Imperial College London

Department of Electrical and Electronic Engineering

Final Year Project - Final Report 2017

Project Title:	**Distributed Road Traffic Speed Monitoring**
Student:	**Jeremy Chan**
CID:	**00818433**
Course:	**4T**
Project Supervisor:	**Dr. Ed Stott**
Second Marker:	**Dr. Christos Papavassiliou**

Abstract

Speed cameras have been shown to slow traffic down and prevent accidents. Specifically, average speed cameras are more effective than fixed position cameras as they prevent speeding after the motorist has passed the camera. However, around 84% [1] of the UK's average speed cameras are placed on busy trunk routes as it is not economically feasible to place them in rural areas. Moreover, the design of traditional speed cameras means their widespread deployment on residential roads would be detrimental to the streetscape.

This project therefore enables the use of average speed cameras in any area. It is a low-cost, distributed vehicle speed monitoring system that communities can install themselves within private property next to a street. The system enables civilians to use their own personal cameras to detect and share license plate information by leveraging local binary pattern and optical character recognition computer vision algorithms. Plates, along with time and location data, are directly broadcasted to other peers in the network without the need for a central server. Violations are detected based on the average speed method using public mapping data, and users notified if an infraction occurs. Simulations, along with real world testing, show positive results under multiple lighting and camera conditions.

[1] 'Number of speeding convictions from average speed cameras - a Freedom of Information request to Driver and Vehicle Licensing Agency', *WhatDoTheyKnow*, 28-Jul-2015. [Online]. Available: https://www.whatdotheyknow.com/request/number_of_speeding_convictions_f. [Accessed: 22-Jan-2017].

Acknowledgements

I would like to express my heartfelt gratitude to my project supervisor, Dr Ed Stott, for his enthusiastic guidance, encouragement and useful critiques of this final year project.

- Parents
- Friends
- Other imperial staff

Nomenclature

-list of symbols here-

UPDATE ALL TITLES TO BE Like This Case

Cross refernce all figures, code, tables

Do acknowledgements, nomenalcture

Find suitable cover page photo

Modify project spec

Change everything to past tense

Table of Contents

1 Introduction

1.1 Motivation

The number of people speeding on the roads in the UK have actually decreased over the past 2 years [1] as a result of there being more speed cameras being installed across the country [2]. From [3], it is clear that 'the higher the speed, the greater the probability of a crash'. Hence, it is in the best interest of the authorities to monitor speeding convictions using average speed check cameras.

However, most of the average speed check cameras across the UK are installed in busy trunk routes, i.e. motorways. They use automatic license plate recognition technologies to track the entry and exit times. This data is not accessible for the general public and is kept for two years [4]. This project aims to remedy that, given that many countries have supported the idea of using license plate readers and that there should not be any restrictions surrounding who and when can license plates be collected [5].

There are also very little amounts of speed cameras in rural and local roads. By using a decentralised network that can find out peers in its vicinity, anyone can download, compile, and use this project as a pseudo-speed camera in their local area, removing the need for expensive cameras and reliance on a closed database of license plates that only the police have access to. Anyone caught speeding will have their license plate posted onto a social network. Given that there is a set procedure that police use in issuing speeding tickets, the end goal is just to have the evidence on the social networks and not to actually ticket the offender.

1.2 Project Specification

From the project description, the goals of the project can be separated into the following:

1.2.1 Project goals

- Implement a number plate recognition system using existing computer vision algorithms on a low-cost, readily available hardware platform.
- Set up a peer-to-peer network to share vehicle passing times and detect violations without the need for a central server.
- Publish photo evidence of any violations

1.2.2 Advanced project goals

- Use the changes in the number plate geometry as the vehicle passes to detect the instantaneous speed of a vehicle. This provides a stand-alone mode that will aid adoption in areas where there isn't already an established network.
- Implement automatic peer discovery so that each device can find its neighbours and calculate the minimum legal transit time between them using a public mapping database.
- Add an encryption layer so that a hacker or rogue peer cannot use the network to track the movements of law-abiding vehicles.
- Package the system so that it can be easily installed in a home by an inexperienced user.

1.2.3 Clarifications

After discussions with Dr. Stott, the following clarifications were made:

1. The system should target license plates and deployment in the United Kingdom, and license plates of the UK format [6]. Custom license plates will not be in the scope of the project for simplicity.
2. The number plate recognition system should be targeted at an off the shelf package, so there should be minimal setup and calibration done. This also means anyone, with the right equipment, should be able to download and compile the system if they have existing hardware.
3. The low-cost, readily available hardware platform will be a Raspberry Pi (RPi), with a camera attached to it. Using an RPi combines the best of cost (~£40 at time of writing), power (quad core CPU [7]), flexibility (camera can be any USB webcam or RPi's official cameras), and support (development work on the RPi is extensive and there are ample tutorials/information online).
4. The peer to peer network should ideally be fully decentralised, so the system should be able to find peers without the help of a central server.
5. Publishing photo evidence will most likely be done onto a social network.
6. The public mapping database will be one accessible to most people – Google Maps API. There is a speed limit API but it is only open to premium users (https://developers.google.com/maps/documentation/roads/speed-limits).
7. A hacker or rogue peer should not be able to extract license plates from the system remotely, nor should they be able to spoof detect license plates as another user to avoid framing an innocent driver.
8. If possible, there should be a whitelist of emergency vehicles in the case of there being an emergency vehicle over the speed limit.

9. The final package should be a software package uploaded onto GitHub, with clear instructions on how to compile and run the program with examples. As this project involves testing on hardware as well, a list of recommended hardware should also be provided (after successful testing), so the project can be easily replicated in the future.

2 Background and Related Work

Research into already existing products was done on the three main goals of the project, to judge the market feasibility of the project, and to prevent overlap with existing work where possible.

2.1 License plate recognition

2.1.1 Hardware

There are many license plate recognition systems available on the market, with many of them being used in speed cameras around the country [8]. Traffic cameras can generally be separated into radar based and optical based systems, with radar based cameras checking the instantaneous speed of the vehicle as it moves past, and optical based systems checking the average speed of the vehicle as it passes between two points. There has been a steady increase in average speed check cameras over the years [2], mostly along motorways but also by local councils [9]. These average speed check cameras often include multiple enticing features like 24/7 operation and 4G connectivity, as such in Jenoptik's VECTOR cameras [10].

2.1.2 Software

Most of the license plate recognition software available is proprietary and closed source. Some are only available for commercial use. These systems generally have features such as video stream processing, ability to detect plates from multiple points of view, and cloud services to take in a video stream and output detected license plates along with their timestamps [11]–[13]. However, the price of the

commercial systems (if available for purchase) are definitely out of budget for this project, as shown in Table 1. ARES (http://platesmart.com/) did not respond to an enquiry regarding cost of usage, so is not shown.

As the project specification specifies that 'existing computer vision algorithms' should be used, the choice was made to use the consumer version of OpenALPR. The main reasons for this was that it was open source, and very feature rich. Additionally, a choice was made to instead replicate the commercial features of OpenALPR using computer vision processing libraries instead of ignoring the extra features or paying for them as they were standard vision processing techniques, e.g. background subtraction, motion detection.

Table 1: Prices of different commercial license plate detection systems

System	Price (one-off)	Price (per month)
DTK LDR SDK	190 – 1430 EUR	N/A
OpenALPR	1000 USD	50 USD

2.2 Peer to peer network

Peer to peer networks (P2P) have existed for a long time and there are numerous implementations available for use on the Internet. The main uses for peer to peer networks are web, messaging, and file sharing.

Bitmessage (https://bitmessage.org) is used for encrypted messaging, and is decentralised. It also uses strong authentication to prevent spoofing of the sender of a message [14]; however, it removes information about the sender and receiver.

It is therefore not suitable for this project as this system needs to know information about the sender to accurately work out an average speed.

Telehash (https://github.com/telehash/telehash.github.io) is open source, fully end-to-end encrypted, enforces strict privacy rules, and cross platform. It also supports JSON message sending with unique sender and receiver ID's. Unfortunately, the development is focused around using Node.JS and not Python or C++ (the Python binding repository is still empty at time of writing). Therefore, it is not suitable as it would take too much time to implement.

2.3 Photo evidence publication

There are official API's for most social networks on uploading images to their respective social networks so there is no need to reinvent the wheel, no is there a need to use a third party service to upload images.

2.4 Overall Feasibility

Overall, the project is very feasible as there has been a lot of work in all three areas, proving that the ideas in the project are not a dead end. Moreover, there is a free and open source implementation for the license plate recognition system which is immensely useful in providing a solid head start in license plate detection, a primary goal of the project. As most advanced features are behind a commercial paywall, a novel aspect of the project is to implement new algorithms / research to better improve the detection rate of the license plate detection.

The network is also a very feasible part of the project. P2P networks have extensive research and Python has libraries which support development using web sockets and different communication protocols. However, since the license plate detection is done using existing libraries, there has to be extra emphasis placed on the networking side to make sure this is the primary novel aspect to the project. At the time of writing, there has not been any projects tying license plate detection to a decentralised network which seeks out peers in its immediate vicinity.

Lastly, there are extensive documentation on uploading images to social networks – no foreseeable problem there unless the API access is revoked.

3 Background Reading and Literature Review

3.1 License plate recognition

Most license plate recognition systems operate on a similar basis. Pre-processing, detection, and character recognition [15]. However, image processing and detection can used interchangeably out of order, as in the case of OpenALPR [16].

In the pre-processing stage, a combination of techniques can be used to make the image more recognisable to the following pipeline stages. Azad et al. [17] utilises the HSV colour space instead of the RGB colour space to better determine the location of the plate as the plate is assumed to be of a certain colour. Some implementations implement both the RGB and HSV colour space to get saturation and intensity maps [18]. Duan et Al. [19] first converts the image to greyscale, then

normalises the histogram to perform histogram equalisation to get a picture with better contrast. This is especially important during the night when there are a lot of dark areas from shadows. An example is shown below in Figure 1.

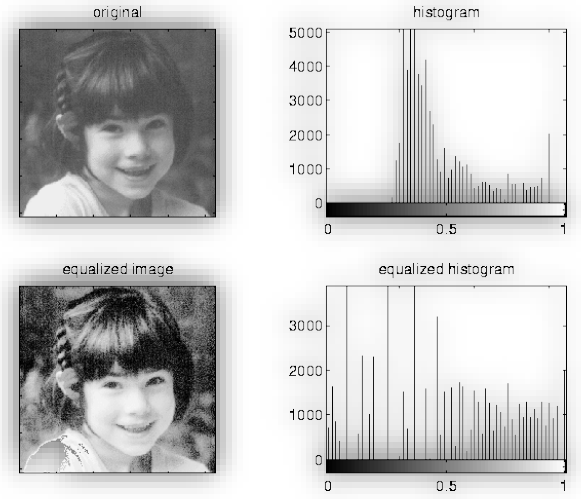


Figure 1: Histogram Equalisation (http://electronicsinourhands.blogspot.co.uk/2012/10/histogram-equalization-in-image.html)

In the detection phase, several algorithms are prominent. Edge detection is used extensively in Badr et al. [20], where the Sobel edge detection method is used in conjunction with thresholding techniques to determine the plate's location in areas with high vertical lines. However, these methods often fail if the assumption that the license plate is captured from a fixed face-on angle is false as the plate can no longer be guaranteed to have perfectly vertical lines. Moreover, using edge detection as the first step often brings false positives, where buildings and road signs can also exhibit perfectly vertical lines – however they are not an area of interest for the license plate detection system.

Hence, other implementations such as OpenALPR's and Liu et al. [21] use an algorithm called Local Binary Patterns (LBP), generally used in face recognition, to get areas of the image where it thinks there is a license plate. However, since LBP uses a sliding window approach, the performance of the detection is quite CPU dependent [16] (can be accelerated using a CUDA but the RPi does not support this). This method has been proven by Nguyen and Nguyen [22], who successfully used LBP with extra classifiers and algorithms in a real time license plate detector.

Edge detection is not unused, however. The most often used combination is a mixture of edge detection algorithms (Sobel, Laplacian, Canny), the Hough transform, and the contour algorithm. The Hough transform is used in implementations like [19] and OpenALPR [16] to find the actual edges of the plate as detection only gives a rough area for the location of the plate, and may end up with an area that does not have a plate in, or slightly bigger than the plate.

Other implementations such as Kwaśnicka and Wawrzyniak [23], and Chang et al. [18] use a technique called connected component analysis before applying the Hough transform. This technique is applied to a binary image of the license plate, and rejects any connected components whose aspect ratio is outside a prescribed range (a license plate is rectangular). The reliability of each implementation has yet to be tested against each other, however.

Lastly, the plate area needs to be warped to be a rectangle for the Optical Character Recognition (OCR) algorithms to work their best. A perspective transformation is often used to warp the image – the final image looks like it has been taken from another perspective (Figure 2). Moreover, straight lines remain straight [24] after

the transformation so the next stage, character recognition, is not compromised. OCR libraries such as tesseract (https://github.com/tesseract-ocr) are then used to get the alphanumeric characters from the plate.

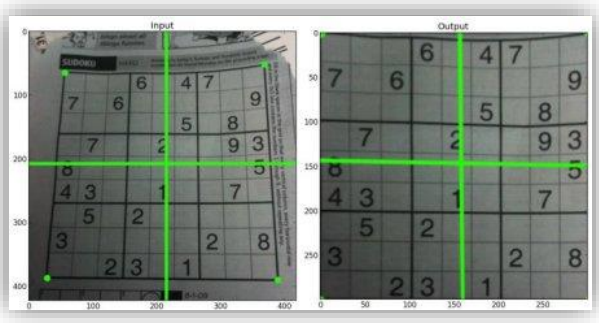


Figure 2: Using a perspective transformation [24]

3.1 Peer to peer network

A P2P network is a network where many machines connect in an ad-hoc manner (they do not join and leave the network based on a schedule), and serve as a server for others. This means there is no need for files and messages or whatever the network is serving to pass through a central server, providing no one single point of failure.

P2P networks account for a significant portion of the web's traffic [25], but its widespread nature means it comes with a few downsides as well. Making sure all copies of a file is free from corruption is one big challenge, as well as security. As all machines are servers of their own, a malicious hacker may attempt to distribute a virus or malware over the network to the many users using the network.

In this project, the main area of P2P networks that will be examined are decentralised networks. Centralised networks still have a central server that does routing, and peer finding, among other tasks [26], whereas decentralised networks do not.

Peer finding in a decentralised network is a challenge. As there is no central server, decentralised networks use techniques such as flooding the network with discover requests [26] and randomly pinging IP's around the world to see if they are part of the network [27]. A well-known decentralised network, Gnutella, uses another technique where it finds new peers by expanding its search radius in its broadcasting phase. When a peer accepts this connection, it rebroadcasts the new peer's address to its own peers, but decreases a counter called Time-to-Live (TTL) to ensure the message eventually dies out (when TTL=0) [28].

However, the broadcasting strategy in decentralised networks means as the search radius increases, there is an increasing amount of traffic in the network, and this may cause congestion in the network [28].

Other types of peer-finding techniques can be found in Mastroianni et al. [29].

3.2 Photo evidence publication

Image upload API's exist for many social networks, and the following table shows the documentation link for three popular image sharing sites.

Table 2: Photo upload API's

Network	Link
Facebook	https://developers.facebook.com/docs/php/howto/uploadphoto
Twitter	https://dev.twitter.com/rest/reference/post/media/upload
Imgur	https://api.imgur.com/endpoints/image

4 Design

The design section is used in this report to highlight the overall structure of the system, justify different design decisions given alternative implementations, and explain how the different modules of the system work together.

As this system is quite modular in nature, i.e. the system can be split into different modules with different roles rather clearly, the discussions will be done by module, as opposed to other groupings such as chronologically. Given the specifications of the system as shown previously, it was logical to split the system into 3 separate modules.

4.1 Required Hardware

From the requirements, a "low-cost, readily available hardware platform" is needed. Several products were considered, as shown in section XXX however the Raspberry Pi won out, given its wide support for linux applications (it runs Raspbian, a fork of Debian), wide support for peripherals (it has GPIO pins and USB inputs), and high performance (4 core CPU).

The minimum hardware required for this project is therefore:

1. A Raspberry Pi running Raspian
2. Any camera
    a. Preferably a camera that does not filter out infra-red, as an infra-red flash improves night time operation (XXXX increase)
    b. This project assumes the use of Raspberry Pi's official NoIR camera

Other hardware which would complement the project include:

1. A large SD card for storing video files
2. An infra-red flash for improved night time operation

4.2 High Level Overview

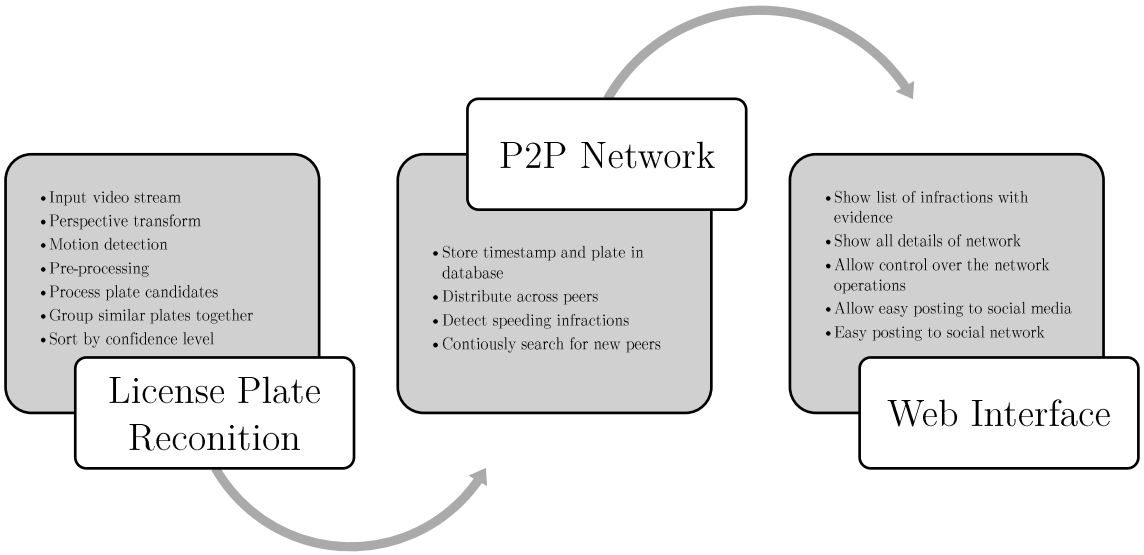


Figure 3: High level view of modules

Figure 3 shows the high-level flow of an instance of the system. The output of each module is fed into the input of the next module. Note that although the figure is depicts an instance of the system, the P2P network has only one global instance, and consists of many peers (not shown in the figure) – license plates are distributed to all peers.

First, real time input video is fed into the license plate detection algorithm after being pre-processed for noise and perspective changes. Secondly, the license plate

candidates for each frame of the image are tallied up, grouped into the different unique plates, before being sorted by confidence. Then, all these license plates are written into the database of the peer in the network, before being broadcasted to all the peers near itself. Lastly, if any of the broadcasted license plates match a local detected plate, the peer will automatically calculate the minimum legal transit time possible between the two peers (location data of the peer is also transmitted across the network) using public mapping APIs, before deciding if the vehicle was speeding. If the vehicle is determined to be speeding, the user of the system is notified through a web interface. The user can then easily upload a templated post onto social media with all the necessary details for prosecution (location of the two peers, the time of detection, the average speed determined by the network, and the two evidence images showing the car was indeed at that location at that time).

The following sections will focus on justifying the different design decisions taken for the final system, as well as give deeper details on the high-level design.

4.3 License Plate Detection

From reading related literature, license plate detection normally consisted of the following steps:

1. Get regions of a license plate
2. Execute line transforms and find the highest probability of a rectangular object and find the corners using corner transforms
3. Using the line and corner data, warp the license plate into a perfect rectangle for easy detection of the characters inside.

4. Using the high difference in contrast of the foreground and background of the license plate (black on white, black on orange), come up with the areas that each license plate character is in.
5. Feed these areas to an optical character recognition system, then join all the characters together to form the final license plate candidate.

As the project specification mentions using existing libraries, an initial design decision was to choose between implementing the license plate detection using existing computer vision algorithms for each step, or find an existing open source package which does all the steps above.

Given the timeframe of the project and that the novelty of the project was not in the license plate detection, the decision was to use an existing open source package as implementing all the steps using existing computer vision algorithms such as those from OpenCV would not only be very time consuming, but also replicating what open source license plate detection packages already do.

In the end, OpenALPR, was chosen for its ease of use, open source release, and its usability in having Python links to the underlying source code. Multiple other options were possible (see related work, section XXX), but were ultimately not used due to function or budget concerns. Moreover, the OpenALPR developer was active on GitHub in responding in issues and queries, further cementing OpenALPR as the top choice for the project.

Using OpenALPR, the license plate detection module now consists of the following stages: readying input video, pre-processing, and finally the OpenALPR license plate detection.

4.3.1 Readying Input Video

OpenALPR has a free version and a commercial version, as discussed in related work. The commercial feature of processing video streams is replicated with a few changes here.

Instead of processing a video stream, this project applied motion detection to the input from the webcam/camera and stored periods of motion as video clips on the local disk. A busy road would therefore generate a lot of videos, while a quiet road would generate less videos. Applying motion detection allowed for periods where the CPU was idle to save power as OpenALPR wasn't being called every second. It was also much more friendly to the testing phase as recorded videos can be used as input to testing scripts – something that a live stream cannot do.

4.3.2 Pre-Processing

Further pre-processing of the input video to filter out noise, or filter out unnecessary colours such as those except black, white, and orange, was not done as that would interfere with OpenALPR's operation. From verification with the developers of OpenALPR [30], stripping out unnecessary colours would make "plates no longer look like plates since they're disconnected from the surrounding context".

Warping was also explored in the literature review, with OpenALPR preferring head on footage as opposed to sheared images [30]. This could be implemented

either dynamically, or having a configuration stage where a warping matrix is determined and stored.

As the final system is aimed at residential use, a safe assumption is that the system will reside in a stationary position for its operation, and hence a dynamic de-warping system is not necessary, and would only waste computation power. The final design decision was for a configuration stage to determine an ideal warping matrix that produces a satisfactory input image from its relative raw image from the camera.

The final design decision was to leave input video untouched except for warping, barring that done by the camera's own hardware (noise filtering, white balance, exposure, etc.).

4.3.3 OpenALPR License Plate Detection

OpenALPR takes as input still images or video. As the readying input video stage produces videos, the initial design was for OpenALPR to take input video. However, testing, as shown in secion XXX revealed that the Raspberry Pi was too slow at running OpenALPR for real time operation. Hence, the decision was made to instead supply OpenALPR with still frames, extracted from the input video. The system instead processes every nth frame instead of every consecutive frame. This gives a variable performance boost based on how many frames the system skips.

However, the more frames the system skips, the more likely that a car will be undetected if it passes by the camera during the skipped frames. Moreover, less

frames of a repeating car would mean less plate candidates, in turn producing less confident results.

The final design uses input images extracted from the input video instead of the whole video.

4.4 The Peer-to-Peer Network

As the main component of this project, much care had to be put into justify each design decision of the network. The project requirements merely state that the network should share license plates without the need for a central server. It does not specify any way that the P2P network should be built or what protocol to use – all of that was left up to debate.

The first design choice was the lowest level choice, the protocol used in the transport layer of the network. The transport layer sits beneath the application layer of networks and the two most well-known are TCP and UDP. As TCP guarantees delivery of the payload at the expense of overhead and UDP does not, the logical decision was to use a TCP based protocol [31]. UDP is more suited for media streaming, where a lost packet here and there does not affect the overall function of the system. Here, a lost packet would mean lost plates or peer data in transmission, something which is unacceptable.

The next design choice was not to make a custom P2P protocol that sits on top of the choice of TCP. The predicted time spent designing and debugging would be far too great for the aims of this P2P network. Moreover, as this P2P network is not sharing files or media of any kind, a custom P2P protocol like that found in

Bittorrent where the media to be transmitted is broken down in chunks, each with its own hash and availability [32] would bring absolutely no benefit at all to sharing plaintext. Performance of the network is also not a primary concern as plaintext is very small compared to other media types such as files, images, or video. Moreover, in-depth statistics that are broadcasted as part of a normal custom P2P protocol like uptime, number of neighbors, and a file list are not useful in this project.

Therefore, the next design choice was to choose a suitable application layer protocol. Several well-known protocols include HTTP for websites, FTP for files, and SMTP for email. As the peers' will be communicating with each other over the web, and as their payloads contain text, HTTP was chosen as the protocol of choice. Hence, the project's P2P network will emulate the four basic P2P network characteristics in HTTP actions. They are, in no particular order, resource publishing and lookup, P2P network maintenance, heterogeneous connectivity, and request response [33].

Resource publishing is done by each peer sending its plate data to other peers via a HTTP POST request. Resource lookup is done using HTTP GET requests. The P2P network is maintained by each peer sending regular keep-alive POST requests to other peers to inform that they are still part of the network and that plates should still be sent to them. Heterogeneous connectivity, where multiple devices of different hardware should be able to connect to the network is not considered as this project uses a pre-determined set of hardware, but in theory be achievable as the server is cross-platform. Lastly, request response is achieved by using the HTTP response codes received from each HTTP request to do different actions based on whether the response code indicates success or failure.

The final network design choice was how to build the server to listen to requests. As a custom P2P protocol is not used, there is no need to make a custom server using sockets – the time estimated for developing a custom server that is both efficient and interfaces well with multiple network hardware is equivalent to a whole project itself. Instead, a normal web server can be used as they are able to listen to requests on a port of choice [34]. A normal web server can also serve as overlap for the web interface, described in section XXX.

Hence, this project uses a web framework to serve as both the source for the web interface and the P2P network. The P2P network operates inside the web framework, using HTTP requests provided by the Python requests library to carry out network actions based on the response it gets from other peers.

4.4.1 Server Architecture

Prior experience in setting up a server running a LAMP stack (Linux, Apache, MySQL, PHP) from scratch rendered that inadequate for the project as trying to hand code PHP without a web framework when creating any sort of complex network would take an extremely long time. Laravel, and Yii, both modern PHP frameworks, were initially shortlisted as usable frameworks. However, the verbose and sometimes confusing syntax of PHP meant getting things done was emphasised more than code readability [35]. As project development may continue in the future, reusability and code readability meant the final design decision was not to go with a PHP framework. Further research from Srinivasan et al. also showed PHP to suffer from more security issues compared to other web frameworks [36]. Therefore, a framework with protection against attacks such as SQL injection, and Cross Site

Request Forgery (CSRF), all part of the top 10 application security risks as defined by the Open Web Application Security Project (OWASP) [37], was needed.

Emphasising code readability, rapid development, and security features meant the choice was narrowed down to two frameworks in two programming languages: 'Django' in Python, and 'Ruby on Rails' in Ruby. Both offer extremely fast prototyping and development, extensive documentation, security measures against common attacks, and multiple libraries to assist development. The final decision was to use Django, the Python web framework, as the ease of use of Python (smaller learning curve compared to Ruby) and the ample documentation on Django, along the active community meant decreasing the time needed to create the MVP.

Multiple Django libraries were leveraged to add extra functionality, the most notable being: django-rest-framework, a library which provides the skeleton of the API on which the peers use to communicate. This is in addition to the multiple Python libraries used for issuing the HTTP requests.

4.4.2 Database Architecture

The main design decision when choosing the database was whether to go with a Structured Query Language (SQL) or a NoSQL database.

SQL databases are known as relational databases, where databases are linked together by keys and values [38], held in entries in database tables. In SQL databases, all the incoming data must match the format of the database table, whilst NoSQL operate on the premise that the incoming data is of a large volume and of a rapidly changing format [39].

The most well-known NoSQL database is MongoDB, and it offers several advantages over SQL databases. MongoDB claims scalability and performance improvements in [40], claiming that NoSQL databases are horizontally scalable (add more servers) instead of vertically scalable (have to make the one server more powerful). However, the flexibility of NoSQL data means there exists consistency issues when dealing with many similar data objects – unacceptable for license plate or peer data. Nayak et al. compares NoSQL's data formats, showing the data being held in a binary Javascript Object Notation (JSON) object [41], which allows it to be accessed using object oriented methods. However, this advantage is nullified with Django as it has its own object oriented wrapper for any type of database. Django supports its own 'Models', which abstract away the complicated SQL statements needed to modify the database [42] in favour of treating database tables as objects, nullifying yet another advantage of NoSQL databases.

Hence, the final decision was to use SQL databases. Having a SQL database means structured, and relational relationships mean data can be linked with others very easily. In the case of the P2P network, new plate data can be linked with their respective source peer, and new violations can be linked with their respective plate data.

There are a few popular SQL databases, the most popular 3 being SQLite, MySQL, and PostgreSQL. From [43], the pros and cons were evaluated; the final decision was made to use SQLite, a SQL database that comes shipped with Django by default, with the main justification coming from portability (copy and paste the database across testing machines, committable on Git), and it supports enough

features to not be considered bloated. Scalability issues have been moved down in priority as, according to SQLite, they only occur at high volumes of data [44], an unrealistic target. Lastly, NoSQL support is not part of the official Django development effort, and is only supported via third party forks [45].

4.4.3 Communications Architecture

Creating an API was a top priority for the P2P network as it enabled a consistent communication format between peers. The network's API exposes URLs in which data can be sent or retrieved, including but not limited to peer and plate data.

To create the API, a communication format and architectural style had to be decided. Nurseitov et al. compares the two main communication formats, eXtensible Markup Language (XML), and JSON [46]. XML follows a rigid pre-defined structure while JSON does not have any pre-defined structures, so initially it seemed XML was the way forward as health data follows a pre-defined format. However, since everything in XML is stored in strings, parsing the XML data takes relatively more processing power than that of JSON, which can have single entries or arrays of strings or integers – making JSON much more efficient, especially on platforms with lower computation power as demonstrated by Sumaray et al. [47], making it the choice for the network's data format.

To decide on the architectural style, the pros and cons of Simple Object Access Protocol (SOAP), Representational State Transfer (REST), and Remote Procedure Call (RPC) were compared based off information from [48], [49], [50].

As SOAP relied on XML, it was not chosen. Based on these results, the network was designed to use the RESTful architecture as the main API functions consist of mostly data management commands. The main purpose of the network, data retrieval and insertion, aligned well with the uniformity and URL design of REST [51], [52].

On another note, as the P2P network was only responsible for distributing license plates, all communication would be in plaintext. Hence, there was no need for any complex file transfer protocols, nor any requirements to break down the payload into chunks before sending to peers.

4.4.4 Security Features

From the advanced project goals, communication should be protected against rogue peers trying to steal plate data. The design of the P2P network achieves this in three ways.

First, all the outgoing HTTP requests are encrypted locally before being sent. This design decision came after concluding that it would be cumbersome to setup HTTPS on all peers, given that HTTPS requires a user setting up their own certificate [53]. Not only is this un-enforceable for all peers, the user might not know how to set up a certificate easily. As the project aims to abstract away much of the underlying technology to the user, HTTPS was not considered as a method of encrypting outgoing data.

Two encryption schemes were considered – symmetric key encryption, and asymmetric key encryption. Symmetric key encryption relies on having only one

secret key – everything is encrypted and decrypted using this key, while asymmetric key encryption has a public and private key [54]. Symmetric key encryption requires both parties to have knowledge of the key prior to transmission, while asymmetric encryption has no need for pre-exchanging keys [55]. Symmetric key encryption is faster than asymmetric key encryption. Whilst asymmetric encryption seemed the better choice, the fact that the public key needs to be transmitted, normally by the means of a certificate, runs into the same problem as HTTPS. Therefore, the encryption is done using symmetric key encryption. The key is pre-generated and stored in the settings of the P2P network. While this is not a secure method of storing the key, it allows successful operation as a proof of concept.

The symmetric key encryption used for this project uses the newer Python cryptography library as opposed to the more feature-rich PyCrypto, as official support stopped in 2012 - current development consists of multiple third party forks on GitHub. Moreover, the cryptography library has more user-friendly templates and high level encryption recipes, compared to PyCrypto's buffet style, where there is a lack of clear and concise instructions on why a combination of hashing/encryption works better than others, or the steps needed to ensure proper encryption using the many low-level cryptographic primitives available [56].

Secondly, a local trust system was implemented in each peer with the main aim of preventing rogue peers from connecting to the network just to leach license plates from other peers. Trust values are local to a peer, and therefore each peer may have differing trust values to other peers at a given time. Trust scores were also not transmitted to others, to prevent another form of attack where a rogue peer could attempt to deface another peer by transmitting a low trust score to others. In the

design, trust values of each external peer were increased when a matching plate was received, with the trust decreasing over time if there were no plates or no matching plates. Therefore, the system protected against rogue peers from leaching license plates as the peers were designed to not broadcast license plates to peers with low trust levels.

Finally, a token authorization system for P2P network requests was designed. Since a rogue peer may attempt to pretend to be another peer in a spoofing attack, every HTTP request was designed to include an authorization token in the header of the request to ensure the request came from a genuine source. This authorization token was designed to be randomly generated and of enough complexity that it could not be brute forced. Any request with an incorrect authorization token will be rejected by peers. Lastly, although the token resides in the database, it was designed to never be shown to the user – the only way a user would be able to get the token would be to query the database (the database has in-built protection mechanisms), or find a way to decrypt the HTTP request and extract the token from the header of the HTTP request.

4.4.5 Bootstrapping

The first action of a newly-connected peer was the find a list of the currently connected peers in the network. From section XXX, a common bootstrapping method was to ping every possible IP address in its vicinity on a pre-determined port with a pre-defined message and await an acknowledgement response. However, the chance of success with this method in this project was near zero as there were only a few peers in the network at any one time during the project. Hence, the design decision was to explore another method of bootstrapping.

The method of bootstrapping used in this system was a central bootstrapping server, with the URL hardcoded into each peer's settings. When bootstrapping, a peer consulted the bootstrapping server for the peer lists. A list of all peers including their details was held in the bootstrap server. By design, the bootstrap server held all the details of all peers as there were only a few peers in the network. However, scalability issues were decided to not be a problem, given the scalability of Django (it is used for Instagram and Discus [57]). Moreover, the design of the bootstrap server was a replicable one – that is, users could set up their own bootstrapping server by adding a bootstrapping server's URL to the settings of the project, taking the load off the main bootstrapping server. Finally, the bootstrapping server was designed to either hold as little information as possible, i.e. IP address and port, or be modelled as a peer. The final choice was to use a mixture of both.

As the central bootstrap server was designed to operate independently, it had no access to any of the peer's databases. This provided isolation against unwanted changes to the peer's databases. Moreover, the bootstrap server is not another system, but integrated into the P2P network so a peer can either start as a central bootstrapping server or as a normal peer. This approach has the advantage in that if the central bootstrapping server goes offline, anyone can setup their system as a bootstrapping server – peers can then continue normal operation provided the hardcoded URL in the settings file is changed. Therefore, anyone can be a temporary bootstrap server in the case the original one fails. A community that wishes to set up their own private network can also use this method to set up its own private bootstrapping server.

The bootstrap server is designed to be as barebones as possible. It is designed to only contain one database, that is, the list of peers. This list, however, contains more than the absolute minimum required. In addition to the IP address and port, the database holds information the estimated location of the peer and the time since connecting. This information is useful if a peer only wants to request a peer list of all the peers in some city or location.

Lastly, the bootstrapping server is responsible for creating and distributing the tokens in section 5.3.3. To recap, upon successfully bootstrapping, a token, i.e. a randomly generated string of characters, is created and returned to the bootstrapping peer. Any other peer requesting a list of peers from the bootstrapping server will have access to this token and must use it in all requests to the original peer, otherwise the request will be rejected. Therefore, the bootstrapping server plays a critical role in the security of the P2P network.

4.4.6 The Core of the P2P Network: the Peer

Operating as a peer in the network, the system holds databases for multiple objects. It includes a database for the tokens as received from the bootstrap server, a database for the peers in the network, a database for the plates detected from peers and from local videos, a database of videos to be processed, and a database for the detected violations.

The peer also executes all the commands needed to run the P2P network. It does so by running custom network commands on a pre-defined schedule using Linux's cron scheduler. This led to two big design decisions.

The first was between broadcasting and requesting. Broadcasting consisted of the peer sending out plate data to all other peers, while requesting consisted of the peer requesting plate data from all other peers. While requesting data from peers is simpler to implement on the networking side, there is no way of knowing when a peer has detected a new license plate – the requesting peer would have to poll every peer in the network indefinitely. This would very easily lead to congestion in the network. Moreover, polling is inefficient as constantly issuing requests consumes more power than using interrupt based methods, i.e. broadcasting. In broadcasting, a peer only sends out requests when there is a new plate, and therefore can sit idling if there are no cars that pass, giving lower power consumption.

The next big design decision was between running commands reactively based on new data or running commands on a schedule. Running commands reactively made logical sense initially, however, is not scalable. If there are many peers in the network, broadcasting a request to all peers every time a new car passes by will clog up the network and degrade performance as a lot of peers will be broadcasting on average. Hence, the design was for peers to broadcast on a schedule. Data to be transmitted is stored in the database with a Boolean flag to indicate that it needs to be transferred, before being transmitted when the command for broadcasting to peers is run.

Moreover, running commands on a schedule was beneficial for testing and debugging, as the only way of testing a command reactively is to supply the system with an impulse, namely, a car passing by – this approach is very cumbersome to test and cannot be used in conjunction with testing scripts as there is no way of guaranteeing real time input when using a testing script.

The high-level design of the P2P network therefore contained the following transactions and actions. They are split into inter-peer (peer to peer) transactions and intra-peer (commands a peer ran on itself) commands. Although

Inter-Peer

- A transaction to register with the bootstrap server
- A transaction to get a list of the current peers in the network from the bootstrap server
- A transaction to send keep-alive packets to both the bootstrap server and the peers in the network
- A transaction to broadcast detected plates to others in the network

Intra-Peer

- A command to detect violations from plates in the plate database
- A command to modify the trust of each peer based on the number of matching or non-matching plates

These commands were run on a pre-determined schedule and summarise the operations of the P2P network.

4.5 The Web Interface

The web interface is what the user sees and interacts with, i.e. a GUI. It is designed as a website running on the aforementioned Django web server. Its main use is to be the link between the user and the network.

A dashboard was designed for the user to control the network. All the available network commands are available to the user via buttons. Moreover, the details of the network are shown on the dashboard, including, but not limited to, a map of the peers connected to the network, a list of violations and one-click buttons to post the violation on social networks, and a settings page showing all the user defined settings of the network, e.g. name, address, port, and API keys.

With the web interface being the only link between the P2P network and the user, the web interface should be modern, intuitive, and easy to use. The following design principles were followed during development:

1. Compatibility with mobile devices for viewing on multiple platforms
2. Use existing HTML styling frameworks to assist development
3. Simple navigation buttons should provide directions to all parts of the website
4. Prioritise reader comfort and readability of text, use neutral colours in graphs and text

5 Build and Implementation

This section contains more low level information on the design from the previous section. Instead of justifying why some approach was used, instead, this section contains the steps followed to execute the approach. This section is split per module function, and is pre-faced with a discussion on the framework used in all three modules, Django, and its implementation.

5.1 Django

5.1.1 Use

From section XXX, this project uses the web framework Django to serve as the building block for the P2P network and web interface. However, the license plate detection module also utilizes several aspects of Django.

As the license plate detection module and the P2P network both require the use of databases, it did not make sense to have an external database other than that of modularity. Instead, the database inside Django was used, bring three main advantages. First, all three modules could simultaneously access the data inside the database, and change it synchronously. Another advantage was that all three modules could take advantage of Django's object oriented style of database access, removing the need to execute SQL queries when trying to add or remove database entries. Entries in database tables were also easily modified using the in-built Django shell. Lastly, the in-built data types with Django's databases could be leveraged when creating the databases themselves. This ensured any data being added to the database was of the correct type as any data added to Django's databases must pass verification with the database.

5.1.2 Development Style

Django is a MTV framework, meaning development is split into 3 tranches – models, templates, and views [58].

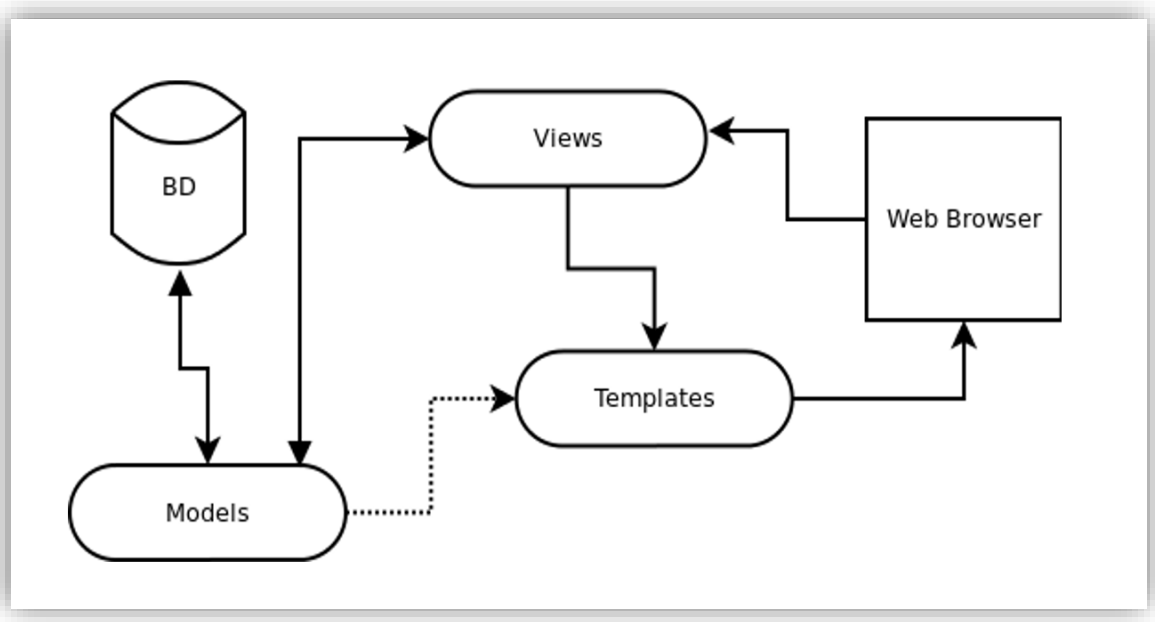


Figure 4: Django's MTV style (BD = database) [59]

The model describes the data access layer, and in this system's case describes all the databases that hold valuable information, i.e. peer information, plate information, and violation information. Models are also responsible for knowing how to access these data, what the format the data should be in, and what relationships it has with other data. Development in this section included creating database tables with the appropriate data types and accessors.

Next are views, which form the link between models and templates. Views contain logic that extract useful information from models (e.g. a developer might request database entries from a certain day, from a certain source, or between some threshold values only), before passing them onto templates. Moreover, views may

contain logic which execute commands based on user input (e.g. URL parameters). Development in this section included getting and organising relevant data from the databases before passing them to templates, or as a JSON object in the return HTTP request.

Finally, templates describe how data should be presented, and contains instructions and placeholders for how the user will see the final data. Examples contain tags inside an HTML document that will be replaced with whatever data is passed to it from the views, and tags which indicate whether an HTML document a block that should be displayed as part of another parent document. Development in this section included the classic web development ideas such as CSS, HTML styling, and webpage design. Base templates with block tags were designed to be used across all the webpages inside the web interface, giving a unified look to the interface as opposed to varying designs on each page of the interface. A specific template for a specific page could then override the relevant block in the base template to display useful data.

5.1.3 Ramifications for core modules and programming style

Although using Django in part, or fully, for each of the modules brought many irreplaceable advantages, development was initially hindered. As all database data is verified before entry, initial development was slow as dummy and fake data would not pass the verification checks. However, this hindrance in hindsight brought vast improvements to the programming quality of the project, bringing in the addition of logical defaults to empty database entries where necessary, and the catching of all kinds of exceptions like integrity (unique constraint of an entry failed) and type errors. Code responsible for the P2P network and the web interface were also

refactored to give out useful debug information in a structured form in the case of exceptions and errors, instead of relying on printing text and values manually during debugging.

5.2 License Plate Recognition

The following figure shows the logic flow of the module. Input video is processed and fed into the license plate recognition software, OpenALPR. The resultant license plates are processed and then saved in a database.

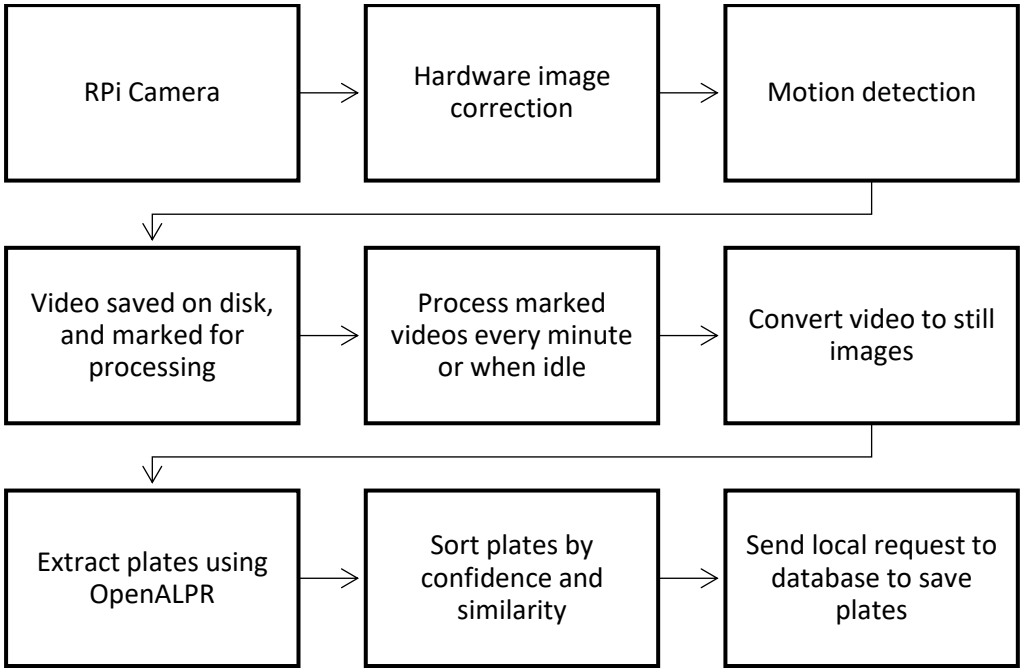


Figure 5: Flow of LPR

5.2.1 Camera Input

As this project assumes the use of Raspberry Pi's NoIR camera, the inbuilt raspistill and raspivid commands were initially used to capture footage from the camera.

An extremely basic background subtraction motion detection script using raspistill was created to trigger the capture of the camera when there was motion detected. This script assumed that the first frame of capture would be the background - this is not true at all in practice. The script's logic is described in the following pseudocode snippet:

- Save first frame, assume it to be the background
- While True:
    - Capture a new incoming frame
    - Subtract the background and calculate if the resultant differences mean motion
    - If motion detected, run the image through OpenALPR

Code Snippet 1

However, the latency of those commands was extremely high during early development, reflected by multiple reports online of the same problem [60]. The impact of this latency was seen when trying to trigger the camera capture command when a car was passing resulted in an image that was taken after the car had passed.

Therefore, another solution was desired. The answer came in the form of an open source software library, RPi-Cam-Web-Interface (RPCWI). RPCWI, a web interface for the Raspberry Pi camera, came with out of the box support for a live preview of the camera running off an Apache server, low latency and high framerate video and image recording [61], in built motion detection with tweakable parameters, and a well-documented architecture.

RPCWI utilizes a low-level API, raspimjpeg, to interface with the Raspberry Pi's camera. It copies every new frame from the sensor into RAM (/dev/shm/), and hence does not have any I/O bottlenecks. Therefore, the next iteration of development was feeding the current frame into OpenALPR in an infinite while loop. However, not only is this power hungry, it is inefficient as it calls OpenALPR even when there are no cars going past – leading to many repeated OpenALPR calls on the same image.

Therefore, the last step in this module's development was to incorporate the inbuilt motion detection of RPCWI to record video snippets to the local disk when there is motion detected. Several motion detection parameters, along with other image quality parameters such as brightness, contrast, and resolution (OpenALPR suggests the maximum resolution as being 720p in its docs, as a high resolution does not necessarily bring better plate detection, and requires considerably more processing power) were tweaked form the defaults. This not only saves on a lot of computing power during idle stages, it ensures OpenALPR is not repeatedly called on images containing no cars. Another advantage of this is that it gives useful input video that can be saved and backed up for further testing, instead of having to gather new footage of cars during further development.

This is done by the following flow. Using RPCWI, videos are saved onto disk upon successful motion detection. Alongside, a background cron job scans the video folder for new videos every minute. If new videos are found, their file path is added into the local video database and marked as 'Not Processed'. When the Raspberry Pi is idle or the number of unprocessed videos exceeds some threshold, a script is run to

call OpenALPR on each of the unprocessed videos. The resultant data from OpenALPR is added into the plate database for use in the P2P network. Not only does this approach ensure all videos eventually get processed, it ensures a smooth user experience as the CPU-heavy OpenALPR is not called when the Raspberry Pi is busy executing other tasks. This approach also ensured that every video that got saved got processed eventually (not a problem as the videos are timestamped). This implementation point is critical as the prior method of calling OpenALPR on live still images or a live video stream would fail if the system shut down suddenly in the case of a power outage, or if the raspistill or RPCWI processes were killed from user input.

The aforementioned both reflect and conclude blocks 1 to 4 out of 9 from the flow shown in Figure 5.

5.2.2 OpenALPR

This section refers to blocks 5 to 7 out of 9 from the flow shown in Figure 5.

By itself, OpenALPR has multiple settings that can be varied in its configuration file. As this project is aimed at UK plates only and no custom plates, the GB setting was used inside OpenALPR to automatically reject any plates that do not follow the UK license plate naming nomenclature – letter, letter, number, number, space, letter, letter, letter. Another setting that was varied was that minimum confidence that OpenALPR will allow for a plate to be considered a candidate. As the input video that OpenALPR received from testing was all high quality, the default value of minimum confidence did not need to be changed. The incorrect plate candidates with low confidence were rejected using the next variable, the number of plates to

return. As OpenALPR automatically sorts the license plate candidates by confidence for every frame, setting the number of plates to be returned to one meant only the highest confidence plate was returned.

Another important setting was the pre-warping matrix. As the camera is likely to not be facing head on to cars, the camera input image had to be undergo a perspective transform to ensure the license plate area is warped to as close to a rectangle as possible. A Python script was created and implemented initially to be between the camera interface and OpenALPR, however, upon further exploration of the secondary features of OpenALPR revealed OpenALPR had itself a built-in perspective transform too.

Therefore, to keep the system pipeline as de-cluttered as possible, OpenALPR's warping tool was used instead of the Python script. Testing to find the resultant parameters can be found under section XXX. Otherwise, any other settings that were varied were all practical (return data type, how verbose the returned data should be, etc.).

The first major change in development was to realise that the video processing implementation of OpenALPR was merely to split the video by frames and call OpenALPR per image, before concatenating all the results in a final array. This method of processing meant that a group of detected license plates were associated to a video, and not a specific time. If the video was 10 minutes long, every detected plate would be detected at the timestamp that corresponds to the start of the video. The last car in that video would be detected a full 10 minutes earlier – a critical problem when the speed of the vehicle depends on its detection time.

Additionally, the high framerate video being recorded was overkill for license plate detection if every frame was fed into OpenALPR. With the FPS set at the minimum of 25 in RPCWI, the difference between consecutive frames was negligible (Figure 6 and 7), again highlighting the problem of wasted CPU usage.

Figure 6: Consecutive Frames

Figure 7: 5 frames apart

Hence, the final implementation of OpenALPR was wrapped in an outer script which solves both problems.

For every video that needed to be processed, a separate folder on the local disk was temporarily created. This folder housed every n^{th} frame of the video, as extracted using a counter in a while loop. Every n^{th} frame (now in image form), was then timestamped with the exact timestamp of that frame, calculated by adding on a

time delta of 1/FPS. This ensured the OpenALPR stage would return the license plates corresponding to the exact time that the vehicle appeared in. The actual code implementation returned a tuple of an array of the detected license plates, the timestamp as extracted from the filename, and the file path of the image, for ease of adding to the database in the next section.

Even though processing every n^{th} frame gave a performance increase of n times, the speed of OpenALPR on the Raspberry Pi was still slow when compared to running on a desktop class CPU (testing section XXX). As multicore processing was locked away behind the commercial version of OpenALPR, the next implementation was aimed at making the OpenALPR wrapper script multicore.

Since processing multiple images at the same time did not need to be done in the same memory space, multicore processing was desired. Python's multiprocessing library was utilized as opposed to the multithreading library (Python's implementation of multithreading utilizes a single core only) [62], [63]. In choosing a pool size for multiprocessing, a pool size that utilised all available CPU cores led to borderline temperatures of operation (85 ^{0}C), which in turn led to thermal throttling [64]. Hence, the final implementation of a pool size of 3 meant getting close to 3x performance (section XXX), while using 3 out of the 4 available CPU cores (Figure 8).

To provide another way of combatting the thermal problem, heatsinks were purchased and added to the SOC's on the Raspberry Pi. While the added heatsinks were a welcome addition - they did decrease CPU temperatures by 3-5 ^{o}C, keeping one CPU core available was more important in the grand scheme of things as it

meant other processes would not be affected when OpenALPR was running. The final implementation led to thermals of around 75 ^{0}C.

Here it is interesting to note an unused optimization for this project's multicore processing. As OpenALPR requires loading configuration files every time the OpenALPR object is instantiated, an attempt was made to instantiate X amount of OpenALPR objects with their required configuration files before calling all of them in a pool queue to ensure time wasn't being wasted in loading configuration files. However, as X is equivalent to the number of images needed to be processed, and assuming X = 150 images for a 30 second video (5 FPS), 150 OpenALPR objects had to be instantiated. This led very quickly to memory problems and segmentation fault crashes at random times throughout processing –hence this optimization was left unused as the time spent debugging the memory issues would not outweigh the negligible time gained in speedup.

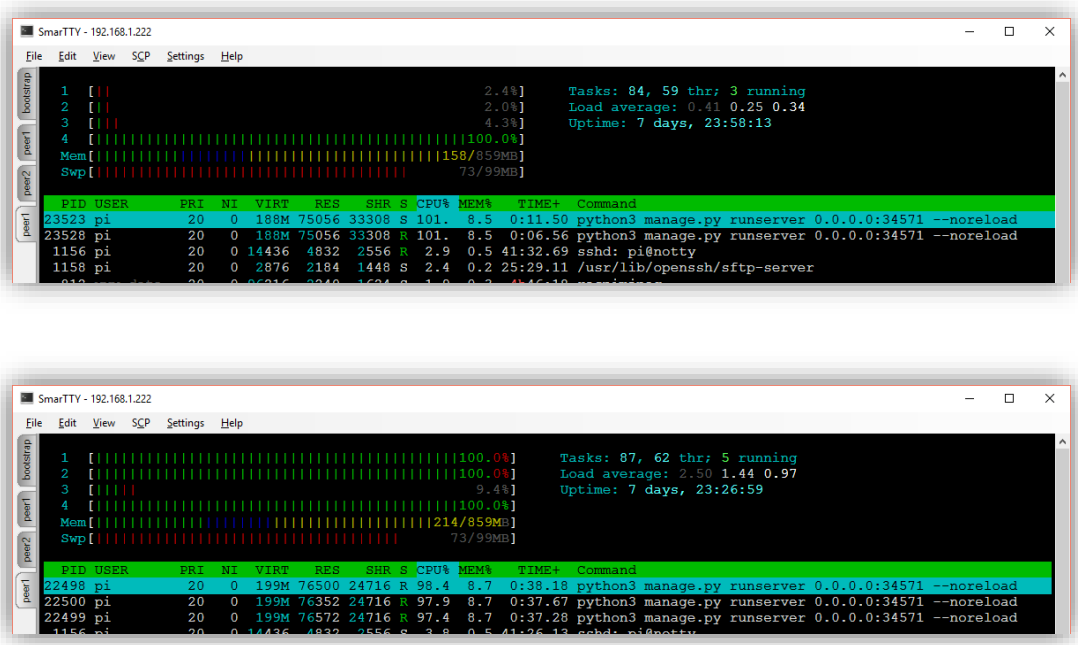


Figure 8: Before and after multicore (3 out of 4 cores)

5.2.3 Processing the resultant plate data

The output data format from OpenALPR is a JSON object containing plate candidates for every frame. However, as consecutive frames most likely contain the same car, some post-processing was needed to extract the unique plates from the list of all plates. A 'group similar plates' feature is available in the commercial version of OpenALPR, but as that was behind a paywall, something similar had to be developed, tested, and implemented.

The initial method was to sort the resultant array of plates by their confidence as determined by OpenALPR, and select the top few. However, there might be repeat plates with high confidence so this method was not robust.

The next method was taking all plates from the beginning to motion detection to the beginning of the next motion detection. However, again, there might be a few cars within one video so this would not get all the unique cars in the video.
The final implementation uses the consecutive frame property to its advantage. As consecutive frames are likely to contain the same car, the similarity of the license plate string must be high. If the similarity of the previous license plate and the current license plate is low, then it was highly likely that the plate belonged to another car. Hence, an algorithm to extract unique plates is shown by code snippet 3. Python's SequenceMatcher function from difflib is used to determine the similarity of two strings.

```
return_list_outer = [ ]
retun_list_inner = [ ]
previous_plate = “ “
for current_plate in plates:
        if similar(current_plate, previous_plate) > some threshold value:
                # it must be the same car
                append current_plate to return_list_inner
        else: # it must be a different car as strings are not similar
                sort return_list_inner by confidence
                append the highest confidence plate to return_list_outer
                clear return_list_inner
        previous_plate = current_plate
return return_list_outer
```

Code Snippet 2: Getting unique plates

Unique plates were successfully extracted from the complete list of plates given by OpenALPR using this algorithm. This method also solved a problem with the plates given by OpenALPR – sometimes, depending on the input image, OpenALPR would return a license plate with one or two missing/incorrect characters. An advantage of this method is that plates with one or two missing characters would still be regarded as the same car. Table 1 and 2 shows this algorithm separating unique cars, with the most confident candidate plate the one being added to the database.

Table 3: Unique plates extracted from a list of all returned plates from real world footage

Actual Plate	KP08UDE	LM65ERT	FE16RRX
Variant 1	KPO8UDE	M65ERT	FEIRRX
Variant 2	KP08UD	LM65E	RE16RRX
Variant 3	KP08U0E	LM5ETT	FERRX

Table 4: String similarity

Plate 1	Plate 2	String Similarity
KP08UDE	KP08U0E	0.857
KP08UDE	LM65ERT	0.143
KP08UDE	FE16RRX	0.143
LM65ERT	M65ERT	0.923
LM65ERT	FE16RRX	0.286
FE16RRX	FEIRRX	0.769

The final list of plates contain the aforementioned OpenALPR return type of a tuple containing the plate string, the timestamp, and the file path of the source image. This concludes block 8 out of 9 from the flow shown in Figure 5. Hence, each car was then added to the local plate database with this information – completing block 9 out of 9 from the flow shown in Figure 5, and the license plate recognition module.

5.2.4 Database Implementation

The last section of the license plate recognition deals with the implementation of the two databases needed: the videos to be processed database, and the license plate database.

The videos to be processed database is the simpler of the two, with only three fields: the file path of the video to be processed, a Boolean value of whether the video was processed or not, and the time when processed.

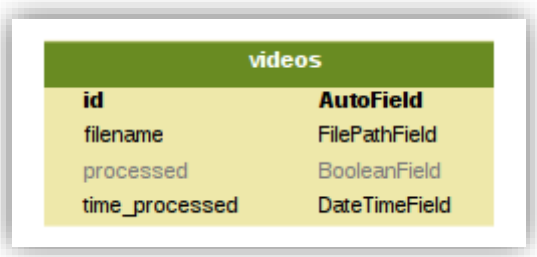


Figure 9: Videos to be processed database table

As the plate database has foreign key relationships with the P2P network databases, it will be explored there instead – see section XXX.

5.3 The Peer to Peer Network

The core of the project, the P2P network constitutes the backbone on which the license plate detection and web interface operate on. As such, it required the most thought in implementation.

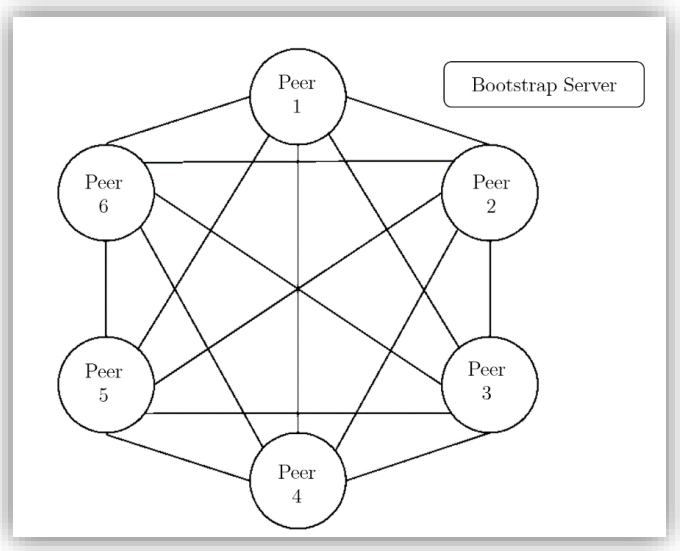


Figure 10: Structure of Peer to Peer Network (lines indicate persistent connection)

Figure 9 demonstrates the typical P2P network connections. Since the project does not use a pure P2P network (no centralized server, even for bootstrapping), there does exist a bootstrapping server whose only job is to register new peers and broadcast to them the details of existing peers. The bootstrap server is isolated from the communications of the peers that are already in the network, apart from keep-alive packets, and as such, is not presented with a line in the figure as the connection to the bootstrap server is both private and short.

As the P2P network was the most complicated out of the three modules, the implementation is split into 3 parts. The first talks about the underlying design of the databases that store all the data of the license plates and the network. Next, the operation of the P2P network is described, building on the data that is stored inside the databases. Finally, the added features, including the trust system and the security features, are elaborated on, further extending the discussion on the operation of the P2P network.

5.3.1 Database Implementation

The system's P2P network requires just two database tables to function (all other functionality disabled), a database table to record the details from the bootstrap server, and another database table to store the details of the peers in the network. As the system's P2P network's main aim is to distribute license plates and detect violations of speeding cars, two other database tables were created: a plates table and a violations table.

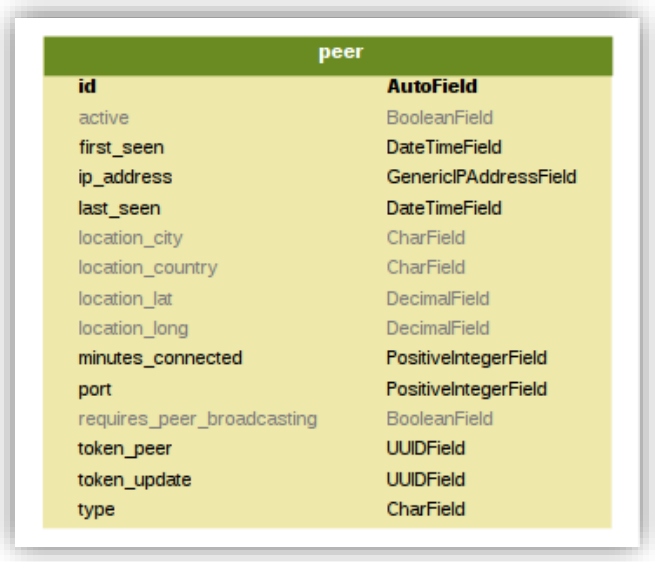


Figure 11: P2P network bootstrap database tables

Figure 12 shows a visual guide to the bootstrapping server's database. It only contains one database table, the list of peers in the network.

Figure 13 shows a visual guide to the peer's database tables along with the foreign key relationships. There is a foreign key relationship from the violations to plates as every violation comes from the detection of two cars, hence two plate entries. Each plate entry has a foreign key relationship to the peer who the plate was detected by. Along with Django's object oriented style, these relationships allowed rapid discovery of the source of each violation. Finding where the evidence image

that was linked to a violation was stored, locally, on a peer was as simple as querying Django for 'violation_object.plate1.source.img_path', instead of the complex SQL queries that other non-modern frameworks required.

The entries in each database table were also assigned different data types, including but not limited to Python datetime fields, floats, character strings, and Booleans. However, special care was given to the location fields and the token fields.

The location fields represent a GPS latitude and longitude, in turn representing a location on Earth. Since the resultant system would be used for calculating speeds of vehicles based on the location of the peer, the location fields needed to have the correct precision. From [65], 5 decimal places of latitude and longitude gave a precision of 1.1m, 6 decimal places gave 0.11m, and 7 decimal places gave 11mm. The final implementation used the future proof 6 decimal places. This implementation decision was justified further not by predicting the accuracy of a user's GPS equipment when setting up the system, but by the default number of decimal places upon clicking 'What's Here?' on Google Maps (a recommendation that was added to the user's instructions). GPS locations were also not instantiated as a float, nor as an integer, nor as a character field, but as a decimal field, as that guaranteed the accuracy of every decimal place, unlike floating point [66].

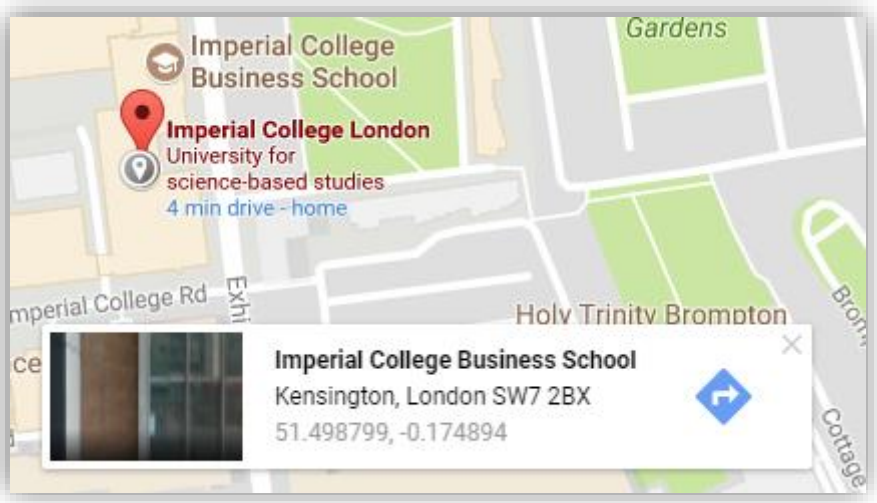


Figure 12: Google Maps' 6 decimal places default

The token field which used to store tokens generated by the bootstrap server for authentication in requests to other peers, as discussed in section XXX, was also put through several development iterations before settling on the final implementation. The first few implementations used a random number generator, seeded by the current time. As using a random number generator did not guarantee the absence of collisions of the generated token, it was unacceptable as the generated tokens distributed to peers should never be the same. If they were the same, a peer could pretend to be another peer by supplying the correct token in its requests. The next iteration used a hash of the current time, however this also was unacceptable as two peers requesting tokens in different time zones may lead to a collision. The last iteration used a UUID, as defined in RFC4122 [67]. A UUID is a randomly generated string using a mixture of the current time, hardware, and random numbers – giving a close to zero chance of a collision.

Lastly, unique primary constraints on the database tables to ensure not more than a copy of some data is entered (e.g. there should only be one copy of a license plate from a peer at some time) exist, but are not discussed further.

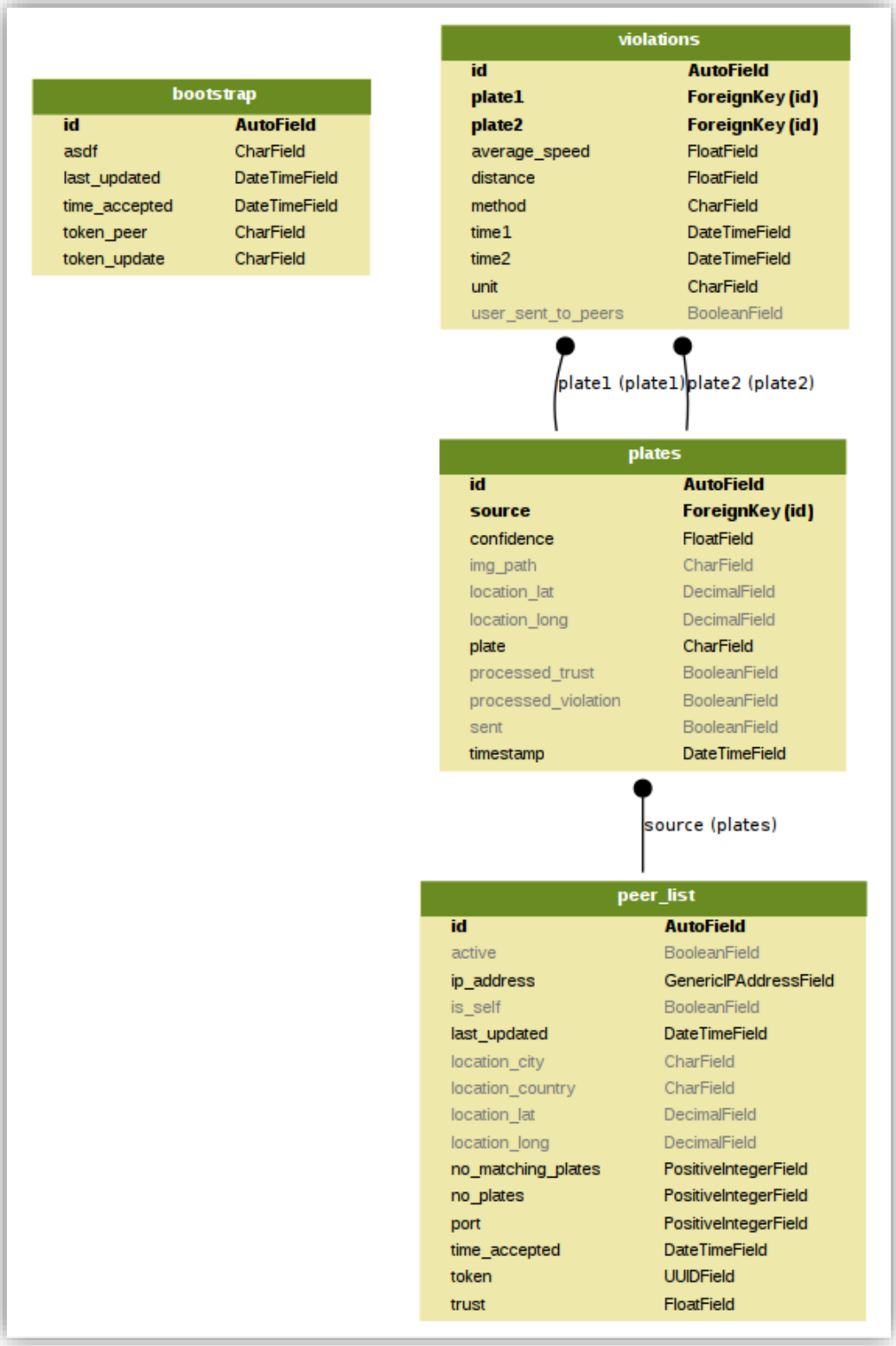


Figure 13: P2P network peer database tables (lines indicate foreign key relationships)

5.3.2 Network Operations

This section details the commands the P2P network runs to ensure the operation of the network. The commands are run on a pre-defined schedule, and do so with the help of Linux's cron jobs. Cron jobs are commands which run automatically on a predefined schedule. However, the convenience of cron jobs is contrasted with the fact that cron jobs come with no environment defined – this system's cron jobs configure the environment every time commands are run.

Justification on why the P2P network functions on a schedule instead of executing commands on impulse given by Django signals (a signal can be fired when a new database entry is added, for example) can be found in section XXX.

A copy of the inter and intra peer transactions is copied below from section XXX.
Inter-Peer

- A transaction to register with the bootstrap server
- A transaction to get a list of the current peers in the network from the bootstrap server
- A transaction to send keep-alive packets to both the bootstrap server and the peers in the network
- A transaction to transmit detected plates to others in the network

Intra-Peer

- A command to detect violations from plates in the plate database
- A command to modify the trust of each peer based on the number of matching or non-matching plates

5.3.2.1 Example use of transactions

To ensure each of the following sections have the relevant context, a sample operation of the P2P network is shown in Figure X. The left tab, registration, is only executed once upon startup, whereas the middle and right tabs operate continuously. The circles denote the completion of the initialization of the network.

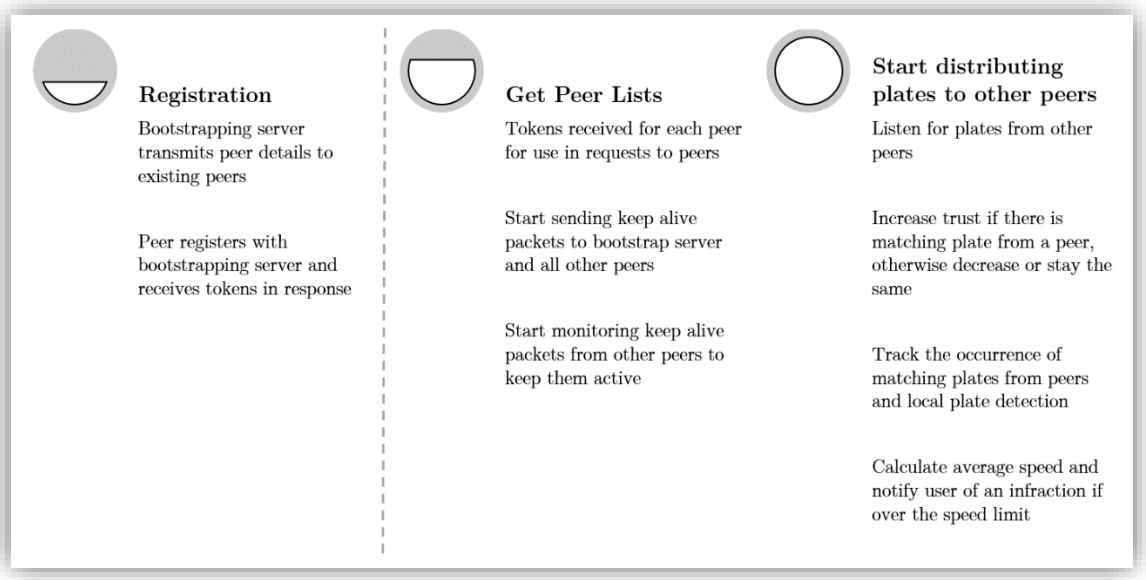


Figure 14: Sample operation of P2P network

5.3.2.2 Registration with the bootstrap server

Registering with the bootstrap server was the first step in the operation of the P2P network. A new peer to network must register with the bootstrapping server before it could communicate with the rest of the network as otherwise, it would not have any of the required tokens for authenticating with the other peers.

Referring to Figure 12, the bootstrapping server took, as a JSON object, a minimum of two data fields: the IP address and the port. The bootstrapping server then worked out the location of the peer if not given, and assigned to it two tokens. The

first, token_update, was used to authenticate the peer if wanted to change any of its details. This token_update was only distributed to the peer upon successful registration, and was not sent to any other peers. The second token, token_peer, was the token responsible for other peers authenticating their requests to the peer. Therefore, token_peer was distributed to all peers currently in the network, so they could successfully authenticate themselves to the newly registered peer.

Other implementation details included returning different HTTP status codes upon successful or failed registration, something used for transmitting changes in peers.

5.3.2.3 Transmitting changes in peers

To ensure the P2P network had the quality of 'P2P Network Maintenance' as specified by design in section 5.4, the bootstrap server, along with all peers, was responsible for distributing up to date information to all the peers connected to the network.

Out of a few common situations that were theorised and dealt with, the most common was in a peer registering, then losing internet connectivity. When regaining internet connectivity, the peer would automatically attempt to register to the bootstrap server again. However, the bootstrap server would then realise that the peer had connected before as an existing database entry would exist for the peer. Therefore, to ensure the network is up to date with the re-registering peer, the bootstrap server was implemented so that a new set of tokens was generated. This new set of tokens was then set to be transmit to all peers in the next iteration of a transmit-all command, therefore ensuring the information of all the peers in the network is both updated, and consistent.

5.3.2.4 Getting the peer list

After registration with the bootstrap server, peers then attempted to get a list of all the current peers in the network, along with their details and respective tokens. The command was scheduled to run as often as every minute, to ensure the peer list was up to date. Along with the bootstrapping server transmitting changes to all peers, the P2P network was guaranteed to have correct information about all peers at any given time (the maximum period of stale information was at most a minute).

In the case the bootstrap server went offline, peers also routinely compared their peer lists with other peers to ensure details matched – refusing to transmit to a peer whose details didn't to prevent a rogue peer.

5.3.2.5 Keep alive

An important feature of the P2P network was to keep track of peers that had disconnected from the network. As disconnected peers were by definition unable to reach any other peers through the internet, there was no way for the disconnected peer to inform current peers about the change in status of the peer.

Hence, a software dead man's switch was implemented for all peers. The implementation revolved around peers having to send keep alive packets to both the bootstrapping server and other peers. Otherwise, a peer would be marked as inactive by other peers after a period of not receiving any keep alive packets. Inactive peers would then not be recipient of any license plates to prevent wasted time in trying to contact the peer, only to result in a timeout. After an extended

period of not receiving any keep alive packets, the disconnected peer would be deleted from the database. This implementation ensured the peer lists of all the peers in the network could not be cluttered with disconnected peers, as well as stale and old peers.

5.3.2.6 Sharing plates

The main objective of the P2P network was to share detected license plates across all peers. Implementing the plate sharing command was straightforward.

Every minute, plates that were marked as unsent was tallied up, and sorted per their respective source peer. This list was then sent to each peer with the peer's token in the header of the request, with the small modification that plates from the destination peer would not be re-sent to themselves. Upon a successful HTTP return code from all peers, the plate object was marked as sent. Otherwise, if at least one peer failed to receive the plates, the plate object was kept as unsent, ready for re-sending in the next iteration of the command being run.

5.3.2.7 Modifying trust scores

This section specifies the implementation of the action of a peer in modifying other peer's trust scores. For the implementation of the trust system itself, a more in-depth explanation can be found in section XXX.

To modify a peer's trust, upon receipt of license plates from an external peer's transmission, the external peer's local trust score would increase if at least one of the received plates matched a plate detected by the local peer. If not, the trust was

decreased. Not receiving keep alive packets after a while also triggered a decrease in trust.

5.3.2.8 Detecting violations

After the sharing of license plates with all neighbouring peers and the modification of trust scores, the final step of the intra peer commands was to detect any speeding cars from the license plates gotten from all peers.

As the project specification specifies detecting violations using an average speed method, the equation of speed = distance / time was used. Plates from the local peer and plates from external peers were sorted in two lists, before a comparison between the two sets revealed matching plates. For each matching plate, the time delta was calculated as the time difference in the times of detection of the two plates. The distance between the two peers was then calculated using the Google Maps Directions API, which returned the road distance between the two peers as opposed to the birds-eye distance. Therefore, an average speed could then be calculated.

As access to the Google Maps Speed Limit API could not be obtained with the project budget (premium API licenses started at USD $10000), an assumption had to be made in that all roads were deemed to have a speed limit of 30 miles per hour. Since some residential roads have a lower speed limit of 20 miles per hour, this speed limit was added to the settings to be easily changed if needed.

Before a comparison of the average speed of the two matching cars and the speed limit as specified in the settings was made, a small percentage of the speed limit

was added on top of the existing speed limit to emulate the 10% that police officers add to the speed limit in real life speed cameras [68], mostly stemming from device tolerance margin of errors and uncertainties. For example, the system would not classify 32 mph as an infraction, whereas 33 mph would lead to a violation.

Cars that had an average speed over the speed limit + 10% were deemed to be violating the Highway Code, and led to an entry being made in the violations database table with the necessary details for prosecution (locations, speed, times, evidence).

5.3.3 Network Features and Security

Apart from the aforementioned tokens required for all requests being sent, the P2P network has a few added features to prevent rogue peers from obtaining license plates from peers without itself transmitting any license plates, along with features to prevent man-in-the-middle attacks where an attacker could theoretically modify the HTTP packets in transit.

5.3.3.1 Trust System

This section differs from the section detailing the modifying trust transaction as it entails the implementation of the design of the trust system, and not the implementation of a command that modified the trust of peers, a needed distinction.

One of the primary concerns during the design phase was the possibility of an attacker stealing leeching license plate data from peers without supplying any of their own, or an attacker sabotaging the network by bombarding the peers with fake license plates, thereby diluting the genuine license plates. A trust system was

therefore implemented to combat this. Echoing section XXX, each peer had a local set of trust scores for each other peer, and would only distribute and accept plates from peers that had a score over a certain threshold.

Every peer, upon successful registration, started off with a set amount of trust – a necessary assumption. Although logically, a newly registered peer should have no trust at all, a deadlock situation occurred if a couple of peers registered at the same time at a time when the network was devoid of any peers. No peer had enough trust to transmit to other peers, and no peer received any plates from the new peers as everyone had a trust of zero – hence the implementation that every peer started off with a non-zero trust value.

If a peer's trust was over a pre-defined threshold, license plates could travel to and from other peers. If a plate from a peer matches a locally detected plate, the trust of the source peer was increased by a pre-defined amount. Hence, the perfect world, the trust of all peers would keep rising.

However, upon a period of there being no license plates received from a peer, or if none of the license plates from a peer matched any of the locally detected plates, the trust of the peer was slowly decreased. A ratio was used to decrease the trust instead of a flat amount, to ensure that the amount of trust gained for a matching plate already outweighed the amount of trust lost through no matching plates or no plates at all.

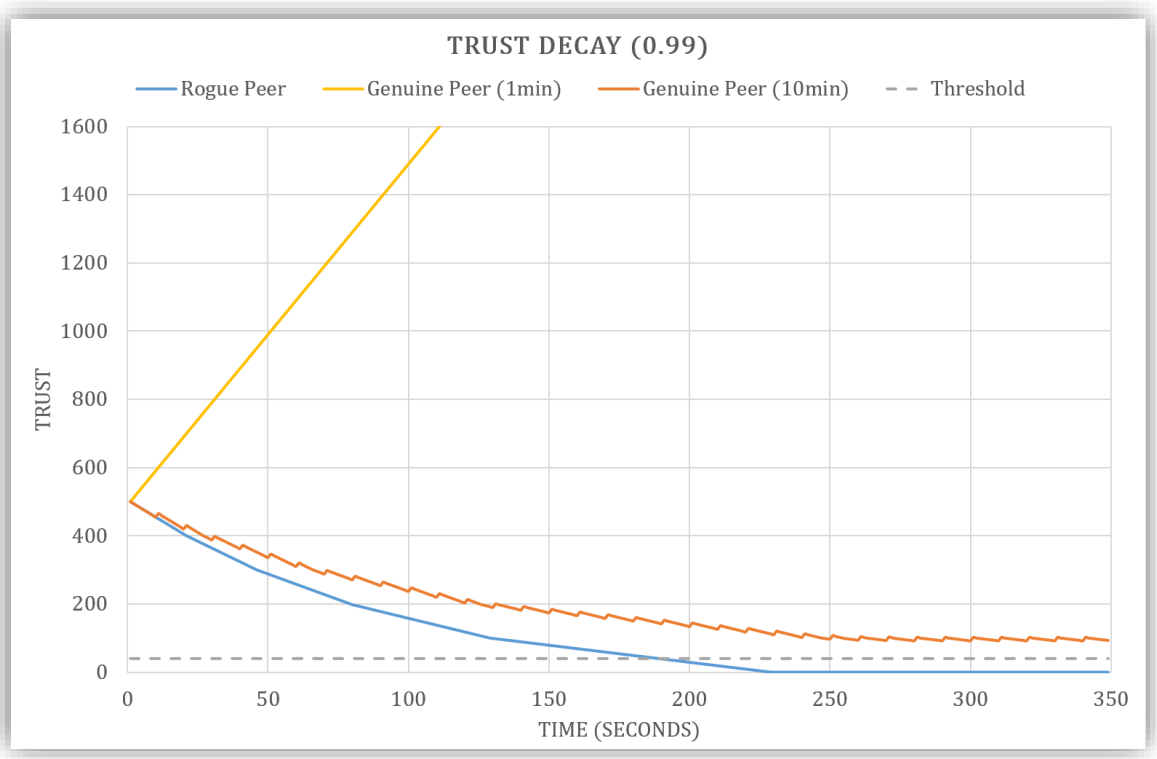


Figure 15: Trust Decay using a ratio of 0.99, applied every minute (genuine peer (1min) is truncated)

Figure XXX represents the trust system in action. The three lines correspond to the trust of a rogue peer (no matching plates whatsoever), a peer with matching plates every minute, and a peer with matching plates every 10 minutes. From the figure, clearly, if peers continued to transmit matching plates, the trust will never fall below the threshold (shown by the dotted line). However, if a peer did not transmit any matching plates, the trust soon fell below the threshold (in approximately 3 hours), with the only redemption being increasing the trust again through transmitting matching plates.

Table 5: Example modifying trust scores with genuine peer

Local Plates	Received Plates

AE56FTY	LTZ 1077
PB12OFF	**PB12OFF**
FY65UGG	**DB16QAZ**
DB16QZA	BB55LPE

Another example, Table 5 would lead to the trust score being increased as at least one plate matched. Matching plates are bolded and underlined for clarity.

Table 6: Example modifying trust score with rogue peer

Local Plates	Received Plates
AE56FTY	XXXXXXX
PB12OFF	TEST123
FY65UGG	!@#$%^&*()
DB16QZA	NOT_A_PLATE

Likewise, the rogue peer attempting to flood the network with random plates in Table 6 would lead to a decreased trust score and eventually an untrusted peer.

5.3.3.2 Encryption

The cryptography library's Fernet AES 128bit encryption was used to "take a user-provided *message* (an arbitrary sequence of bytes), a *key* (256 bits), and the current time, and produces a *token*, which contains the message in a form that can't be read or altered without the key" [69]. This encryption process was done on all objects (JSON) before any external transmission – that is, to other peers, or the bootstrap server, therefore preventing anyone with a packet sniffer or packet interception software to edit the HTTP request in transit. The fact that the HTTP

request in transit cannot be decrypted was extremely useful as this project does not utilise HTTPS for the complexity in each user having to get their own certificate (section XXX); encrypting all data before transmission prevents any rogue hackers from trying to frame a vehicle as speeding by modifying a peer's license plate data transaction with other peers.

The received encrypted JSON object is decrypted by reversing all the encryption procedures with the same key that was used in encryption.

5.4 Web Interface

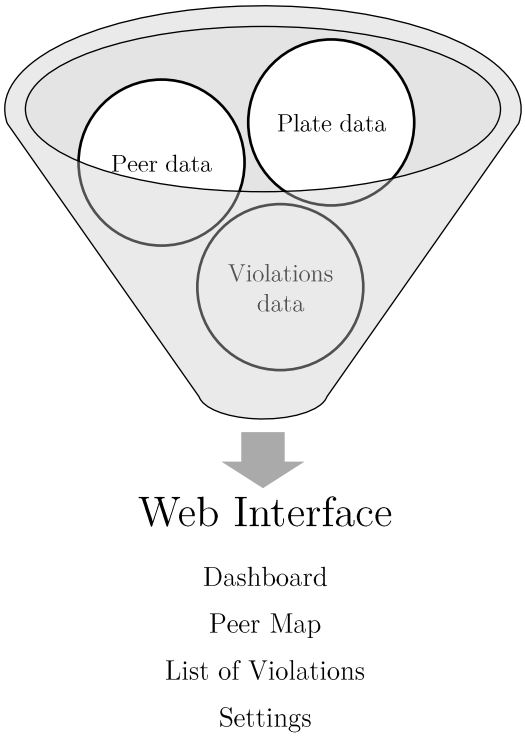


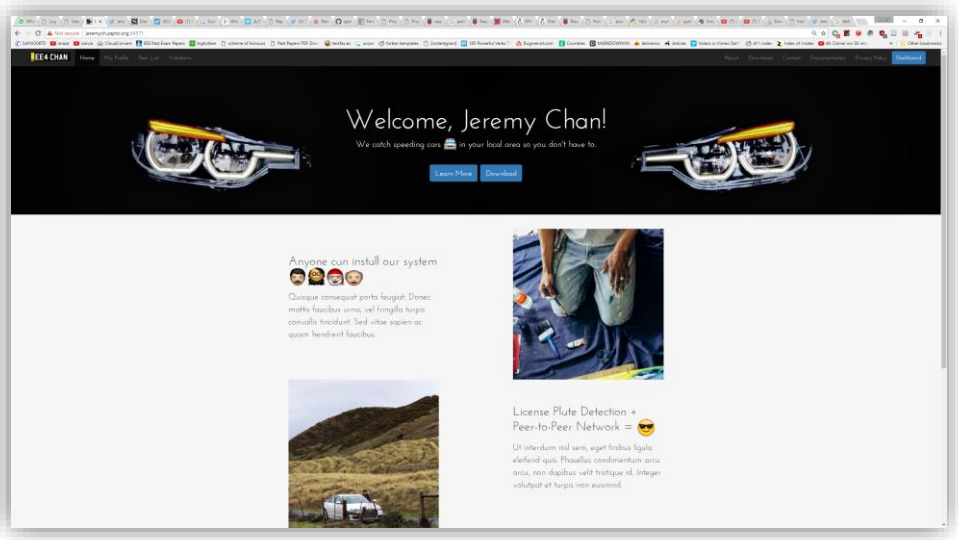
Figure 16: Flow of data in web interface

The function of the web interface is to provide a link between the database data of the P2P network and the user through a graphical user interface. Figure 9 depicts this relationship. The main component of the web interface was a dashboard, where

the user could modify settings, view peer details, and operate the P2P network manually. A peer map was also created to show the locations of all the peers in the network. Lastly, the violations page presented a list of speeding violations, along with a button to post on social networks as per the project specifications. Non-essential pages are also described in brief.

5.4.1 Overall Principles

This section refers to the design principles in section XXXX. To tackle the compatibility with mobile problem 1., *"Compatibility with mobile devices for viewing on multiple platforms"*, responsive web design was a priority in the development of the web interface. By utilizing viewports (show different amounts of data based on the device width and display density) as described in [70], [71], a responsive web design that changes the location of buttons and images depending on device resolution was created as a product of extensive testing on different resolutions.



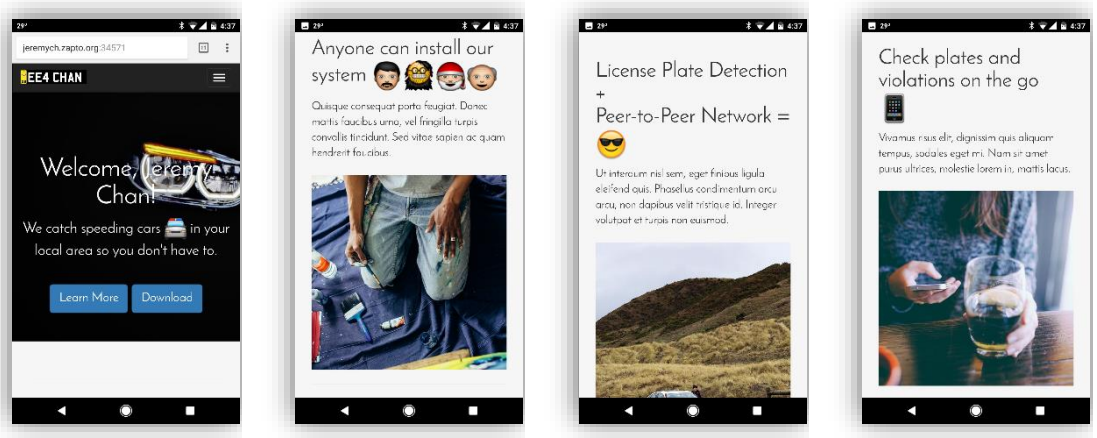


Figure 17: Responsive design based on device (top – desktop, bottom, left to right – mobile)

To tackle the framework problem of 2., *"Use existing HTML styling frameworks to assist development"*, Twitter Bootstrap was used as the library of choice as it is lightweight (only a few static files needed for setup), flexible (allows overriding of default styles with custom ones), and powerful. Bootstrap is also built for mobile first when compared to other styling frameworks such as Foundation and Skeleton [72]. Bootstrap gives predictable websites at the cost of slightly verbose HTML [73]. By following code styles and practices from Chapters 2-5 of [74], a navigation bar, a jumbotron (big heading type text with a banner image at the top of every page to provide context), sidebars, and footers were created as the base template for the web interface, thereby fulfilling 3., *"Simple navigation buttons should provide directions to all parts of the website"*.

No. 4, *"Prioritise reader comfort and readability of text, use neutral colours in graphs and text"*, was quite subjective, however, effort was taken in adhering to the design principles shown in Chapters 2,4,5 of [75], to create a flat, minimalistic, and stylish website, as opposed to other design styles such as skeuomorphism in Chapter 1 of [75]. The web interface respects Bootstrap's grid system and whitespace, and

utilizes a modern font (Josefin Sans) along with short line lengths, to improve readability and usability [76].

5.4.2 Dashboard

The dashboard was envisioned to be a place to control all the facets of the network, from registering with the bootstrap server to manually transmitting plates to peers. Hence, the normal command line commands are transferred over to be run-on-demand using buttons, coloured depending on their function. Care was taken to ensure harmful commands such as clearing everything from all databases had confirmation dialogs to provide an extra step for the user to ensure nothing that was necessary was deleted by accident.

By serializing the database data as a JSON object in the views, then passing onto the dashboard template, before formatting the data as a responsive table, database data was displayed fully for the user. Several improvements in the implementation were added to enhance the user experience. First, the number of each type of returned object (peers, plates, or violations) was displayed next to the respective section to give quick access to the amount of information in the network. Secondly, the tables were made to be sortable so the user could easily pick out peers from some location, or plates from some county (the beginning two letters of a license plate denote the area) if need be. Lastly, the banner image, a set of headlights, turn on and off depending on the status of the network – an elegant way of conveying information to the user without explicitly stating it.

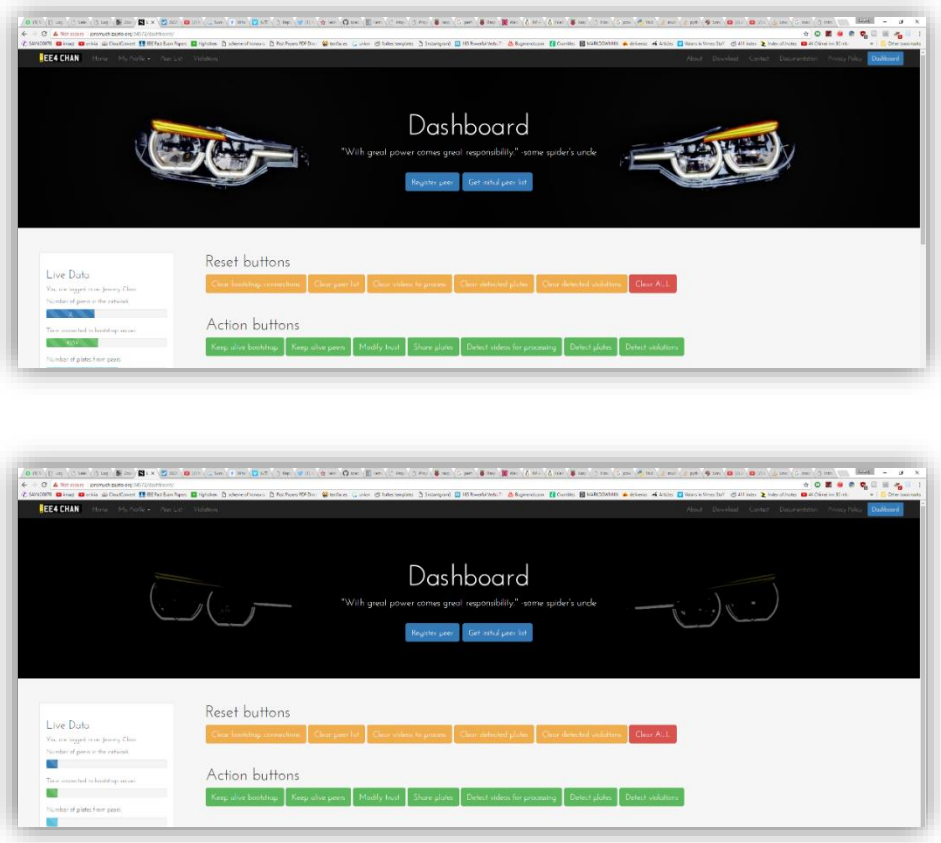


Figure 18: Headlights showing status of network (top – running, bottom – reset)

5.4.3 Peer List and Map

The dashboard contains the latitude and longitude of all the peers in the network, but provides no context on the city or country of the peer. Hence, a separate page was constructed to show the locations of all peers on a scrollable map, provided by the Google Maps Javascript API. The latitudes and longitudes are provided by the views, and passed to the template as a list of coordinates, before being fed into the API. A red marker was put on the coordinate of each peer (Figure 12), with a popup containing the IP address and port of the peer. The zoom level is determined automatically based on the distances between markers. Hence, a user can easily see locations of peers in their local area with this page.

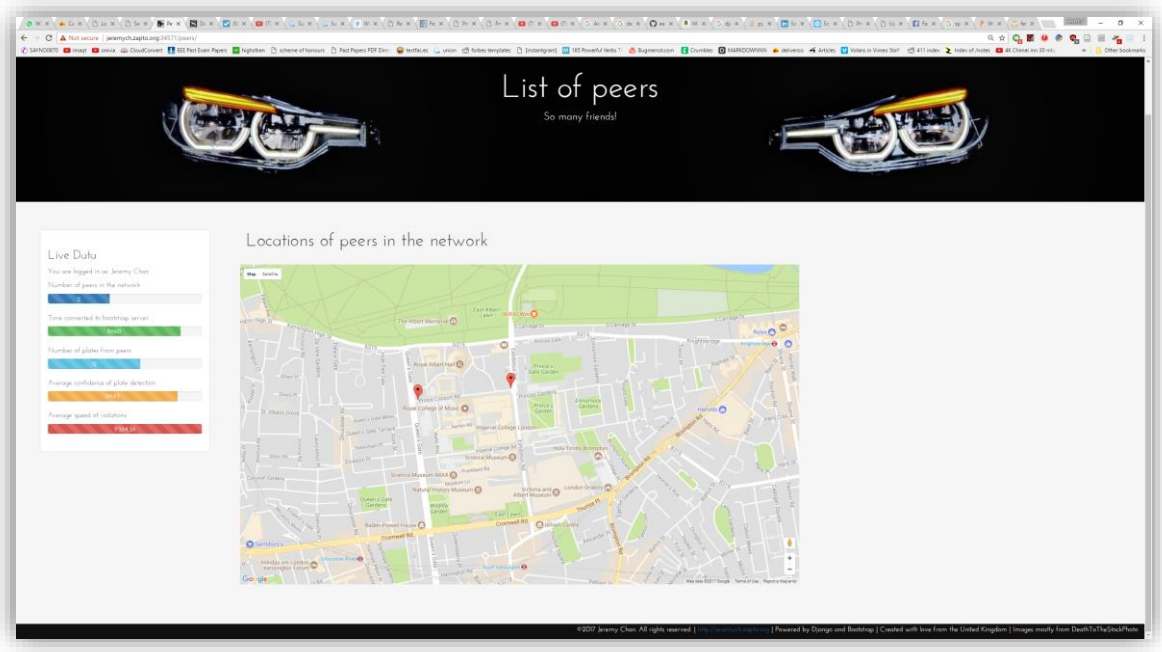


Figure 19: Peer Map

5.4.4 List of Violations

To implement the last requirement of the project, i.e., being able to publish photo evidence of any violations, a tab was implemented in the web interface to show a list of all violations. Data from the violations database was serialized, then put into a table using templates. The most critical data needed to be able to penalize an infraction is shown – the license plate, the first time of detection, the second time of detection, the driving distance in between the two locations, the average speed of the car, a map showing the route, and the two evidence images.

The map showing the route was generated dynamically using the Google Maps Static Maps API, with the route clearly shown in red to contrast the background, and two markers shown in blue to infer the start and end points, i.e. the locations of the two peers that detected the infraction.

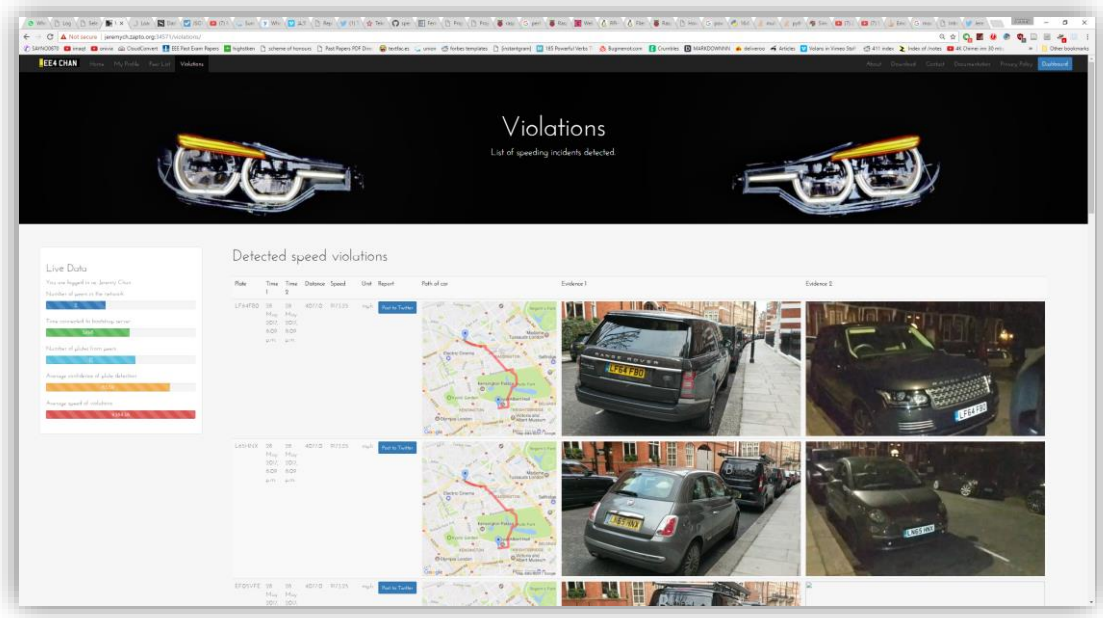


Figure 20: List of violations

These cosmetic features do not allow the publishing of photo evidence, however; the publishing of photo evidence was done through social networks as discussed in section XXX. Due to time constraints, only one social platform was utilized and interfaced with – Twitter, for simplicity.

A simple one-click button, Amazon style, was added to each row of the violations table. Clicking it set off a request to Twitter's API with the required information for a successful tweet – the API keys, the string to tweet, and the two evidence images. An example of a successful tweet is shown in Figure 20. By design, this request includes links to the evidence images instead of the images themselves. This implementation gave a faster response time, albeit small in normal use, between clicking the one click button to the post being visible on Twitter, as the evidence images were downloaded from Twitter's side as opposed to being uploaded to Twitter from the peer. Once tweeted, the evidence images are served from Twitter's servers and not directly from the peer.

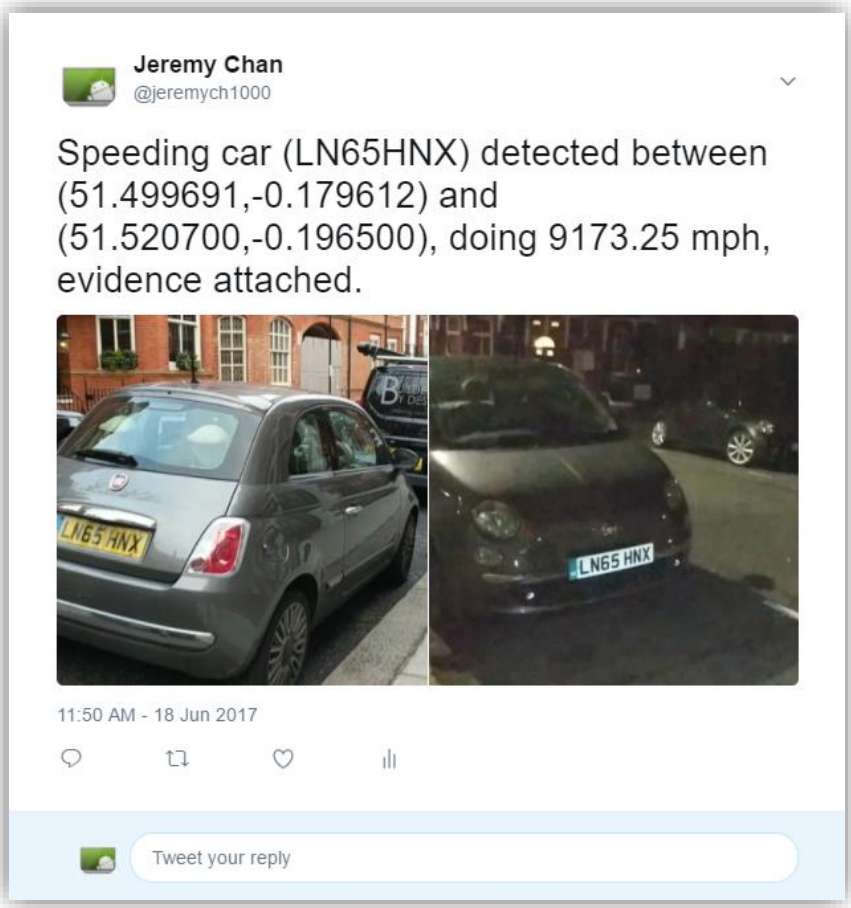


Figure 21: Twitter Post

5.4.5 Other Web Pages

To complete the web interface, several other pages were created. These new pages reflected what would be found in a production ready system, i.e. an about page, a contact page, a download and instructions page, and finally, a privacy policy and terms and conditions page. However, as this section was entirely flair, the level of detail was aimed at brevity instead of verbosity to save time for other development.

6 Testing

Testing was an important aspect of this project. Each module was extensively tested during development, either as part of the development process, or as part of an automated test suite. The sections below detail the tested procedures and results.

6.1 Methodology

Tests were conducted as part of the development process to ensure the final specifications were met at the conclusion of the project. Testing methods varied from manually writing entries in a logbook, running test commands on individual files, and the creation of an automated test suite for the license plate detection module.

6.2 License Plate Detection

The plates that the P2P network transmit depended on the effective extraction of license plates from videos of moving cars. Hence, having an accurate license plate detection module was desired. However, incorrect plates are not catastrophic as transmitting an incorrect plate was unlikely to cause any problems apart from a missed violation detection (akin to a disabled/inactive speed camera). False negatives, however, were preferred to false positives in all cases as false negatives posed no harm to the network and both the trust of the peers and the integrity of the system as a whole.

Real world testing was initially planned after confirming OpenALPR worked. However, several significant obstacles were encountered that prevented real world, real-time testing from ever coming to fruition.

For initial testing, it was difficult to find security camera footage that showed cars passing clearly past the camera, with good enough lighting and resolution.

It was extremely hard to find test spots, especially in central London. Finding a location with the following characteristics was extremely difficult and time consuming:

- on the ground floor to reduce distortion from not looking front on to a car
- sufficient shelter (had to be indoors to protect from theft and weather)
- mounting points for the tripod to hold the Raspberry Pi
- within close reach of a both a power supply and a Wi-Fi network

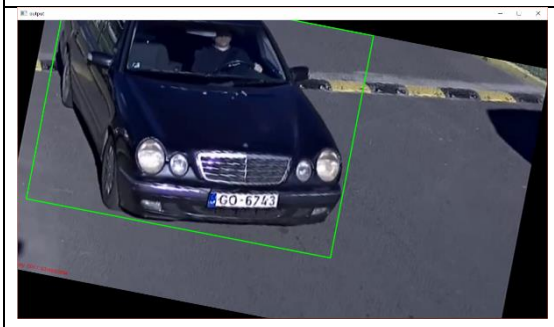
Moreover, had such a spot been found, the requirement for port forwarding the Django server would've likely posed another roadblock on secured Wi-Fi networks.

6.2.1 Still Pictures

The first step to verifying the license plate detection system was experimenting on still pictures. The input images to OpenALPR were already warped and rotated to ensure the license plate is rectangular.

Table 7: Still Picture ALPR

Input Image	LPR Output	Confidence

	LMN703	76.92
	GO673	77.80
	HF1699	73.78
	LS070WP	86.32

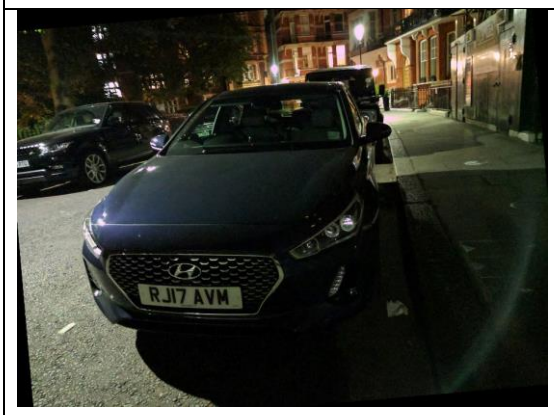
	LS58FXB	85.36
	RJ7AVM	82.19

Table 7 was therefore formed from a small subset of the test images used. From the results, OpenALPR was deemed to be working in daylight and at night (with street lighting), but with some imperfections such as missing a character occasionally (especially 1's), and misinterpreting O's as 0's.

Confident that OpenALPR was functioning as expected, the testing phase moved onto testing with video as opposed to still images as videos represented the final implementation more.

6.2.2 Walking Around

The next tests revolved around using test video gathered from walking around along a street, posing a different problem than video gathered from a fixed

position.

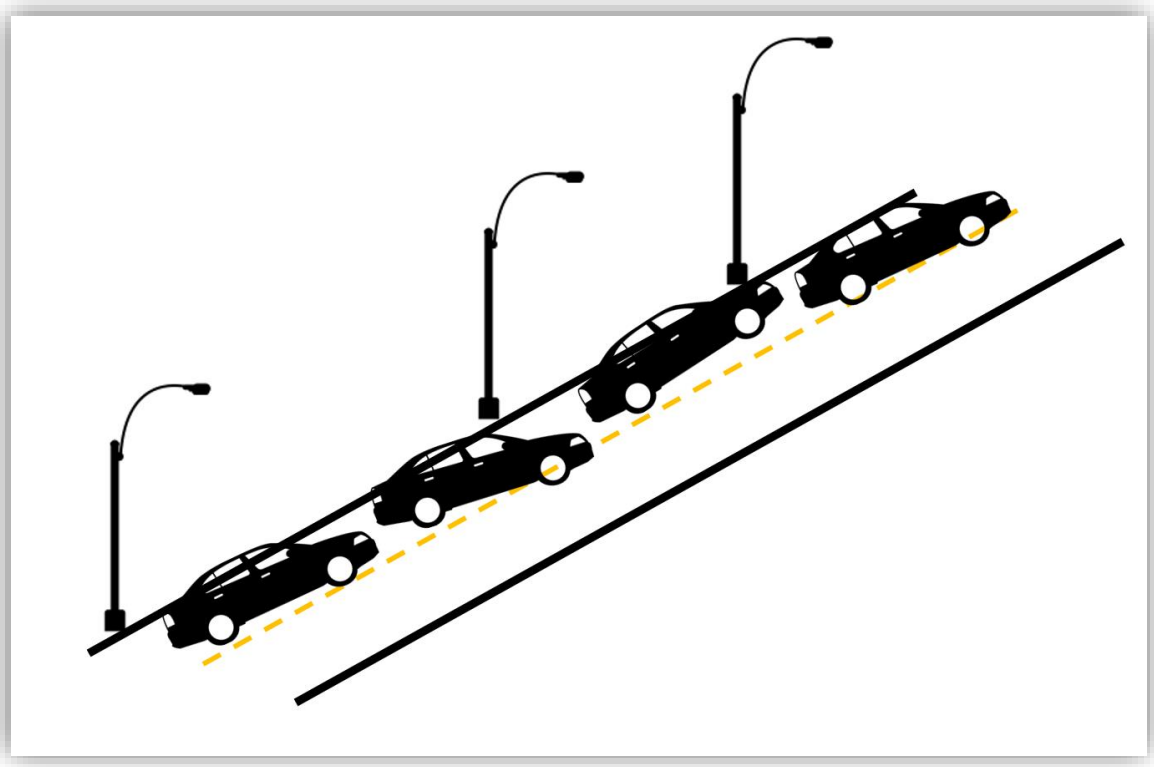


Figure 22: Cars parked in different positions on a road

Differences include, but not limited to: the change in lighting conditions, the change in the horizontal angle of the camera, and the position of plates (parked cars may be facing at an angle as opposed to a car driving directly at the camera, as shown in Figure 22).

Table 8: Walking around ALPR

Input Image	LPR Output	Confidence

	LR64FB0	85.54
	LR64FB0	85.39
	LRG4FB0	74.59
	55HNX	77.31
	LN65HNX	82.56

	LN65HNX	85.63
	EF15VFE	84.71
	EF15VFE	84.26
	EF5VFE	76.68

Table 8, formed from screenshots of videos of both the front and backs of consecutive cars parked on the side of a street, gave several interesting observations. First, OpenALPR performed generally well in detecting plate data. However, OpenALPR consistently misinterpreted some characters, especially O's as 0's.

Secondly, the more rotation and warping an image has undergone, the lower the output confidence from OpenALPR. Night time plates also, on average, gave a lower confidence – this was however very dependent on lighting. If ambient light struck the license plate directly, the plate would be overexposed, leading to no detected plates. If the light fell at an angle, it gave the optimal reflectivity for a high contrast between the black characters and white background. A side note was that the car headlights were all off, giving the camera ample chance of adjusting to the exposure levels needed to capture the plate.

Finally, OpenALPR tended to require very precise pre-processing in pre-warping the input image for all characters to be detected clearly. As each consecutive car was parked differently, the singular pre-warp matrix for each video caused some mistakes in OpenALPR – ideally each image would have its own manually calibrated pre-warp matrix.

The last remark suggested that given video captured from a fixed position, OpenALPR would perform much better.

6.2.3 Fixed Position

6.2.4 OpenALPR tweaks

6.2.5 Testing Suite

6.2.6 Performance and Summary

- Ability to detect a license plate after pre-processing
- Reliability in coming up with the same result from subsequent video frames
- Accuracy in detecting the correct license plate (not skipping any characters, numbers, not getting false readings such as O being 0, 6 being 5, etc.)
- Flexibility in detecting license plates of differing character lengths
- CPU time used in detecting a plate (real-time operation is required)
- Maximum resolution supported/required to ensure real-time operation

6.3 Peer to Peer Network

- Initially test while developing
- Use logbook to mark down stuff
- Save useful links/compilation instructions
- Hard to find good test spots (portable battery, need wifi, hard to develop on the go, need roads, hard to find good footage online)
- Started off by still pictures
- Then transitioned to videos of me walking around with parked cars (moving license plates anyway)
- Then transition to recording videos at the ends of roads

- Hard to find sheltered places to put camera overnight (safety, etc)
    - Dissapointing
- Most sheltered places are not on ground floor – leads to warping, hard to compensate, resolution problems
- Measure performance, FPS, CPU usage, memory
- Automated testing for correct plates (manually type out plates)
- Testing suite for new software changes

7 Evaluation

- User feedback on web interface
- Feel how useful this thing is
- What are main selling points
- What features would you be looking for?
- Is recording license plate moral?
- How effective is this system?
- How well can it detect plates? In what environment? In what weather? What camera angle?
- How fast can it share this data? How fast until notification to the user?
- Accurate sharing? To the right peers?
- How useful is the violation report shown to the user?

8 Deployment

- Github
- Readme
- Startup scripts
- CRON jobs

LIST OF REQUIRED PACKAGES AND PIP SUDO APT-GET STUFF

9 Conclusion and Future Work

10 References

[1] 'Number of speeding convictions from average speed cameras - a Freedom of Information request to Driver and Vehicle Licensing Agency', *WhatDoTheyKnow*, 28-Jul-2015. [Online]. Available: https://www.whatdotheyknow.com/request/number_of_speeding_convictions_f. [Accessed: 22-Jan-2017].

[2] 'Number of average speed cameras have doubled in three years', *This is Money*, 31-May-2016. [Online]. Available: http://www.thisismoney.co.uk/money/cars/article-3617584/Number-average-speed-cameras-doubled-three-years.html. [Accessed: 23-Jan-2017].

[3] 'FS_Speed.pdf'. [Online]. Available: http://m.swov.nl/rapport/Factsheets/UK/FS_Speed.pdf. [Accessed: 29-Jan-2017].

[4] 'Automatic Number Plate Recognition - Police.uk'. [Online]. Available: https://www.police.uk/information-and-advice/automatic-number-plate-recognition/. [Accessed: 29-Jan-2017].

[5] '2014_04_29-Californians-Overwhelmingly-Support-Use-of-License-Plate-Readers-and-Their-Ability-to-Solve-Crimes.pdf'. [Online]. Available: https://vigilantsolutions.com/wp-content/uploads/2014/04/2014_04_29-Californians-Overwhelmingly-Support-Use-of-License-Plate-Readers-and-Their-Ability-to-Solve-Crimes.pdf. [Accessed: 23-Jan-2017].

[6] 'inf104-vehicle-registration-numbers-and-number-plates.pdf'. [Online]. Available: https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/533255/inf104-vehicle-registration-numbers-and-number-plates.pdf. [Accessed: 29-Jan-2017].

[7] 'Raspberry Pi 3 Model B', *Raspberry Pi.* .

[8] 'The UK's Speed Camera Types | Fixed and Mobile speed cameras explained'. [Online]. Available: https://www.speedcamerasuk.com/speed-camera-types.htm. [Accessed: 23-Jan-2017].

[9] Y. D. Silva, 'Average speed enforcement (ASE) camera'. [Online]. Available: https://www.birmingham.gov.uk/info/20163/road_safety/364/average_speed_enforcement_ase_camera. [Accessed: 23-Jan-2017].

[10] 'specs3_vector_v1.1_final.pdf'. [Online]. Available: http://www.jenoptik.co.uk/sites/vysionics.vmdrupal04.lablateral.com/files/specs3_vector_v1.1_final.pdf. [Accessed: 23-Jan-2017].

[11] 'ARES | Fixed ALPR', *PlateSmart*, 18-Jun-2015. .

[12] 'LPR/ANPR License Plate Recognition SDK'. [Online]. Available: http://www.dtksoft.com/dtkanpr.php. [Accessed: 23-Jan-2017].

[13] 'OpenALPR Features'. [Online]. Available: http://www.openalpr.com/features.html. [Accessed: 23-Jan-2017].

[14] J. Warren, 'Bitmessage: A peer-to-peer message authentication and delivery system', *White Pap. 27 Novemb. 2012 Httpsbitmessage Orgbitmessage Pdf*, 2012.

[15] V. Sharma, P. Mathpal, and A. Kaushik, 'Automatic license plate recognition using optical character recognition and template matching on yellow color license plate', *Int. J. Innov. Res. Sci. Eng. Technol.*, vol. 3, no. 5, 2014.

[16] 'Accuracy Improvements — openalpr 2.2.0 documentation'. [Online]. Available: http://doc.openalpr.com/accuracy_improvements.html#openalpr-design. [Accessed: 24-Jan-2017].

[17] R. Azad, F. Davami, and B. Azad, 'A novel and robust method for automatic license plate recognition system based on pattern recognition', *Adv. Comput. Sci. Int. J.*, vol. 2, no. 3, pp. 64–70, 2013.

[18] S.-L. Chang, L.-S. Chen, Y.-C. Chung, and S.-W. Chen, 'Automatic License Plate Recognition', *IEEE Trans. Intell. Transp. Syst.*, vol. 5, no. 1, pp. 42–53, Mar. 2004.

[19] T. D. Duan, T. H. Du, T. V. Phuoc, and N. V. Hoang, 'Building an automatic vehicle license plate recognition system', in *Proc. Int. Conf. Comput. Sci. RIVF*, 2005, pp. 59–63.

[20] A. Badr, M. M. Abdelwahab, A. M. Thabet, and A. M. Abdelsadek, 'Automatic number plate recognition system', *Ann. Univ. Craiova-Math. Comput. Sci. Ser.*, vol. 38, no. 1, pp. 62–71, 2011.

[21] L. Liu, H. Zhang, A. Feng, X. Wan, and J. Guo, 'Simplified Local Binary Pattern Descriptor for Character Recognition of Vehicle License Plate', in *Imaging and Visualization 2010 Seventh International Conference on Computer Graphics*, 2010, pp. 157–161.

[22] T.-T. Nguyen and T. T. Nguyen, 'A real time license plate detection system based on boosting learning algorithm', in *Image and Signal Processing (CISP), 2012 5th International Congress on*, 2012, pp. 819–823.

[23] H. Kwaśnicka and B. Wawrzyniak, 'License plate localization and recognition in camera pictures', in *3rd Symposium on Methods of Artificial Intelligence*, 2002, pp. 243–246.

[24] 'OpenCV: Geometric Transformations of Images'. [Online]. Available: http://docs.opencv.org/3.1.0/da/d6e/tutorial_py_geometric_transformations.html. [Accessed: 25-Jan-2017].

[25] The Government of the Hong Kong Special Administrative Region, 'PEER-TO-PEER NETWORK'. [Online]. Available: http://www.infosec.gov.hk/english/technical/files/peer.pdf. [Accessed: 25-Jan-2017].

[26] H. Park, R. I. Ratzin, and M. van der Schaar, 'Peer-to-peer networksprotocols, cooperation and competition', *Streaming Media Archit. Tech. Appl. Recent Adv.*, pp. 262–294, 2010.

[27] C. Grothoff and C. GauthierDickey, 'Bootstrapping Peer-to-Peer Networks'. [Online]. Available: http://grothoff.org/christian/dasp2p.pdf. [Accessed: 21-Jan-2017].

[28] Q. H. Vu, M. Lupu, and B. C. Ooi, *Peer-to-Peer Computing: Principles and Applications*. Springer Science & Business Media, 2009.

[29] C. Mastroianni, D. Talia, and O. Verta, 'Designing an information system for Grids: Comparing hierarchical, decentralized P2P and super-peer models', *Parallel Comput.*, vol. 34, no. 10, pp. 593–611, Oct. 2008.

[30] *openalpr: Worse results after external preprocessing*. openalpr, 2017.

[31] 'TCP vs UDP - Difference and Comparison _ Diffen.pdf'. [Online]. Available: http://www.mrc.uidaho.edu/mrc/people/jff/443/Handouts/Ethernet/Specific%20Protocols/TCP%20vs%20UDP%20-%20Difference%20and%20Comparison%20_%20Diffen.pdf. [Accessed: 16-Jun-2017].

[32] 'A Beginners Guide To BitTorrent | morehawes'. .

[33] 'Peer-to-Peer Protocol (P2PP)', *Wikipedia*. 23-Nov-2016.

[34] 'networking - How do web-servers "listen" to IP addresses, interrupt or polling? - Super User'. [Online]. Available:

https://superuser.com/questions/837933/how-do-web-servers-listen-to-ip-addresses-interrupt-or-polling. [Accessed: 16-Jun-2017].

[35] S. Das, 'Which is Better, PHP or Python? A Developer's Take', *LinkedIn Pulse*, 11-Jun-2015. [Online]. Available: https://www.linkedin.com/pulse/which-better-php-python-developers-take-srikrishna-das. [Accessed: 19-Mar-2017].

[36] S. M. Srinivasan and R. S. Sangwan, 'Web App Security: A Comparison and Categorization of Testing Frameworks', *IEEE Softw.*, vol. 34, no. 1, pp. 99–102, Jan. 2017.

[37] Open Web Application Security Project, 'OWASP Top 10 - 2013'. OWASP.

[38] tutorialspoint.com, 'SQL RDBMS Concepts', *www.tutorialspoint.com*. [Online]. Available: https://www.tutorialspoint.com/sql/sql-rdbms-concepts.htm. [Accessed: 19-Mar-2017].

[39] 'NoSQL Databases Explained', *MongoDB*. [Online]. Available: https://www.mongodb.com/nosql-explained. [Accessed: 19-Mar-2017].

[40] 'MongoDB at Scale', *MongoDB*. [Online]. Available: https://www.mongodb.com/mongodb-scale. [Accessed: 19-Mar-2017].

[41] A. Nayak, A. Poriya, and D. Poojary, 'Type of NOSQL Databases and its Comparison with Relational Databases', *Int. J. Appl. Inf. Syst.*, vol. 5, no. 4, Mar. 2013.

[42] 'Models | Django documentation | Django'. [Online]. Available: https://docs.djangoproject.com/en/1.10/topics/db/models/. [Accessed: 20-Mar-2017].

[43] 'SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems', *DigitalOcean*. [Online]. Available: https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-

postgresql-a-comparison-of-relational-database-management-systems. [Accessed: 20-Mar-2017].

[44] 'Implementation Limits For SQLite'. [Online]. Available: https://www.sqlite.org/limits.html. [Accessed: 20-Mar-2017].

[45] 'NoSqlSupport – Django'. [Online]. Available: https://code.djangoproject.com/wiki/NoSqlSupport. [Accessed: 20-Mar-2017].

[46] N. Nurseitov, M. Paulson, R. Reynolds, and C. Izurieta, 'Comparison of JSON and XML Data Interchange Formats: A Case Study', Department of Computer Science Montana State University – Bozeman Bozeman, Montana, 59715, USA.

[47] A. Sumaray and S. K. Makki, 'A Comparison of Data Serialization Formats for Optimal Efficiency on a Mobile Platform', in *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication*, New York, NY, USA, 2012, p. 48:1–48:6.

[48] 'REST vs XML-RPC vs SOAP'. [Online]. Available: http://effbot.org/zone/rest-vs-rpc.htm. [Accessed: 20-Mar-2017].

[49] 'REST vs XML-RPC vs SOAP – pros and cons : Max Ivak Personal Site'. .

[50] 'How REST replaced SOAP on the Web: What it means to you', *InfoQ*. [Online]. Available: https://www.infoq.com/articles/rest-soap. [Accessed: 20-Mar-2017].

[51] 'Understanding REST And RPC For HTTP APIs', *Smashing Magazine*, 20-Sep-2016. [Online]. Available: https://www.smashingmagazine.com/2016/09/understanding-rest-and-rpc-for-http-apis/. [Accessed: 20-Mar-2017].

[52] 'Do you really know why you prefer REST over RPC? | API Handyman'. [Online]. Available: /do-you-really-know-why-you-prefer-rest-over-rpc/. [Accessed: 22-Mar-2017].

[53] 'OpenSSL Essentials: Working with SSL Certificates, Private Keys and CSRs', *DigitalOcean*. [Online]. Available: https://www.digitalocean.com/community/tutorials/openssl-essentials-working-with-ssl-certificates-private-keys-and-csrs. [Accessed: 17-Jun-2017].

[54] 'Advantages & Disadvantages of Symmetric Key Encryption', *It Still Works*. [Online]. Available: http://itstillworks.com/advantages-disadvantages-symmetric-key-encryption-2609.html. [Accessed: 17-Jun-2017].

[55] 'Description of Symmetric and Asymmetric Encryption'. [Online]. Available: https://support.microsoft.com/en-us/help/246071/description-of-symmetric-and-asymmetric-encryption. [Accessed: 17-Jun-2017].

[56] Mike, 'Python 3: An Intro to Encryption | The Mouse Vs. The Python'. .

[57] 'python - Does Django scale? - Stack Overflow'. [Online]. Available: https://stackoverflow.com/questions/886221/does-django-scale. [Accessed: 20-Jun-2017].

[58] 'The Model-View-Controller Design Pattern - Python Django Tutorials', *The Django Book*. .

[59] 'IntroductionDjango < TWiki < TWiki'. [Online]. Available: http://scipion.cnb.csic.es/old-docs/bin/view/TWiki/IntroductionDjango. [Accessed: 18-Jun-2017].

[60] 'camera - Raspistill slow to trigger? - Raspberry Pi Stack Exchange'. [Online]. Available: https://raspberrypi.stackexchange.com/questions/23698/raspistill-slow-to-trigger. [Accessed: 17-Jun-2017].

[61] ‘Raspberry Pi • View topic - RPi Cam Web Interface’. [Online]. Available: https://www.raspberrypi.org/forums/viewtopic.php?f=43&t=63276. [Accessed: 17-Jun-2017].

[62] ‘multithreading - Multiprocessing vs Threading Python - Stack Overflow’. [Online]. Available: https://stackoverflow.com/questions/3044580/multiprocessing-vs-threading-python. [Accessed: 17-Jun-2017].

[63] A. S. Shah, ‘Parallel Data Processing with MapReduce in Python’.

[64] ‘Raspberry Pi 3 - First Look’, *Piratical Tales From Pimoroni*, 29-Feb-2016. [Online]. Available: http://blog.pimoroni.com/raspberry-pi-3/. [Accessed: 17-Jun-2017].

[65] ‘lat lon - Measuring accuracy of latitude and longitude? - Geographic Information Systems Stack Exchange’. [Online]. Available: https://gis.stackexchange.com/questions/8650/measuring-accuracy-of-latitude-and-longitude/8674. [Accessed: 19-Jun-2017].

[66] ‘Required Precision for GPS Calculations - TresCopter’. [Online]. Available: http://www.trescopter.com/Home/concepts/required-precision-for-gps-calculations. [Accessed: 20-Jun-2017].

[67] ‘RFC 4122’. [Online]. Available: http://www.ietf.org/rfc/rfc4122.txt. [Accessed: 19-Jun-2017].

[68] Association of Chief Police Officers, ‘ACPO Speed Enforcement Policy Guidelines 2011 - 2015: Joining Forces for Safer Roads’. .

[69] *spec: Spec and acceptance tests for the Fernet format.* fernet, 2017.

[70] C. Sharkie and A. Fisher, *Jump Start Responsive Web Design*, 1 edition. Collingwood, VIC, Australia: SitePoint, 2013.

[71] T. Firdaus, *Responsive Web Design by Example*. Birmingham: Packt Publishing, 2013.

[72] N. Jain, 'Review of different responsive CSS Front-End Frameworks', *J. Glob. Res. Comput. Sci.*, vol. 5, no. 11, pp. 5–10, 2015.

[73] 'What are the pros and cons of using Bootstrap in web development? - Quora'. [Online]. Available: https://www.quora.com/What-are-the-pros-and-cons-of-using-Bootstrap-in-web-development. [Accessed: 21-Mar-2017].

[74] D. Cochran, *Twitter Bootstrap Web Development How-To*. Birminghan, UK: Packt Publishing, 2012.

[75] A. Pratas, *Creating Flat Design Websites*. Birmingham, UK: Packt Publishing, 2014.

[76] J. Ling and P. van Schaik, 'The influence of font type and line length on visual search and information retrieval in web pages', *Int. J. Hum.-Comput. Stud.*, vol. 64, no. 5, pp. 395–404, May 2006.

11 Appendix