which will be used to carry out non-exclusive communication between server and client; such as registering drones before a private channel is assigned to them, and monitoring the online status of all drones. Once the server receives the register request, it assigns a unique ID to that specific drone and sends it back through the channel to be referenced in all communication between the two endpoints moving further.

```
Subscribed to channel: home
Registering Drone..
Received Message on channel home:
[ 20 ]
Subscribed to channel: drone.20
```

Figure 15 – Client Side Sequence of Events During DDP

Data	Length	Time
{"channel":"home","event":"App\\Events\\NewMessage2","data":"{\"header\":\"register\",\"airframe\":\"Mini-Quadcopter\",\"deployed_by\":1,\"is_online\":true}"}	158	01:10:36.057
{"channel":"home","event":"App\\Events\\NewMessage","data":"{\"message\":20}"}	78	01:10:36.136
{"channel":"home","event":"App\\Events\\NewMessage2","data":"{\"header\":\"online\",\"id\":[20]}"}	98	01:10:36.154

Figure 16 - Server Side Sequence of Events During DDP

As with other design aspects of this system, this architecture contributes to a higher level of abstraction as it refers to drones using their unique IDs rather than their IP addresses, while all communication is abstracted through channels and events rather than packets and ports used in traditional low-level communication.

On the client side, as seen in Fig. 15, the client first subscribes to channel “home”. This subscription is then followed by the register request shown as the first row in Fig. 16. To make a successful registration request, a message with basic drone information, such as that shown in Fig. 17, needs to be constructed. Finally, a new drone record is added to the server’s database to successfully register the drone in the system.

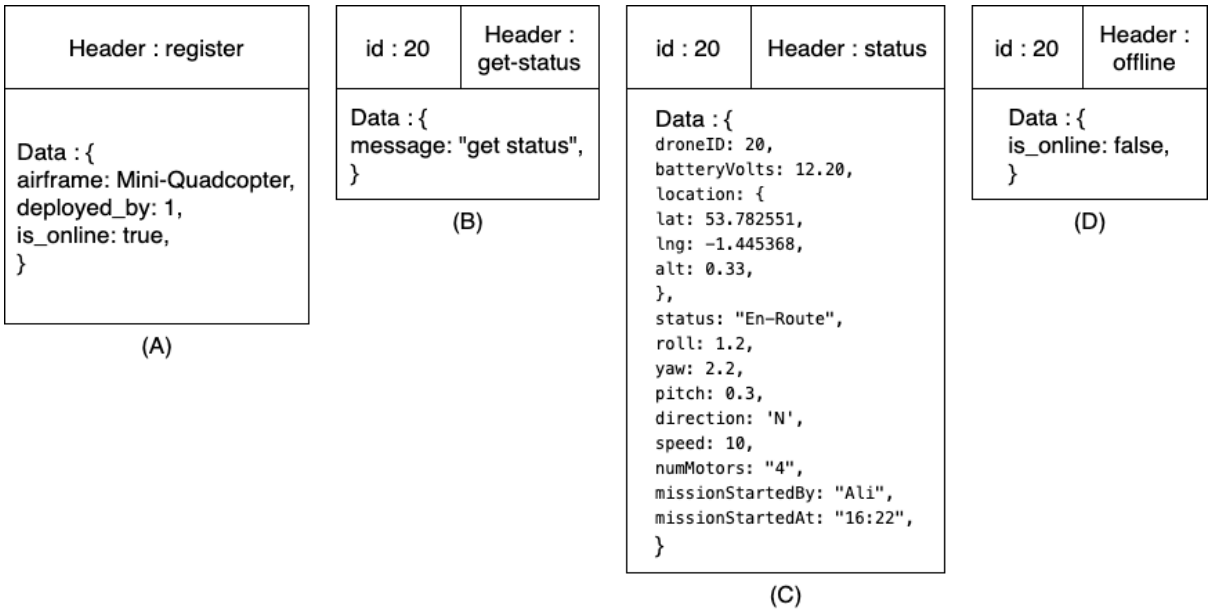

Figure 17 - message types. (a) register request, (b) data request, (c) data response, (d) offline notification

All WS communication taking place within the system will follow the four message types in Fig. 17.

3.2 Database Management

When designing a large distributed system, a log of system records must be stored along with the records of its corresponding endpoint. This log, known as a database, is defined by Oracle [19] as a collection of electronically stored, structured data modelled as rows and columns in tables that can be accessed via specialised software and languages. To access database records, a Database Management System (DBMS) is needed. DBMS [20] is a specialised software responsible for executing database management tasks, such as storage, retrieval, and organization of data. For the purposes of this platform, MySQL will be used as a primary DBMS for its ease of use and high compatibility across different platforms. MySQL is a fast

[21] and reliable open-source DBMS, used in large enterprises including Facebook, Netflix, YouTube, and Twitter [22]. Other DBMS options include PostgreSQL, used by Instagram, and Oracle DBMS, used by Amazon.
In the context of this system, database management takes place in two levels: local database management at the drone level, and global database management at the server level.

3.2.1 Local Database Management

At the drone level, each drone needs to maintain a database to store its own coordinates and other related data needed for the functionality of all systems within the drone. It is important to realize that the drone is not simply a single device, but rather a system consisting of a number of smaller subsystems. Those subsystems are the obstacle detection, thermal processing, and drone autonomy systems. Every subsystem generates its own set of data, which other subsystems may depend on for some features. Thermal processing, for example, may output temperature levels at a given time, while drone autonomy produces current coordinates from the GPS module, meanwhile obstacle detection outputs warning regarding closely approaching obstacles. As it can be seen, all three subsystems output data, but without coordination between them this data cannot be properly utilized. Therefore, all subsystems must publish their data to a local database on the RPI; so other subsystems can fetch and update records as necessary.
An example of this coordination is demonstrated in Fig. 18, where the process in green is one that runs locally on the drone's RPI.

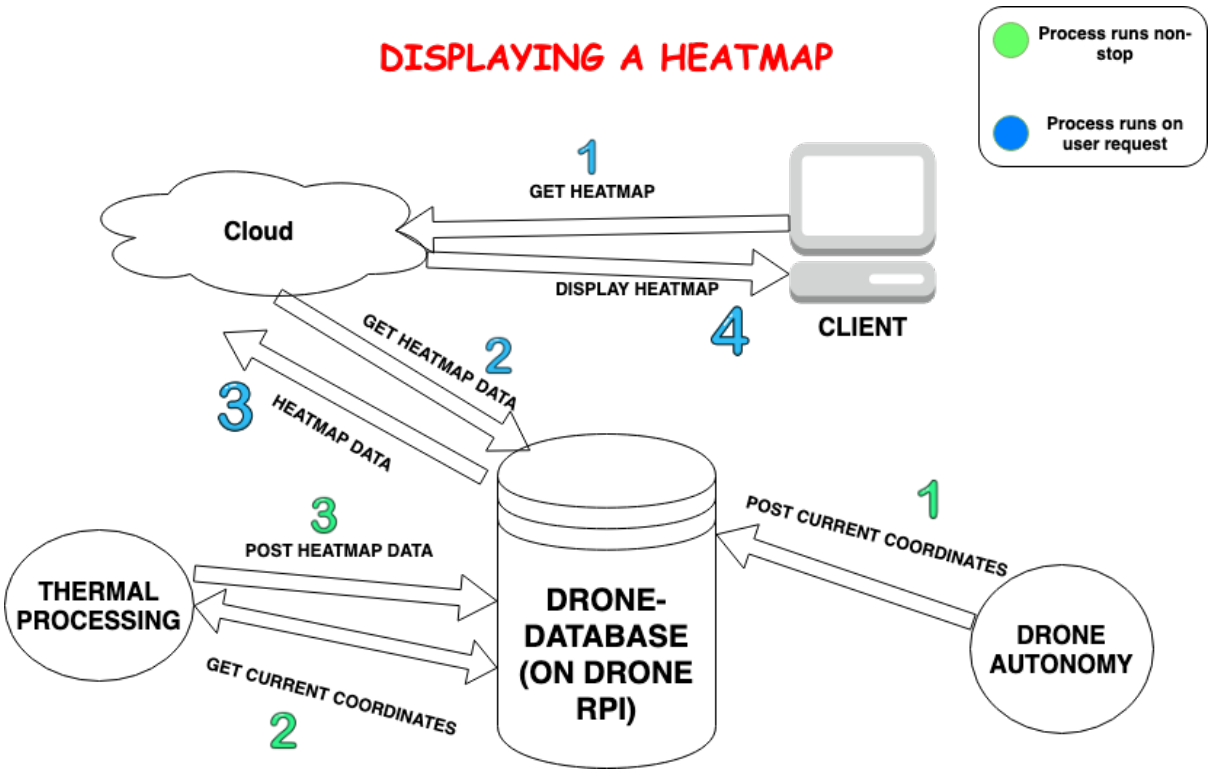

Figure 18 - backend operations when displaying a heatmap

As seen above, the database acts as a middleman between the subsystems within the drone, and with the users through the cloud. The main source of data in this example is the drone autonomy system; as it feeds the database with current co-ordinates that other subsystems need to perform their calculations, whether that's calculating pixel heatmap data through the thermal processing system, or sending status data to the cloud upon user request via the drone management system.

id	lat	lng	alt	weight
1	47.2083	8.54533	14.532	NULL
11	47.3133	8.54549	14.535	NULL
21	45.8081	8.5411	14.521	NULL

(A)

id	lat	lng	alt	weight
1	47.2083	8.54533	14.532	3
11	47.3133	8.54549	14.535	1
21	45.8081	8.5411	14.521	5

(B)

Figure 19 - local database. (a) before thermal processing, (b) after thermal processing

Fig. 19 shows how the contents of the database change at different stages of execution. Here, (A) shows the database contents at step one shown in Fig. 18, while (B) shows how the contents are dynamically changed at step three when heatmap data are attached to coordinates. Utilising MySQL's fast database management operations, this mechanism can be maintained at high frequency; offering real time data reporting to users and other subsystems while ensure data integrity among all subsystems.

For demonstration purposes, and in light of the current situation, a public MySQL database has been used for this system. To make the database accessible, Heroku [23] services have been used to host this database and give it a public URL. Heroku is a cloud platform that offers web hosting services for developers and organizations to host their web applications during development and deployment stages.

3.2.2 Global Database Management

To manage and organize the data flowing across the server, whether its data response messages or registration requests, a global level database needs to be set up on the server. The global database will consist of a number of different tables, with each table handling different types of data while being linked to the other tables through shared identifiers known as keys.

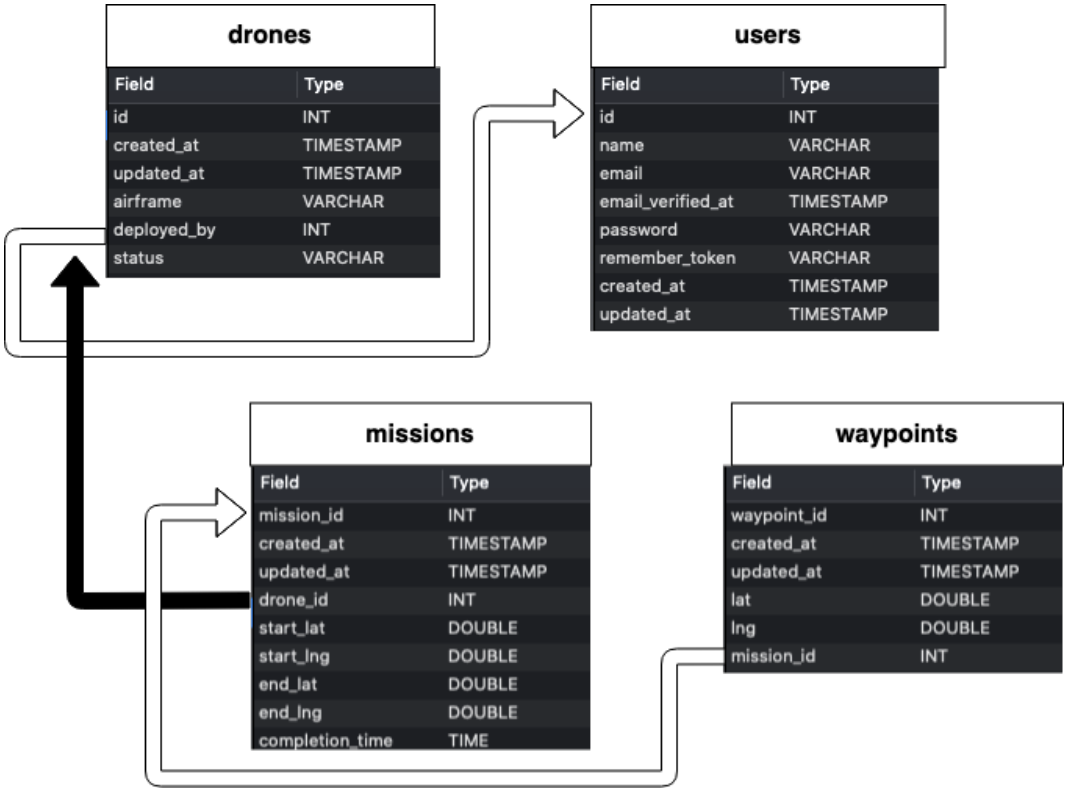

Figure 20 - relationships between the different tables at the global database

Fig. 20 shows the four main tables responsible for system functionality and data flow, along with the relationships between them. The arrows signify a relationship and those relationships are set through the shared keys. For instance, every drone deployed needs to be linked to a single user account, hence the drone's "deployed_by" field must be linked to an "id" in the users table. Additionally, each planned mission will have multiple waypoints and will always refer to a single

database, therefore a relationship was set between missions and waypoints via the key “mission_id” and between mission and drones via the key “drone_id”.

id	created_at	updated_at	airframe	deployed_by	status
1	2020-04-24 18:53:57	2020-05-09 18:08:39	Quad-Copter	1	online
2	2020-04-24 18:54:52	2020-05-02 16:40:53	Octo-Copter	2	offline
3	2020-04-24 18:56:06	2020-05-02 16:40:53	Quad-Copter	2	offline

Figure 21 - a populated drones table

The drones table is especially important in that it keeps track of all registered drones along with some basic information. When a new drone attempts to register, the server inserts a new entry into the drones table, with the ID field incrementing with every new entry. This ID value is then sent back through the WS channel to the drone; for a new WS channel to be opened with its corresponding ID.

3.3 Frontend Operations

DroneIO is the main point of contact between users and their drones. It enables drone management and monitoring services for an arbitrary number of users and drones. The motive behind designing a website rather than a generic graphical user interface application is to allow users access to the service regardless of the device they’re using, whether it’s a smartphone, laptop, or even a tablet. This flexibility eventually contributes to the high level of abstraction stated earlier; as it removes away any hardware-related limitations.

Cloud services enable frontend operations by providing web hosting [24]; services that allow developers to upload their websites onto the internet. When a website is hosted on the cloud, it's webpages are uploaded onto a cloud web server which delivers those pages to users trying to access the website through the public URL supplied.

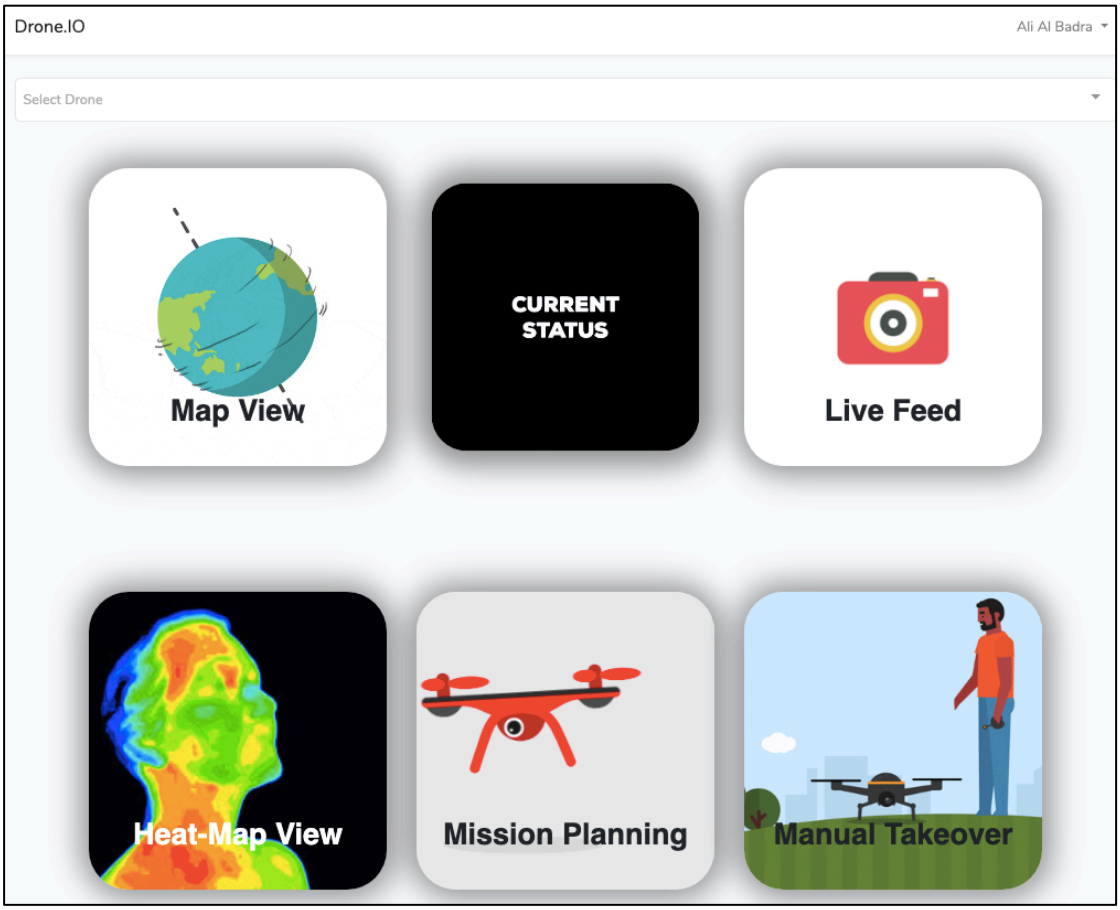

Figure 22 - DroneIO Dashboard

DroneIO consists of a total of nine webpages, six of them serve the main features shown in Fig. 22, while the remaining three are the welcome, login, and dashboard pages. The six main features have been chosen to give users all the information they need to manage and control their drones.

3.3.1 DroneIO Features: Map Views

One of the main requirements for a GCS is the ability to view drone location in real-time. DroneIO provides this feature with the added option of viewing multiple drones on the same map; allowing users to monitor all, or part of, their drones on the same page.

Viewing the drone's location on a map requires knowledge of the drone's most recent GPS coordinates, as well as a map rendering service. Creating a dynamic map of the whole world from scratch is infeasible, therefore, Google makes its Google Maps API [25] available for developers to integrate into their website and web applications; allowing them to make requests to Maps servers just like the Google Maps webpage does. In addition to displaying the generic map, Maps API offers a range of customizable features to enable the integration of Google Maps in various use cases. For the purposes of map viewing, user-defined markers in the shape of drones have been used instead of the generic Maps markers, as shown in Fig. 23.

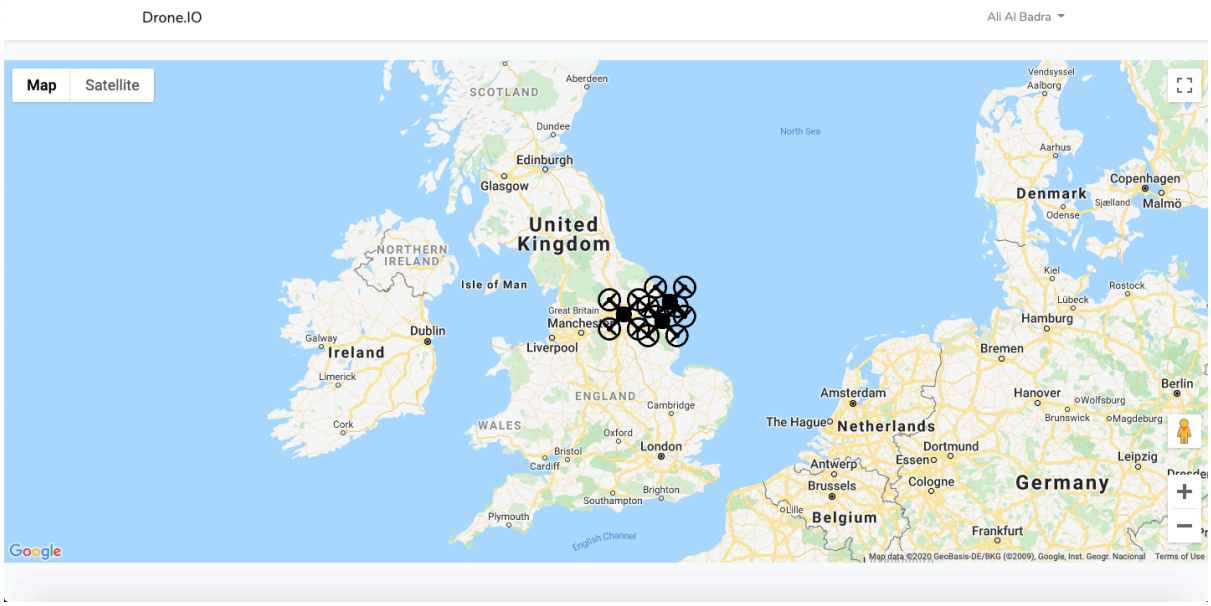

Figure 23 - map view webpage showing three drones.

To obtain the most recent drone coordinates, and observe drone movement in real time, a WS data request message is sent to the individual drones through their dedicated channels. Once received, the drones would start sending their current coordinates at a certain rate; so as not to overwhelm the server. The rate at which messages will be sent, and the amount of data in each message, are key aspects when analysing system performance and network resources consumption in section 5. In general, though, the faster the rate of transmission, the more accurate and real-time map data will be, but the higher the risk of messages getting discarded at the server due to its inability to process the large volume of messages.

Based on the map view model described above, Heatmap View has also been implemented through the use of the heatmap overlay feature in Google Maps.

Heatmap overlay gives developers access to a type of special library, different to that used in the regular map view, where they could attach a temperature value to each set of coordinates; adding a heatmap layer over a regular map. This application is especially useful for the purposes of this platform; since deployed drones will aid fire departments in fighting fires.

The heatmap view webpage demonstrates how three subsystems come to work together as a single system, with each executing in a different environment, programming language, and hardware. The backend operations related to producing the heatmap view can be seen in Fig. 18.

Figure 24 - heatmap view

3.3.2 DroneIO Features: Mission Planning and Control

In a traditional sense, drones are usually controlled manually via a remote controller through radio channels. However, and with the emergence of autonomous robotics, drone autonomy became a much needed feature for large-scale operations. In the firefighting scenario, for example, it would be far more practical to deploy a large fleet of drones on pre-defined routes to survey large areas of land for possible fire outbreaks, rather than employing a team of expert drone pilots to manually control each drone at a time. It not only adds practicality, but also increases the efficiency of the process as more fires can be identified and suppressed at an early stage, while freeing up personnel for more important tasks. The adaptation of drones in emergency rescue situations is an emerging application of drones [26][27], and one that puts great emphasis on drone autonomy technology to make it possible.

In an effort to design a refined GCS, DroneIO implements mission planning through the web using Google Maps API to place waypoints on a map, as shown in Fig. 20, and eventually set up a flight route for the drone to follow. Once waypoints have been defined, the coordinates are stored at both the local and global database; making this information readily available at both endpoints to keep track of each drone and its assigned missions at the server level. Fig. 20 shows the waypoints and missions tables, and how data flow and integrity is maintained among them. A mission is first created and a new entry is inserted into the missions table with a unique “mission_id”. Waypoints, defined by the user using the interactive map, are stored in the waypoints table, with their “mission_id” fields referring to the same mission so as to link them all together.

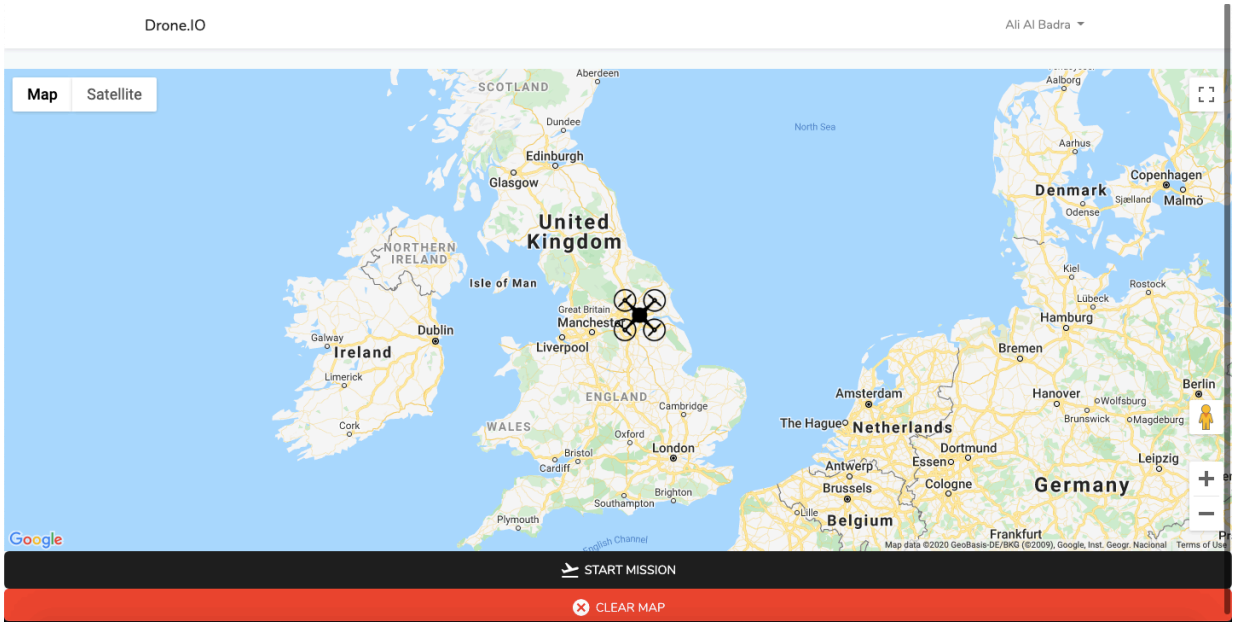

Figure 25 - Mission Planning Page

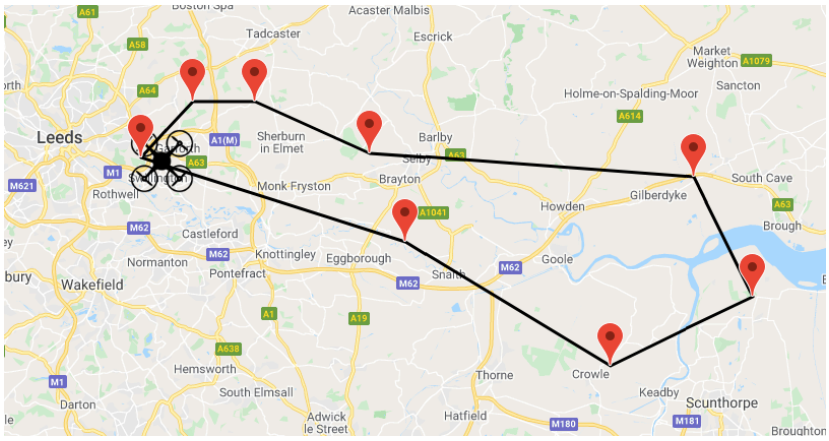

Figure 26 - Mission Planning Example

In addition to mission planning, and to provide users with as much flexibility as possible to make decisions in an emergency situation, DroneIO is also equipped with manual control features; allowing users to fully control their drones through the web-based interface.

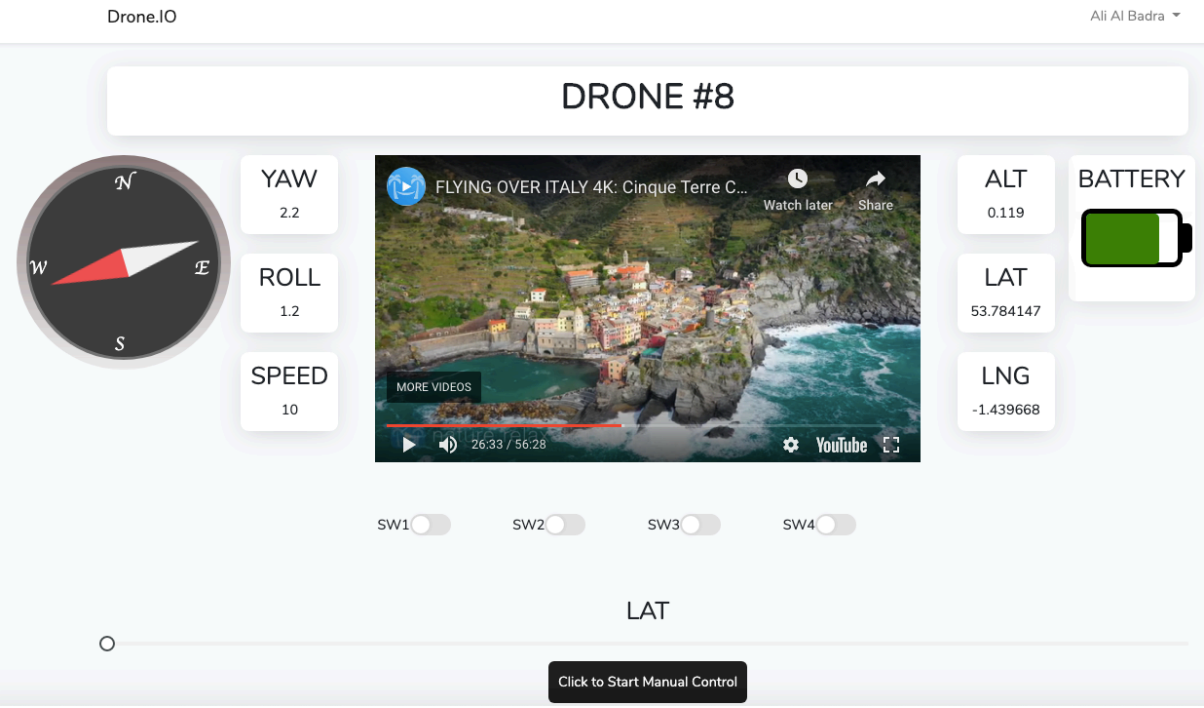

Figure 27 - Manual Control Webpage

While the main challenge with web-based manual control is latency, DroneIO utilizes WS in this feature for real-time data reporting to and from the drone; a purpose it was originally designed for. For demonstration purposes, a YouTube video has been embedded into the webpage, however, if this project were to be deployed then the video would either be replaced by a live YouTube feed streamed from the drone, or a direct stream through the WS channel.

4. Client Application

Following the architecture defined in figure 8, drones must be able to initiate a connection with the server, as well as follow a set of rules when registering, sending, and processing data. To facilitate such behaviour, a simple program was developed to run on each drone; managing subs-system activity and organizing data flow within each drone. This program runs locally on the drone's RPI and each drone must have a copy of the program available in order to be integrated into the system. This program, traditionally called a client application in the client-server model [28], was developed using Node.JS; a JavaScript framework used in the development of

client-server application and is known for being an efficient and lightweight framework [29][30]. Additionally, a large number of libraries are available for Node applications; easing the task of processing different types of data and communicating with the different sub-systems.

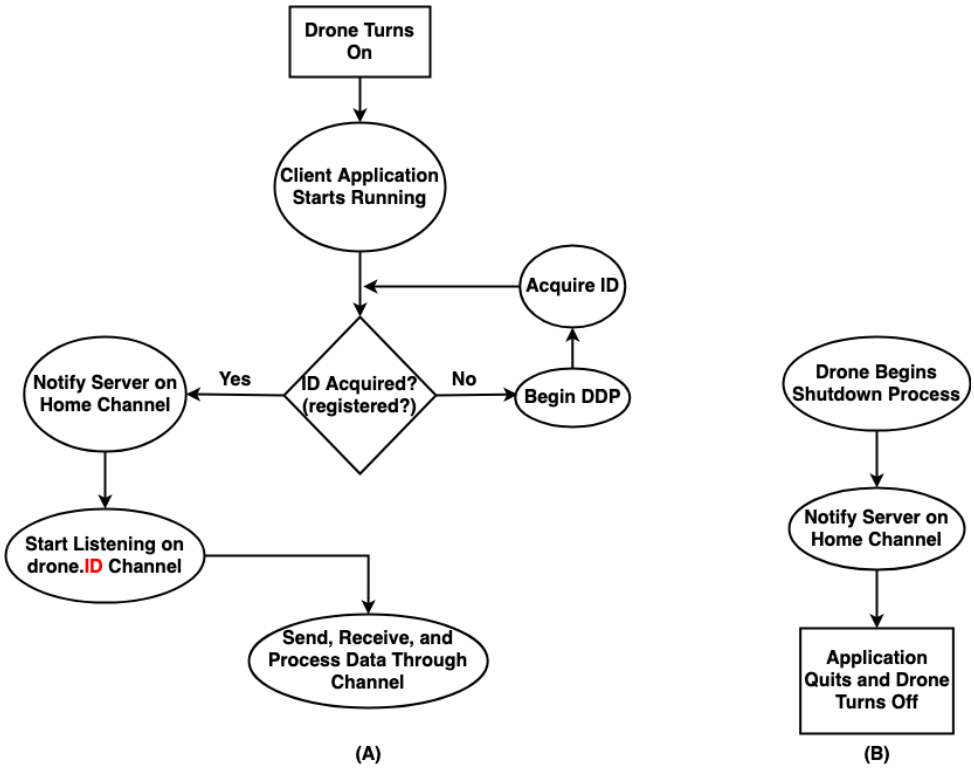

Figure 28 - Client application workflow. (A) Start-up process. (B) Shutdown Process.

Fig. 28 gives an overall idea of the sequence of events occurring at the client application. Most of the intensive processing occurs at the final stage of (A) where the application starts processing server requests; either sending or receiving data. This involves connecting and querying the database for data, encoding data into JSON format so that it can be transferred as efficiently as possible and in a format readable by the server. Finally, in case the shutdown process is initiated, a notification is first sent to the server notifying it of the shutdown before quitting the application.

In some cases, drones might enter zones where network coverage is either minimal or non-existent; causing a loss of connection. This is a fairly common issue with drones as they cover large areas of land. To combat this issue and provide a failsafe mechanism, the server keeps a timer running in the background that then gets compared with the time at which the last message was received. If no messages are received the timer increments indefinitely. Meanwhile, a simple if-statement runs in background to ensure the value of the timer is not past a certain threshold. This

threshold has been set at 3 seconds, if a drone doesn't send a message in 3 seconds then an error is signalled and a useful error message is displayed to the user. The server will also send requests every 5 seconds in an effort to reconnect.

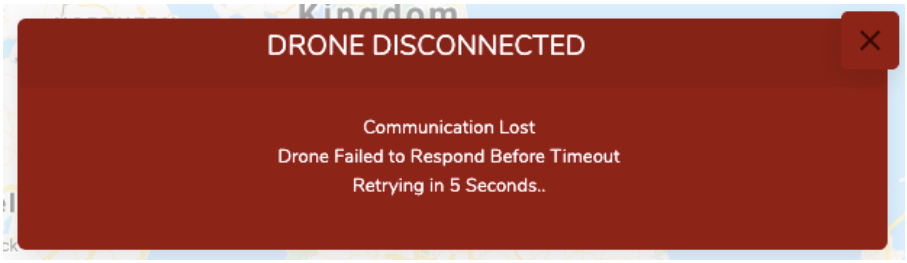

Figure 29 – error message upon disconnect

Once the drone succeeds in reconnecting, it will begin to send its current data as it receives the server request. Lost data is maintained in the local database and can be retrieved manually at a later time.

5. User Management

An important aspect to consider when designing a website is the targeted user-base, whether they are technically advanced personnel, recipients of special training, or inexperienced users; as this helps developers when making design choices. For example, websites designed for the general public, such as Facebook and YouTube, are known for their simplistic designs and abundance of support available, while limiting user options. Meanwhile, designers of an application like GitLab expect their users to have some technical background, which allows them to include a range of features; giving users a greater sense of flexibility and control.

Regarding DroneIO, the main users of the system are expected to be the experienced Unmanned Aerial Vehicle (UAV) pilots within fire brigades, with the possibility of newly trained firefighters using the platform during times of emergency. Just as manual control in the public requires a pilots' license, controlling one via a web-based interface should follow similar guidelines. Finally, the platform is designed for a specific purpose, where many of its features are not meant to be controlled or accessed by the general public; further narrowing down the user-base. Some simple features, however, could be made available to the public, such as a vehicle information page and a map view of some drones for demonstration purposes only.

In order to manage user access to certain features and drones, a user privileges system was adopted. The privileges system identifies user account types with a set of permissions for each; controlling access in a pragmatical way.

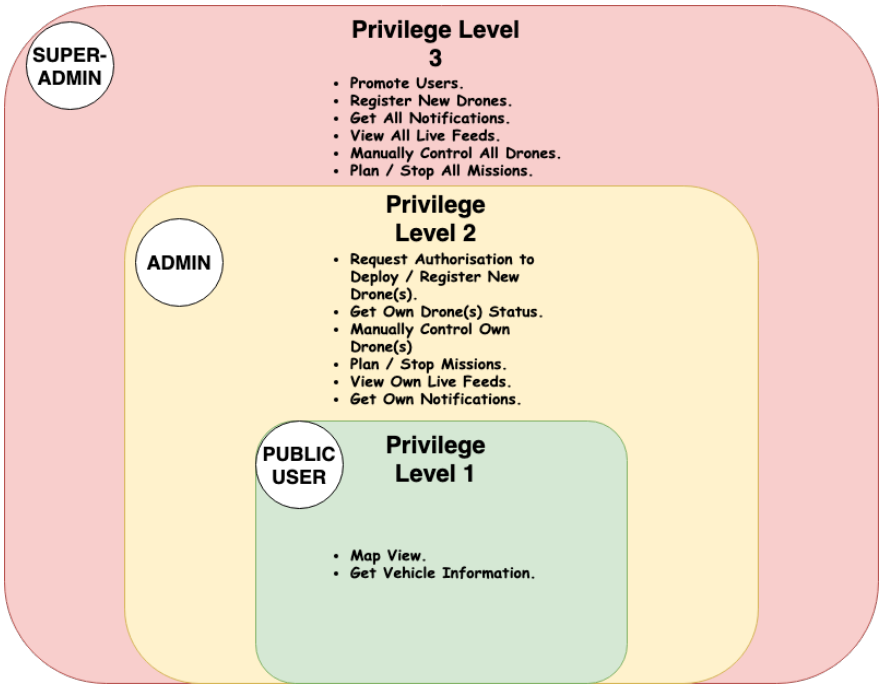

Figure 30 – Hierarchical view of the privileges system

The three account types defined in this hierarchical system are shown in Fig. 31, with public users having high restrictions, while a super-admin overlooks the whole platform and has access to all features and drones. A super-admin account is a single account, hard-coded into the database to give the overlooking authority full control over the platform in case of an emergency.

The privileges system has been adopted from the Laravel-permission package which provides an API to control user roles and permissions directly from a PHP script. Essentially, the package creates a set of tables in the database, much like the set in Fig. 20, through which it maintains special relationships between each table; ultimately allowing developers to associate certain account types with a set of permissions.

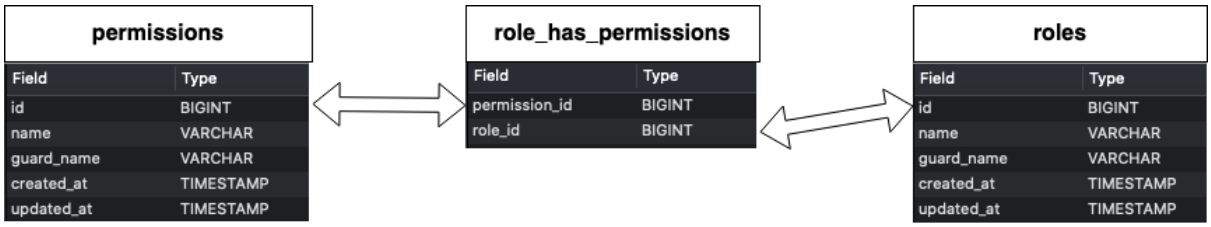

Figure 31 - Database relations behind the privileges system

In case a public user, for example, tries to access a feature he or she is not authorized, a meaningful error message is displayed, as shown in Fig. 32.

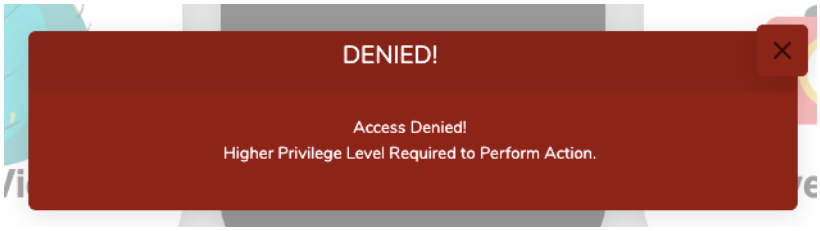

Figure 32 - Access denied error message

5. System Performance Analysis

Analysing system performance is an important aspect when designing any system, as it helps developers make certain design choices with the data to back it. Distributed systems, in particular, require vigorous testing to ensure all subsystems are well integrated with each other. Performance analysis also helps identify performance bottlenecks and asses system limitations, from network resources to computational power needs.

To analyse system performance, a number of test were carried out under different conditions to assess the system's response time, as well as its bandwidth consumption. Finally, a relationship between the two was derived to identify the conditions at which the system performs best, while maintaining efficient network usage.

5.1 System Response Time

The system's response time is affected by a number of factors, such as computational power, network quality, and available bandwidth. Moreover, in a complex system where some subsystems rely on outputs from other subsystems; certain performance bottlenecks might arise. A good example of a performance bottleneck within DroneIO can be seen in Fig. 18, where thermal processing relies on data published by the drone autonomy system. Thermal processing cannot operate at a rate faster than the rate at which drone autonomy operates; hence a performance bottleneck. Based on the same example, another bottleneck can be thought of as the rate at which the drone sends data to the cloud through the WS channel. Generally, the faster the subsystem operate, the more control users have of their drones, such as in the manual control mode. However, it is important to keep network resources in consideration when making such design choices; as a faster

rate leads to even higher bandwidth requirements. It is worth noting that cloud computing allows developers to dynamically change bandwidth requirements and computational power needed for their hosted server to meet the system's needs.
A number of tests were carried out to examine the effects on the WS channel caused by having multiple drones connected at one time. To quantize the effect, two measurements were taken, time taken for a data request-response through the channel, as well as the time taken to load up a full webpage. In all tests, the webpage and the contents of the request-response messages were kept constant.

5.1.1 WS Channel Performance

To accurately measure channel performance, the time at which a request is sent through the channel is first recorded, then compared with the time at which a response is received. Throughout this experiment, the number of connected drones, N, will be varied to observe the effect that has on the overall response time. All connected drones will be trying to communicate at the same time through their dedicated private channels. Individual channels' performance is expected to remain relatively unaffected by the increasing number of connected drones.
In each of the experiments, 200 sample samples have been taken and a resulting gaussian graph has been obtained. Each drone responds with the same message, keeping the size of transmitted data constant. The mean values were then calculated to draw up conclusions regarding the findings.

Experiment one: N = 1.

Average response time = 511.97 ms

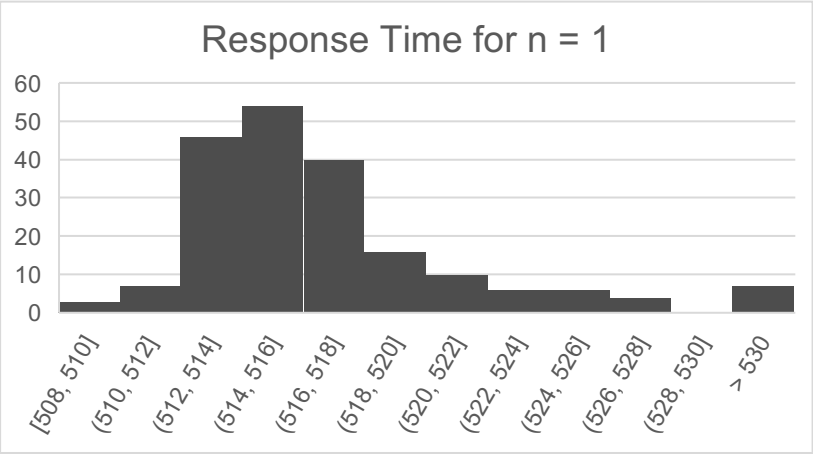

Figure 33 - Response time for N = 1

Experiment two: N = 2.

Average response time = 510.41 ms

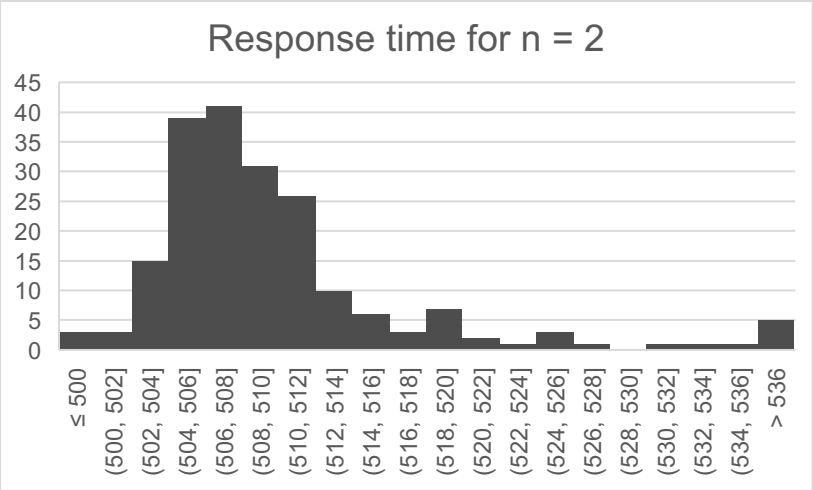

Figure 34 - Response time for n = 2

Experiment three: N = 3.

Average response time = 515.12 ms

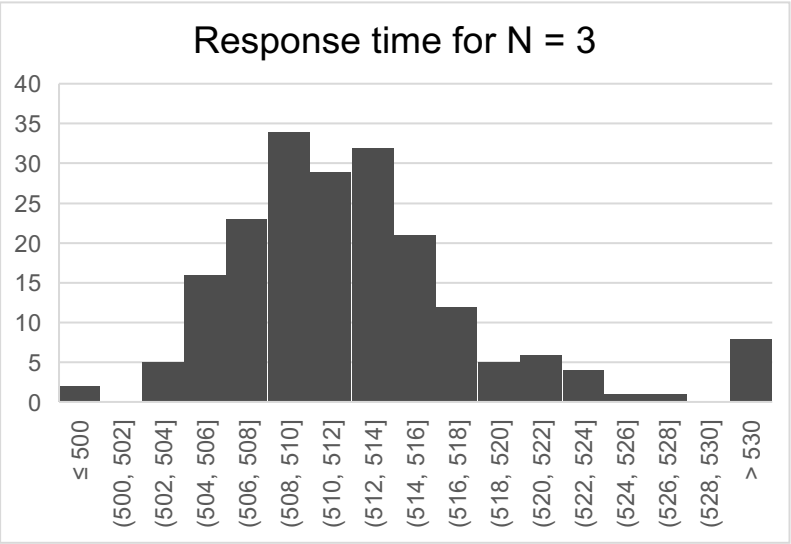

Figure 35 - response time for n = 3

Repeating the experiment for N = 4 through N = 10

Average response time for 10 > N > 1 = 512.17 ms

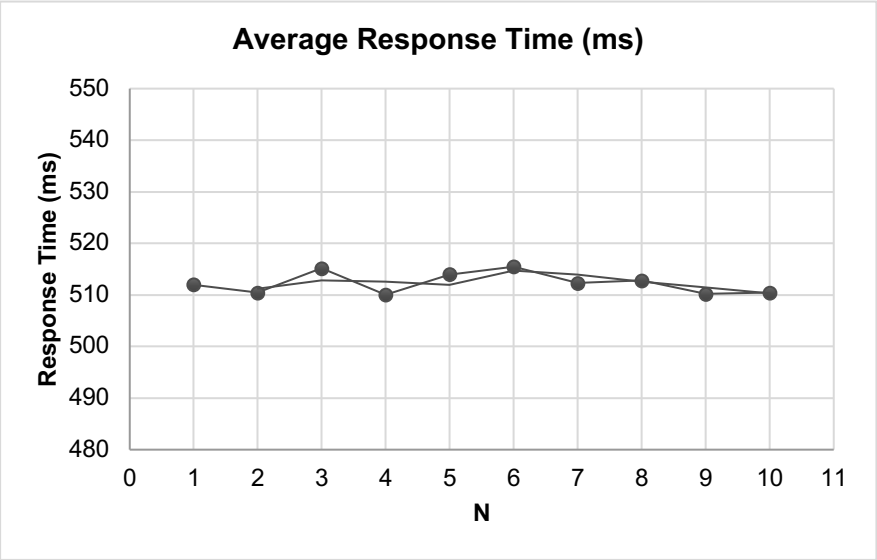

Figure 36 - overall average response time

As Fig. 37 depicts, the average response time of the system remains relatively unaffected by the number of active drones at one time. This relationship proves the system is behaving in a rather parallel manner; since each channel is independent of other channels and no collisions can occur at one channel. Therefore, the rate of transmission at any channel is unaffected by the activity of other channels, and since the size of data transmitted across all channels is equal the result showed a steady average response time.

5.1.2 Analysing Website Performance

An important factor when analysing a website's performance, along with user experience, is the loading time of a webpage. It is important to understand that while each drone communicates with the server through a private WS channel, all communication still passes through the same server port. As figure 8 illustrates, all drones eventually connect to a single server application; therefore as the number of connected drones increases, some processing delays are expected before fully loading a webpage.

To maintain an identical testing environment among all experiments, the webpage and type of content loaded remain the same. To accurately measure full-page load-time, Google Chrome DevTools were used.

Experiment one:

Observing the effect of increasing the number of active drones on the loading time for a fixed payload size of 1760 Bytes.

Table 1 - Average Page Load Time For Increasing N

N	Average Loading Time (ms)
1	871
2	929
3	906
4	1054
5	1063
6	1004
7	1088
8	1216
9	1264
10	1460

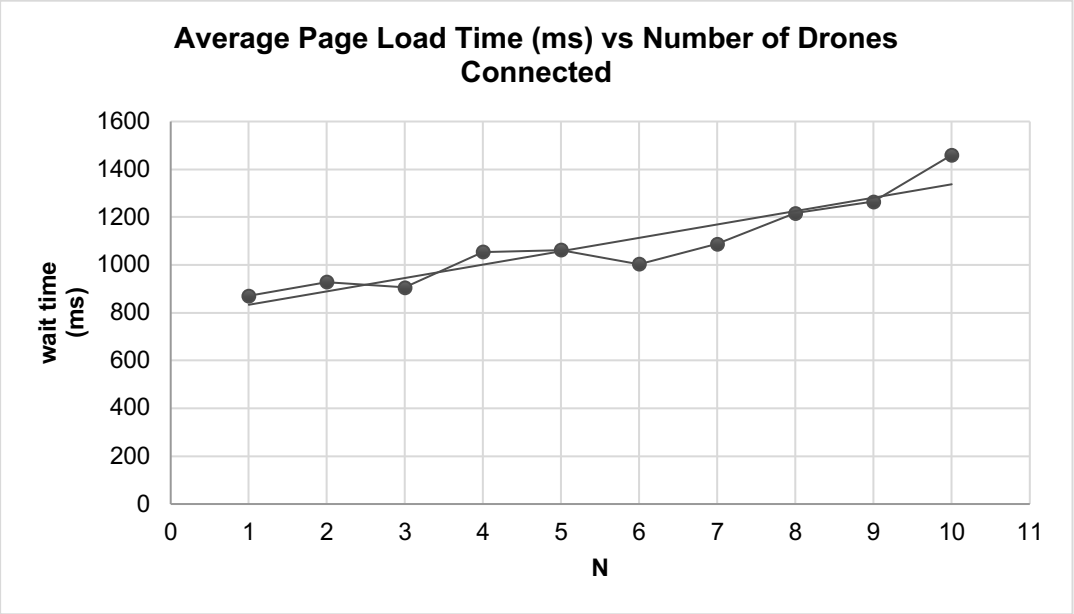

Figure 37 - Average Page Load Time (ms) vs Number of Drones Connected

As Fig. 38 depicts, the average loading time increases as the number of connected drones increases; this is only normal since the server now has a large amount of data coming in at the same time. Upon receiving data through the WS channel, the server begins its processing and rendering stages; extracting the useful bits of data and adding it to certain HTML sections to be displayed on the webpage. This process will naturally take longer as more data is received.

Loading time is affected by various factors, including the server's computational power, transmission rate, type of processing done, and the type and size of data received.

Since most of the above factors are limited by physical resources, the next experiment makes an effort to observe the effect on the loading time caused by reducing the size of data sent; through adding or omitting redundant data. For example, when requesting current location the server does not need to know drone's ID, nor its pitch or roll values. This makes efficient use of the communication without resorting to changing any of the network's physical resources.

When comparing the average page load time against an increasing payload size, it is important to understand that the extra bytes on their own are insignificant when compared to the server's processing power. Therefore, the experiment will aim to maintain a large amount of traffic passing through the server; so as to keep as much pressure on the server. This can be demonstrated by keeping N at a high value.

Experiment two:

Observing the effect of the payload size on the loading time of a webpage, for a fixed value of N = 10.

Table 2 - Average Page Load Time (ms) against Payload Size (Bytes) for N = 10

Payload Size (Bytes)	**Average Response Time (ms)**
66	1091
194	1211
346	1175
406	1125
512	1300
638	1203

722	1337
946	1233
1182	1262
1394	1399

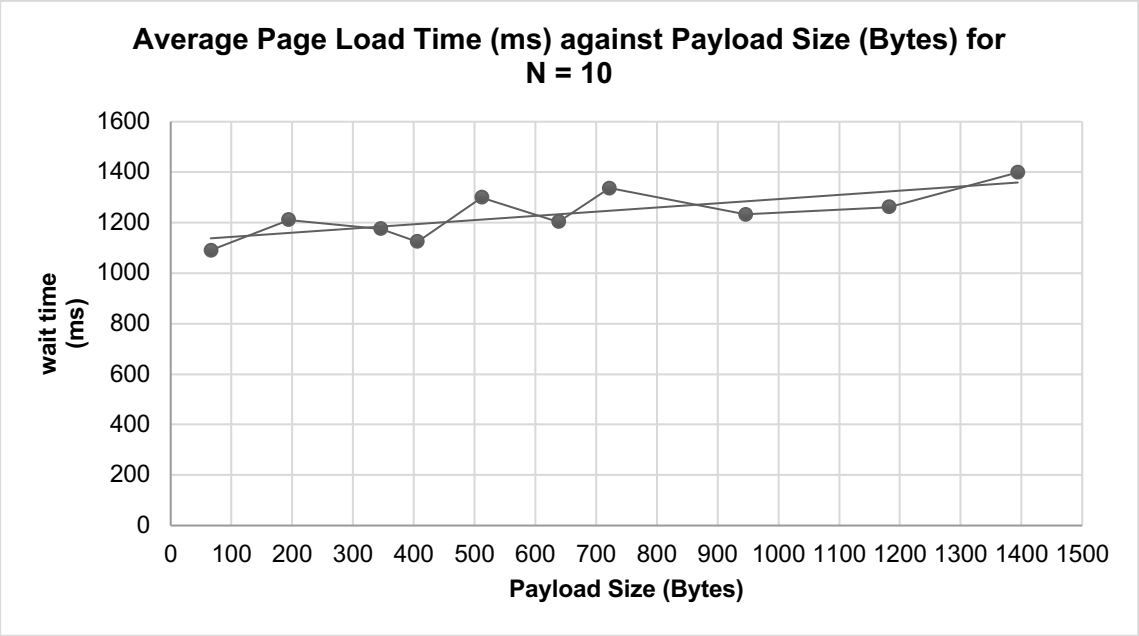

Figure 38 - Average Page Load Time (ms) against Payload Size (Bytes) for N = 10

According to Fig. 39, the overall trend indicates an increased loading time with increased payload at a fixed value of N. However, the difference between the load time at 66 bytes and that at 1394 bytes is very minimal, especially when considering units of milliseconds; this is likely due to the insignificant effect the size difference has on a local machine. However, in a deployment environment this effect is likely to be amplified as data moves through the internet; with different links of varying capabilities. Bandwidth, too, becomes an issue with the increased payload size in a production environment; a property that is insignificant on a local machine.

5.2 Bandwidth Usage Across WS Channels

In web application design, bandwidth has always been treated as the most valuable resource available, and one that must be used efficiently. The term bandwidth relates to the amount of data a communication link can transfer per units of time [31], usually measured in bits per second (bps). Generally, the higher the bandwidth available, the more data can be sent across a certain link. As mentioned earlier, applications hosted on the cloud offer flexibility to developers in that they can dynamically manage their available bandwidth depending on the system's needs.

To measure bandwidth usage across all WS channels, the total data transfer across the channels will be measured over a period of 10 seconds, the final answer bill be in bps and will represent bandwidth usage in a given scenario.

$$Bandwidth\ Usage\ = \frac{size\ of\ data\ transferred\ (bits)}{time\ (s)}\ [31]\ \ (2)$$

From the above equation, it can be deduced that bandwidth consumption is directly related to the amount of data transferred, as well as the rate at which data is sent. The following experiments will aim to form a relationship between the two factors; trying to reach a realization of the most efficient use of the system's bandwidth resources.

To have a sense of control on the transmission rate, a timer function is placed on the client application before sending a data response to the server.

The experiment will observe the effect of increasing the number of active drones, and consequently the number of open channels, on the overall bandwidth usage of the system.

Experiment 1.

Observing the effect of increasing N on the overall bandwidth consumption, for a set wait time(t).

Experiment 1.1: t = 1 s

Table 3 - Average BW Usage for Increasing N. t = 1 s

Number of Drones Connected	Average Bandwidth Usage - bps
1	2974
2	5933
3	8652
4	13450
5	15322
6	19829
7	19829
8	23434
9	25237
10	27040

18	41584

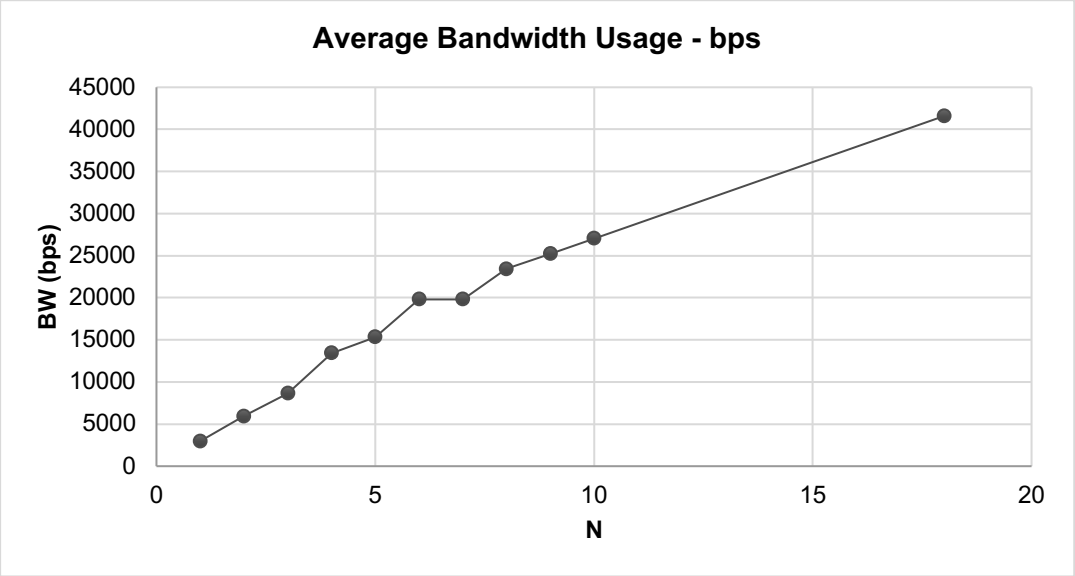

Figure 39 - Average BW usage for an increasing N, t = 1 s

Fig. 40 shows an almost linear relationship between N and bandwidth usage; a finding that is consistent with equation (2) above. Indeed, we expect such a relationship especially when both client and server applications are running on a local machine. In a real-world scenario, however, we might expect some discrepancy since network quality becomes an important factor. The overall trend is still expected to be increasing as N increases in all cases. When comparing bandwidth usage at N=1 and that at N=18, an increase by a factor of 14 is observed; this expected as each drone sends the same amount of data once connected.

Experiment 1.2: t = 3 s

Table 4 - Average BW Usage for Increasing N. t=3s

Number of Drones Connected	Average Bandwidth Usage - bps
1	991
2	1982
3	2974
4	3966
5	4957
6	5408
7	6940

8	7210
9	8923
10	9103
18	16452

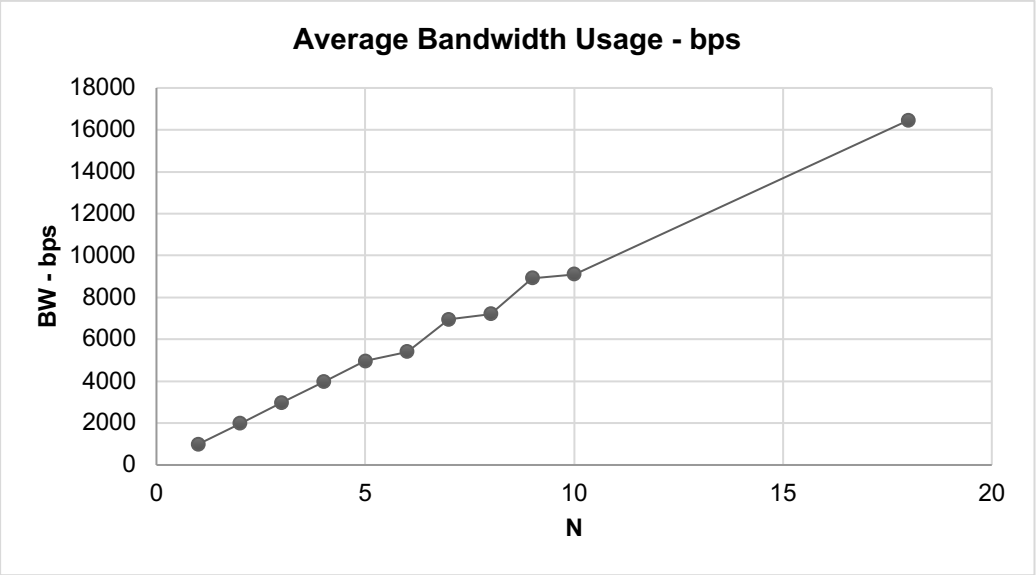

Figure 40 - AVERAGE BW USAGE FOR AN INCREASING N, T=3S

The relationship established earlier holds in Fig. 41, as expected. However, it is important to note the difference in bandwidth usage as the wait time increased. By comparing Table 4 with Table 3, an increase of 150% - 210% is observed; a large yet expected increase as the transmission rate more than doubled. Again, comparing bandwidth usage at N=1 and that at N=18 yields an increase factor of 16.

Experiment 2.

Observing the effect of increasing the transmission rate on the bandwidth usage while keeping N = 1.

Table 5 - Wait time vs BW Usage

Wait Time - s	Average Bandwidth Usage - bps
0.1	27040
0.2	13520
0.3	9013
0.6	4506
1	2974
3	991

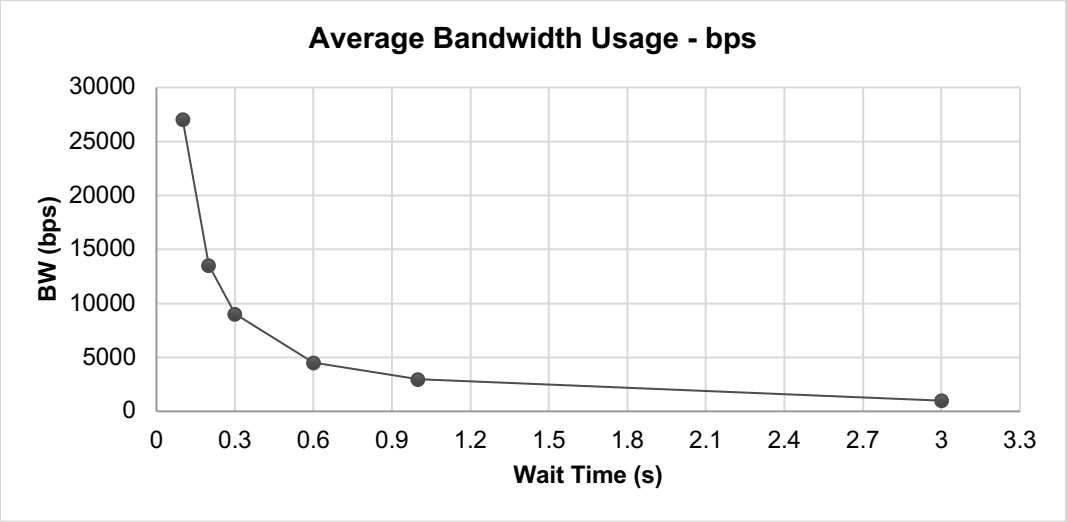

Figure 41 - Average bw usage for a varying transmission rate when N = 1

This experiment aimed to establish a relationship between bandwidth consumption and the transmission rate through varying the wait time; trying to push the channel to its maximum capacity. As the wait time decreases, the amount of data passing through the channel per second increases; hence bandwidth increases. Overall, when comparing the data of Table 5 as well as the trend in Fig 42, it can be seen that a slight change in wait time causes large changes in bandwidth consumption. When comparing bandwidth usage at t=0.1 s with that at t=1 s, an increase of 700% is observed. Moreover, when compared with t=3 s, bandwidth usage at t=0.1 s appears to have increased by 2,325%.

Another individual test was carried out where 18 drones were sending data at a wait time of 0.1s each, the resultant bandwidth usage in that scenario was 64894 bps; an increase by a factor of 2.4 when compared to the obtained consumption from Table 3 at t=0.1. This indicate that, while bandwidth usage increased as N increased from 1 to 18, it did not increase in the same way it did in experiments 1.1 and 1.2. where the increase factors were 14 and 16 respectively. Therefore, the graph is expected to increase at a much slower rate and eventually plateau; indicating channel capacity has been reached.

As those calculations have shown, bandwidth consumption is directly related to the rate of transmission; indeed the slightest decrease in wait time results in an almost exponential increase in bandwidth consumption until channel capacity is reached.

It is worth noting that the transmission rate cannot be increased indefinitely since both channel capacity and the server’s processing power are restricted by physical

resources. Further increase in the transmission rate has been observed to negatively affect the website's responsiveness as well as lead to occasional loss of client messages at the server. Therefore the above experiments operated at values where the server performance was not compromised.

5.3 Performance Analysis: Conclusion

Sections 5.2 and 5.3 tried to understand the relationship between bandwidth consumption, transmission rate, number of active drones, and channel capacity; ultimately reaching a realization of the most efficient configuration that maintains a high quality of service while keeping network resources at a sensible level.

By observing the results obtained from the experiments above, it can be generalized that a wait time of one second results in moderate bandwidth consumption, while maintaining real-time data reporting to the user for a high quality of service.

However, the number of connected drones will vary constantly as more users can be on the platform at the same time, resulting in a large amount of traffic across the communication channel and consequently increased bandwidth usages and pushing the channel to its maximum capacity. Conversely, if the number of active users at a given time is minimal, the channel will not be utilized at its maximum capabilities.

Therefore, a more robust and dynamic solution that modifies transmission rate based on the number of active users should be implemented. The proposed method is derived from the TCP congestion control [32]; a protocol used across all TCP connections that aims to reliably maximize channel performance by gradually increasing the transmission rate of packets, as seen in Fig. 43, until some packet loss is detected across the channel, much like negative feedback. Once packet loss exceeds a certain threshold, the sender drops transmission rate to half its previous value and gradually increases again; thus actively probing the channel for available bandwidth [33].

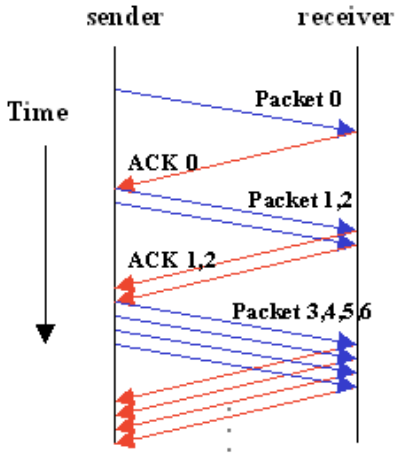

Figure 42 - TCP Congestion Control Increasing transmission rate.
Source: http://isi.edu

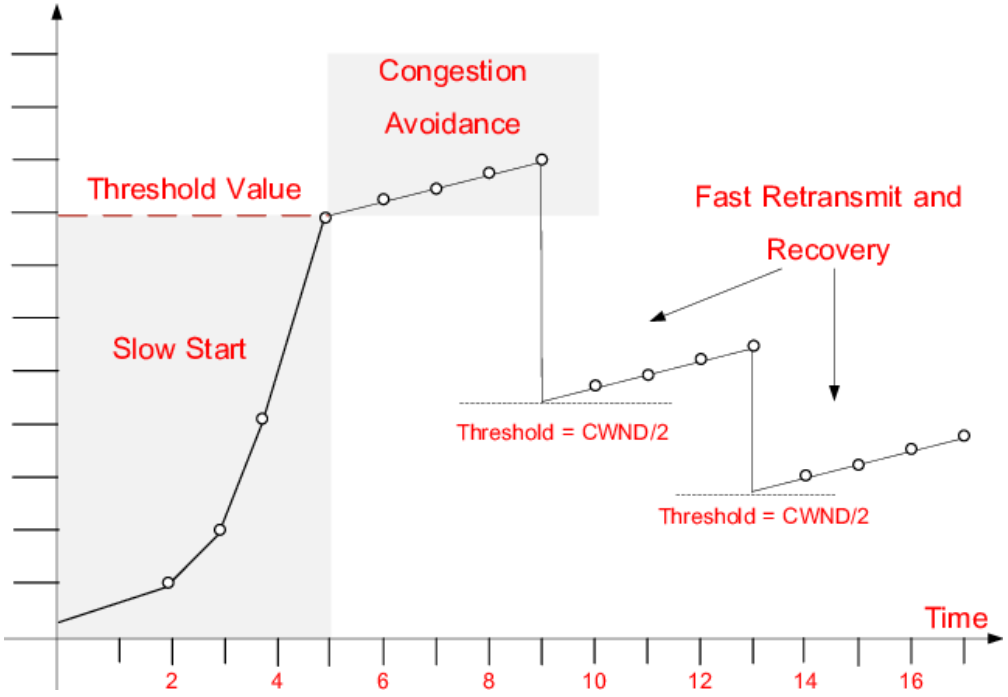

Figure 43 - TCP Congestion Control dropping transmission rate upon packet loss.
Source: http://researchgate.net

A similar method can be implemented in DroneIO, where drones probe the server by gradually increasing and decreasing the transmission rate until some negative feedback is received; thus utilizing channel capabilities and network resources while maintaining the highest possible quality of service under those conditions.

6. Conclusion

Current GCS applications cannot accommodate this project's objectives. Therefore, this chapter introduced the design of a cloud-based GCS featuring multi-drone map

view, heatmap view, mission planning, manual control over the web, and live feed streaming. The proposed platform is an improvement on generic GCS in that it offers flexibility; serving an arbitrary number of drones and users. WS protocol has been utilized throughout this project for efficient and bi-directional communication channels between drones and the hosted GCS application through the internet. This chapter also evaluated the platform's performance in regards to network resources consumption and user experience. Finally, a solution was proposed to further enhance the system's performance while maximizing resource efficiency.

This project has been designed with the intention of integrating it with cloud services, however, due to the recent Coronavirus Covid-19 epidemic a subscription to cloud services was not possible. As a workaround, and for demonstration purposes, all processing took place on a local machine with server and client applications running and exchanging messages just as they would with the cloud, but instead with a local address rather than a public URL. If this project were to be deployed; the usage of cloud services would be essential, but a local machine is sufficient for development purposes and has been treated as a cloud service for this paper.

References

[1] Azure.microsoft.com. 2020. *What Is Cloud Computing? A Beginner'S Guide | Microsoft Azure*. [online] Available at: <https://azure.microsoft.com/en-gb/overview/what-is-cloud-computing/> [Accessed 21 May 2020].

[2]J. Al-Karaki and A. Kamal, "Routing techniques in wireless sensor networks: a survey", *IEEE Wireless Communications*, vol. 11, no. 6, pp. 6-28, 2004. Available: 10.1109/mwc.2004.1368893.

[3]I. Akyildiz, Weilian Su, Y. Sankarasubramaniam and E. Cayirci, "A survey on sensor networks", *IEEE Communications Magazine*, vol. 40, no. 8, pp. 102-114, 2002. Available: 10.1109/mcom.2002.1024422 [Accessed 21 May 2020].

[4]D. Reina, S. Toral, F. Barrero, N. Bessis and E. Asimakopoulou, "The Role of Ad Hoc Networks in the Internet of Things: A Case Scenario for Smart Environments", *Internet of Things and Inter-cooperative Computational Technologies for Collective Intelligence*, pp. 89-113, 2013. Available: 10.1007/978-3-642-34952-2_4 [Accessed 21 May 2020].

[5]J. Kramer and O. Hazzan, "The Role of Abstraction in Software Engineering", *ACM SIGSOFT Software Engineering Notes*, vol. 31, no. 6, pp. 38-39, 2006. Available: 10.1145/1218776.1226833 [Accessed 21 May 2020].

[6]"Aeronautics", *Grc.nasa.gov*, 2020. [Online]. Available: https://www.grc.nasa.gov/www/k-12/UEET/StudentSite/aeronautics.html. [Accessed: 21- May- 2020].

[7]J. Reid, "Understanding Kv Ratings - RotorDrone", *RotorDrone*, 2020. [Online]. Available: https://www.rotordronepro.com/understanding-kv-ratings/. [Accessed: 21- May- 2020].

[8]"A Guide to Understanding Battery Specifications", *Web.mit.edu*, 2008. [Online]. Available: http://web.mit.edu/evt/summary_battery_specifications.pdf. [Accessed: 21- May- 2020].
[9]"What is Cloud Computing", *Amazon Web Services, Inc.*, 2020. [Online]. Available: https://aws.amazon.com/what-is-cloud-computing/. [Accessed: 21- May- 2020].
[10]S. Loreto, P. Saint-Andre, S. Salsano and G. Wilkins, "Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP", p. 18, 2011. Available: https://www.hjp.at/doc/rfc/rfc6202.html. [Accessed 21 May 2020].
[11]L. Chappell, "Inside the TCP Handshake", p. 2, 2000. Available: http://masimoes.pro.br/site/redes/51_TrTCP/05-tcp_files/tcphandshake.pdf. [Accessed 21 May 2020].
[12]D. Skvorc, M. Horvat and S. Srbljic, "Performance evaluation of Websocket protocol for implementation of full-duplex web streams", *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2014. Available: 10.1109/mipro.2014.6859715 [Accessed 21 May 2020].
[13]V. Pimentel and B. Nickerson, "Communicating and Displaying Real-Time Data with WebSocket", *IEEE Internet Computing*, vol. 16, no. 4, pp. 45-53, 2012. Available: 10.1109/mic.2012.64 [Accessed 21 May 2020].
[14]K. Kilbride-Singh, "WebSockets vs Long Polling", *ably*, 2019. .
[15]A. Rahmatulloh, I. Darmawan and R. Gunawan, "Performance Analysis of Data Transmission on WebSocket for Real-time Communication", *2019 16th International Conference on Quality in Research (QIR): International Symposium on Electrical and Computer Engineering*, 2019. Available: 10.1109/qir.2019.8898135 [Accessed 21 May 2020].
[16]Q. Liu, G. Yang, R. Zhao and Y. Xia, "Design and Implementation of Real-time Monitoring System for Wireless Coverage Data Based on WebSocket", *2018 IEEE 3rd International Conference on Cloud Computing and Internet of Things (CCIOT)*, 2018. Available: 10.1109/cciot45285.2018.9032640 [Accessed 21 May 2020].
[17]L. Zhang and X. Shen, "Research and development of real-time monitoring system based on WebSocket technology", *Proceedings 2013 International Conference on Mechatronic Sciences, Electric Engineering and Computer (MEC)*, 2013. Available: 10.1109/mec.2013.6885373 [Accessed 21 May 2020].
[18]"Laravel WebSockets 🐝", *Docs.beyondco.de*, 2020. [Online]. Available: https://docs.beyondco.de/laravel-websockets/. [Accessed: 21- May- 2020].
[19]"What is a database?", *Oracle.com*, 2020. [Online]. Available: https://www.oracle.com/uk/database/what-is-database.html. [Accessed: 21- May- 2020].
[20]"Introduction to Oracle Database", *Docs.oracle.com*, 2020. [Online]. Available: https://docs.oracle.com/cd/E11882_01/server.112/e40540/intro.htm#CNCPT88782. [Accessed: 21- May- 2020].
[21]M. Ahmed, M. Uddin, M. Azad and S. Haseeb, "MySQL performance analysis on a limited resource server", *Proceedings of the 2010 Spring Simulation Multiconference on - SpringSim '10*, 2010. Available: 10.1145/1878537.1878641 [Accessed 21 May 2020].
[22]"MySQL :: MySQL Customers", *Mysql.com*, 2020. [Online]. Available: https://www.mysql.com/customers/. [Accessed: 21- May- 2020].
[23]"What is Heroku | Heroku", *Heroku.com*, 2020. [Online]. Available: https://www.heroku.com/what. [Accessed: 21- May- 2020].
[24]"What is Web Hosting? - Web Hosting - Beginner's Guide | Website.com", *Website.com*, 2020. [Online]. Available: https://www.website.com/beginnerguide/webhosting/6/1/what-is-web-hosting?.ws. [Accessed: 21- May- 2020].
[25]"Overview | Maps JavaScript API | Google Developers", *Google Developers*, 2020. [Online]. Available: https://developers.google.com/maps/documentation/javascript/tutorial. [Accessed: 21- May- 2020].
[26]D. Floreano and R. Wood, "Science, technology and the future of small autonomous drones", *Nature*, vol. 521, no. 7553, pp. 460-466, 2015. Available: 10.1038/nature14542 [Accessed 21 May 2020].
[27]N. Jayapandian, "Cloud Enabled Smart Firefighting Drone Using Internet of Things", *2019 International Conference on Smart Systems and Inventive Technology (ICSSIT)*, 2019. Available: 10.1109/icssit46314.2019.8987873 [Accessed 21 May 2020].
[28]H. Oluwatosin, "Client-Server Model", *IOSR Journal of Computer Engineering*, vol. 16, no. 1, pp. 57-71, 2014. Available: 10.9790/0661-16195771 [Accessed 21 May 2020].
[29]L. Chitra and R. Satapathy, "Performance comparison and evaluation of Node.js and traditional web server (IIS)", *2017 International Conference on Algorithms, Methodology, Models and*

Applications in Emerging Technologies (ICAMMAET), 2017. Available: 10.1109/icammaet.2017.8186633 [Accessed 21 May 2020].
[30]S. Tilkov and S. Vinoski, "Node.js: Using JavaScript to Build High-Performance Network Programs", *IEEE Internet Computing*, vol. 14, no. 6, pp. 80-83, 2010. Available: 10.1109/mic.2010.145 [Accessed 21 May 2020].
[31]R. Prosad, C. Davrolis, M. Murray and K. Claffy, "Bandwidth estimation: metrics, measurement techniques, and tools", *IEEE Network*, vol. 17, no. 6, pp. 27-35, 2003. Available: 10.1109/mnet.2003.1248658 [Accessed 21 May 2020].
[32]A. Afanasyev, N. Tilley, P. Reiher and L. Kleinrock, "Host-to-Host Congestion Control for TCP", *IEEE Communications Surveys & Tutorials*, vol. 12, no. 3, pp. 304-342, 2010. Available: 10.1109/surv.2010.042710.00114.
[33]Ningning Hu and P. Steenkiste, "Evaluation and characterization of available bandwidth probing techniques", *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 6, pp. 879-894, 2003. Available: 10.1109/jsac.2003.814505.