Public Data Visualisation in Traffic, based on Local, Real-time Monitoring

Tom Bokhove

June 30, 2014

1 Acknowledgements

To start off this report, I would like to thank the research group Pervasive Systems, and in particular my supervisor, ir. J. Scholten, and chairman of my bachelor assignment committee, prof. dr. ing. P.J.M. Havinga, for their assistance during my bachelor assignment. I would also like to thank dr. H.K. Hemmes for being part of my bachelor assignment committee, and for his input during my bachelor assignment. This research is part of the COMMIT/SENSAFETY project (commit-nl.nl).

2 Abstract

Quite some people have smartphones nowadays. They often use them for personal purposes like gaming, planning, and education. Several of these apps for personal use make use of sensors or services that are available on the smartphone itself. These can include sensors like the accelerometer, light sensor, temperature sensor, magnetic field sensor, and gyroscope or services like GPS, Mobile advertisements, and Activity Recognition. While these sensors and services are very useful in personal applications, they can also be used for a larger purpose. This report discusses the implementation of a system that gathers data from several smartphones, and applies it to traffic situations, such that a public display can be created with accurate traffic information. The research question therefore is: "Is it possible to design a public system that can maximise throughput of cars in traffic situations, based on local, real-time, flexible monitoring using smartphones?".

For such a system, it is needed to collect this data from these smartphones somehow, and process it to a useful conclusion that can be made visible to the individual car driver. Three main components in the system can therefore be identified: A car driver as being the client to the system, a collecting unit consisting of an access point and a webserver, and a processing unit that allows data to come in and that processes this data. By installing an application on its smartphone, the car driver automatically sends data about its current location, velocity, and sudden decelerations to the collecting unit. The collecting unit collects all data from the smartphones of car drivers within a certain range. These collecting units can send data through to other collecting units, so that more data can be gathered, or to a processing unit. Data that reaches this processing unit will be processed, such that the public signals in the traffic can be changed.

During the scope of this project, a prototype was made to show the workings of the system in a demonstration environment. This prototype implemented the situation of a single crossroad with traffic lights. One of the roads (the crossing road) was hardcoded, which means that traffic was programmed to drive there, but no traffic was created there using a smartphone or collecting units. The ongoing road was used to show the workings of the app and collecting unit. The app sends out an identifier (to identify the smartphone or tablet), and a time till it reaches the traffic light. The collecting unit gathers this data via a webserver (the app sends data to this webserver), and sends it through to the processing unit. The processing unit makes up a scheme for the traffic lights, and changes the colour of the traffic lights whenever needed. Another thing that the processing unit does is to take a look into the future, as to predict what happens with the traffic lights. In that way, a speed can be given to car drivers at a distance of

200 metres away from the traffic light that indicates if they can still make it through the green light with their current speed or if they should drive faster or cannot make it all anymore. The collecting unit was implemented by an arduino on which an ethernet shield was stacked to create a webserver. The processing unit was implemented by only an arduino.

The prototype had several delays. Especially acquiring GPS data took a long time. Another thing that was measured, was the average velocity of the tablet when it was lying flat on the table. The average measured speed was about $0.4\frac{m}{s}$ without moving the tablet. This could be seen as an offset level once the app is used in a real-life application. The maximum velocity measured during leaving the tablet flat on the table without moving it was $1\frac{m}{s}$, which is mainly (except for the offset level) a random error that cannot be overcome.

When a request was made to the server, it depended on the time left till the intersection was reached what the chance was that a car would still make it through green light. It was shown that cars that had still the largest distance to cover had the largest chance of getting a green light once they reached the intersection.

3 Table of Contents

Contents

4 Introduction

Often, smartphones are used for daily life applications. There are lots of games available, as well as useful applications for planning, household, education, and a lot more. These applications can make use of features from within the smartphone itself, like accelerometers, gyroscopes, magnetic field sensors, ambient temperature sensors, and light sensors.[1] There are also features available from external sources. These can be among others GPS, Activity Recognition, Mobile advertisements, and Wallet fragments.[2] As mentioned, these features often are used for personal use like gaming or education. However, the fact that quite some people have smartphones, combined with the large availability of several sensors in these smartphones, makes it tempting to use these sensors for a larger purpose. One of these purposes can be traffic. TomTom, a company that produces navigation apparatus, already implements this idea in their TomTom Traffic HD products, by tracking not only active navigation apparatus, but also the location of mobile phones.[3]

The question now remains if participatory sensing or opportunistic sensing should be used.[4] In the first case, the owner of the smartphone knows that he is participating in giving away data to a sensor network via his smartphone. In the second case, the smartphone is noted by a sensor network and used if its state matches the requirements of the sensor network. Since the choice was made to gather data via sensors and services of the smartphone itself, the participatory sensing method was chosen, since permissions for these services and sensor usages has to be set within the application. On the other hand, a similar system can be built by sensing smartphones via opportunistic sensing. Smartphones within the reach of a sensor node, that would be somehow coupled to a car (for example via linking a unique acceleration of a smartphone to a vehicle, as was suggested in[5]), can be used for this.

There are already several sensors and services available in traffic. In case of a traffic light application in traffic, a common sensing method is the induction loop.[6] These induction loops are mounted into the road. The metal of a car passing over this induction loop in the road changes the magnetic field, due to which the induction loop can sense the presence of a car. In case of highways, there is traffic news on the radio, there are apps for the smartphone that can give accurate data about current traffic jams, and there are signal boards above and along the highway to indicate to the car driver that traffic jams are about to happen.

The TomTom Traffic apparatus is only available to customers from TomTom, which means it is not public. The induction loop is a local measure-

ment system, that is fixed in its position where it measures. Traffic news on the radio, and apps indicating traffic jams are good for the overall overview of the car driver, but cannot react to sudden road conditions that are happening real-time. Therefore, it is needed to go a step further. By using smartphones, it is possible to monitor the positions where cars are actually driving, and not just the prescribed positions of measurement from fixed sensors. Using smartphones also allows the system to monitor on a local scale traffic situations, even though the processing can be done on a more central scale, while returning information about the current road conditions back to the user on a local scale again. A system like this is able to control traffic lights, based on the current traffic intensity on the road, and the velocities of the individual cars. Also traffic jams on highways can be detected directly, and warnings can be sent to cars that are about to go into this traffic jam via signal boards above the highway, or coloured lamps next to mile markers. A system that can be applied to both applications, is shown in figure 1.

Using his smartphone, a car driver can gather data about his velocity, position, and sudden acceleration by using GPS and the internal accelerometer. This data is gathered in an application on the smartphone itself. The application uses the Dynamic Host Configuration Protocol (DHCP) to find the IP-address of the nearest Wireless Access Point (WAP) of the system, and connects to this access point. The Wireless Access Point in combination with a webserver is in this report called the "collecting unit". The application on the smartphone of the car driver can now send its data to this webserver. A collecting unit has a certain range in which data can be sent to this unit. Outside of that range, other collecting units have to take over. They communicate with each other via radio communication. When enough data has been collected, it has to be processed. This happens in the "processing unit". The processing unit gets as input this data, and changes as output the contents on the public display. Processing units often are the end-points of the radio communication network, whereas the collecting units are routers. Finally, the display gives back information to the car driver about the current road conditions.

This report will discuss all aspects of applying this system in traffic. It will also discuss a basic prototype that was made to show the major operation of the system. A research question can be formulated for this report as follows: "Is it possible to design a public system that can maximise throughput of cars in traffic situations, based on local, real-time, flexible monitoring using smartphones?"

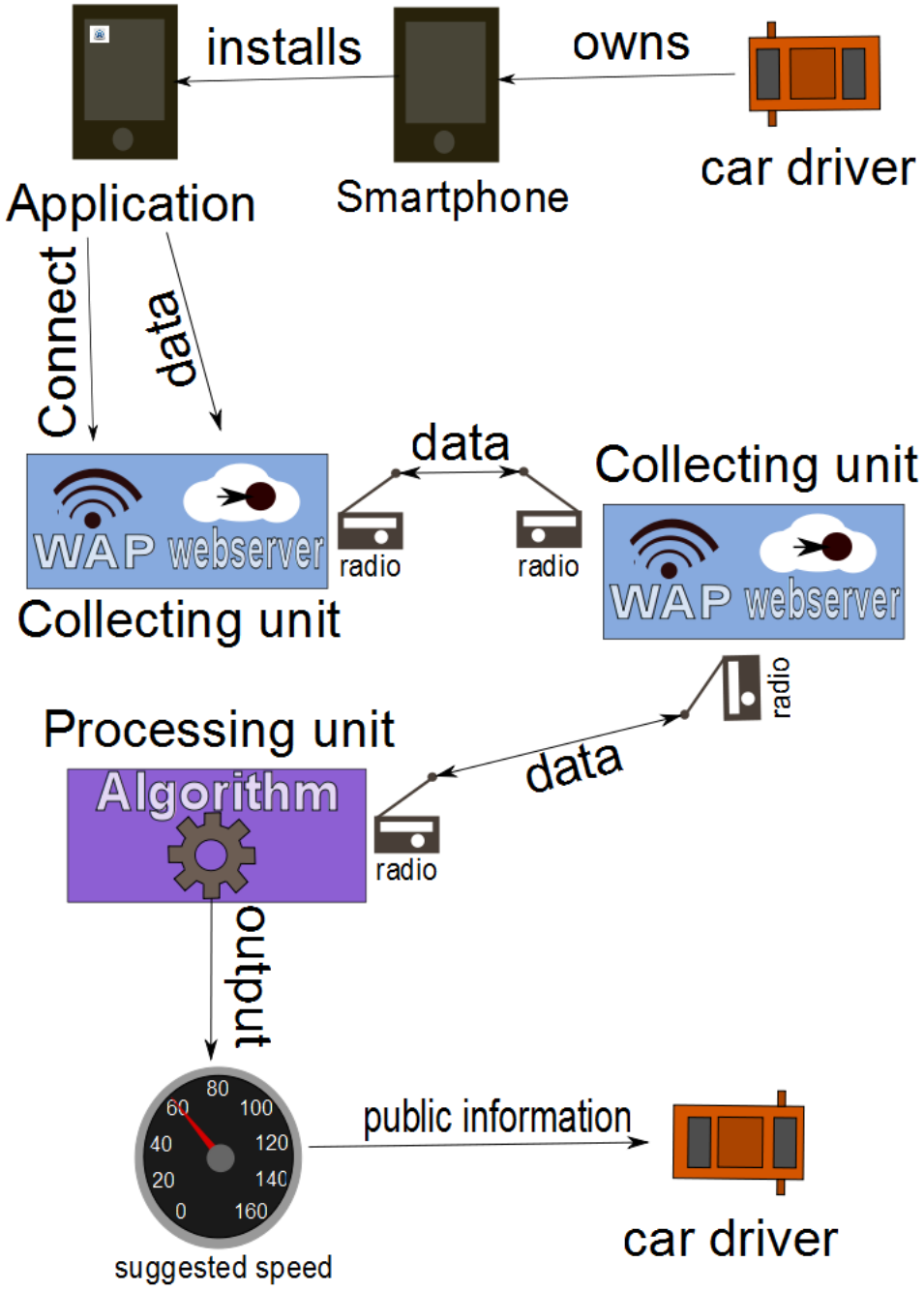

Figure 1: An overview of a system that can monitor traffic real-time and draw conclusions from this.

5 Acquiring Data

5.1 Use of Smartphones

To make a system that can monitor traffic, and make real-time conclusions about the current road conditions, data is needed that can be supplied to the system. This data can be retrieved in many ways. It is possible to use sensors along the road, induction loops (most often used in combination with traffic lights), journalistic data, and many more other data sources. They are all useful options to predict the traffic, but in this system the main source of data was chosen to be a smartphone. Using smartphones as source of data means that you can track individual cars, without the need of additional hardware for the car driver. It becomes possible now to track only those pieces of the road where cars are driving, while still being able to process data on a more central scale by using multiple collecting units (see section 6.1).

The next step would be to make an application for the smartphone, that is available to as many users as possible. Looking to the market shares of smartphone operating systems worldwide in quartile 1 of 2014, it can be seen that Android is by far the most used operating system for smartphones with a market share of 81.1%, followed by iOS with 15.2% and Windows Phone with 2.7%.[7] The ideal situation would be that everyone has a smartphone, and that everyone uses the application. In practice that is not possible. Take for example the Netherlands. On the 13^{th} of December 2013 the "Volkskrant" published an article about the fact that there were for the first time more people with a smartphone than a desktop computer. They claimed the percentage of people owning a smartphone at that time was 67%.[8] Combining the facts that not everyone owns a smartphone, not everyone has a supported operating system, and not everyone installs the application, would mean that the system should be able to make fair assumptions about all the traffic based solely on part of all car drivers that satisfy the demands for running the application.

5.2 Implementation of this Application

To give a little more insight in the functionality of such an app, a UML-diagram can be made. This diagram is shown in figure 2.

For the basics of this diagram, an Android application (also used in the prototype described in section 8.3) was taken, made in the programming language Java. This because Android is worldwide by far the most used operating system, as was shown in section 5.1. The same basic idea however holds for other operating systems as well. The class MyTimer is central in

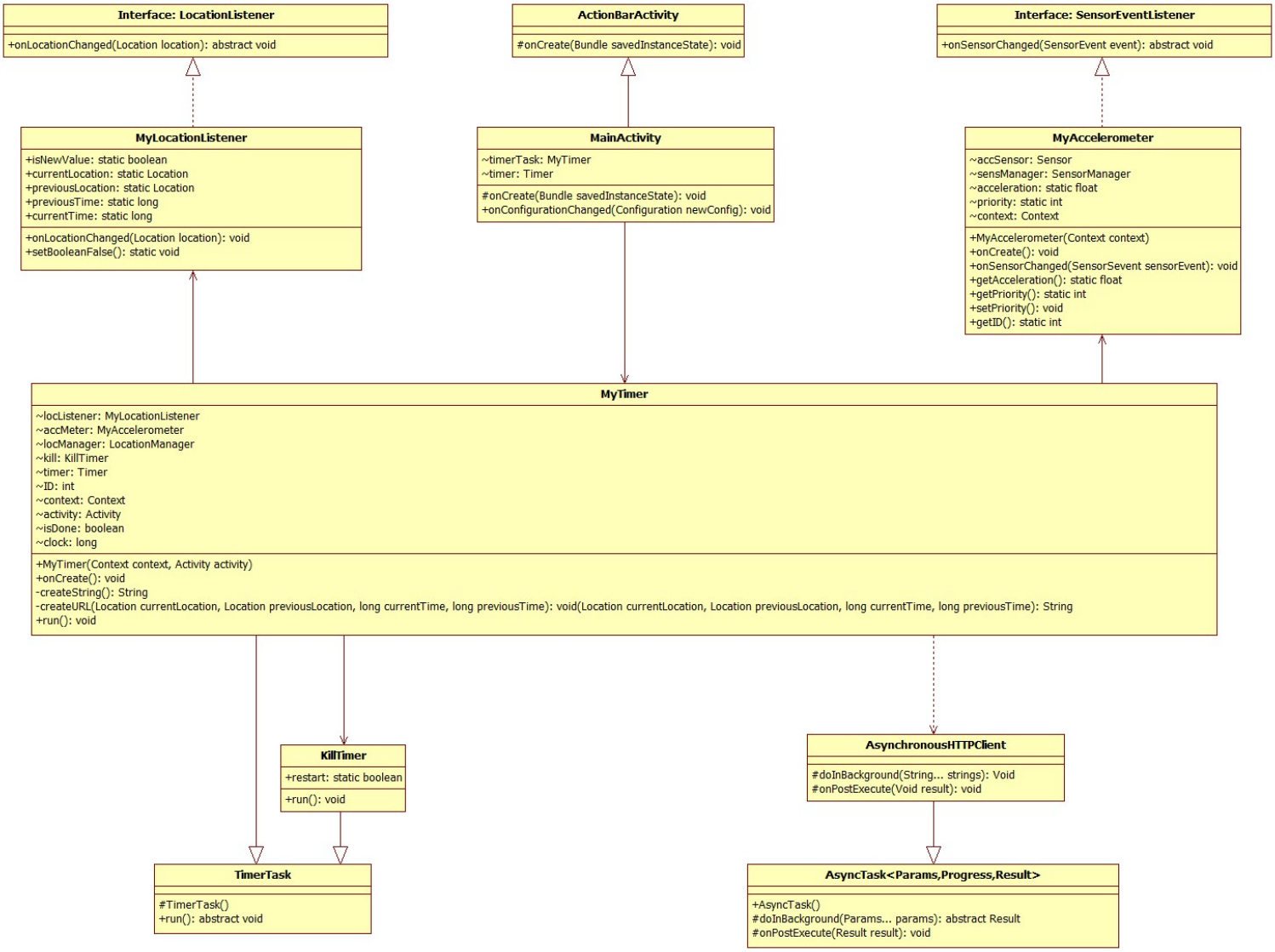

Figure 2: A UML-diagram of a possible implementation of the system described in this report.

this UML. It makes sure that every 15 seconds new data is sent to a collecting unit (collecting units are described in section 6.1). The time limit is needed so that the GPS service is not always active, since it drains a lot of power from the battery of the tablet or mobile phone.

This GPS service makes use of satellites, wifi, and cell towers, and is accessed via the class LocationManager, and Interface LocationListener. The class LocationManager is not shown on the UML diagram, since it is a standard class available via the Android SDK. The interface LocationListener is implemented in the class MyLocationListener. The manager makes sure to search for GPS providers in the vicinity of the tablet, and it also gives the programmer the option to exclude satellite location services (up to API 17 (Android 4.2, Jelly Bean), afterwards satellites were included on default). The GPS service returns an object of type Location, containing among others a latitude and longitude. The class Location also contains a method called distanceBetween(), which returns the distance between two locations in metres. A simple SystemClock was used to determine the time interval between two locations. The method getSpeed() was not used, since it is not supported among all Android devices, and therefore would simply return 0 in these cases, making the application not useful on these devices. This choice was made, since in section 5.1 it was mentioned that the more devices that can supply data to the system, the better the system can function.

Even though the system is fully based on gaining velocities from the cars (i.e. a speed and a direction), it might sometimes be helpful to use the internal accelerometer. Not all applications of this system will use this feature, but a good way of using this sensor is on a highway. Retrieving GPS locations can be slow, but if traffic jams start to occur, you want to know this as soon as possible. The internal accelerometer is always able to do measurements. If it would encounter a situation in which a car would brake very fast, the internal accelerometer would be able to detect this directly, and send a sign to the units along the highway. If more tablets send in signals like that to a certain unit, it is possible to isolate this unit a little. Isolation means in this case that the unit itself focuses mainly on acquiring new data from the car drivers via GPS to confirm the claims made by the accelerometer, and that surrounding units take less data transfer, so that they are ready when the isolated unit wants to send out a warning to them. In the case of the highway, cars that will soon drive into this new traffic jam have to be warned in advance. Therefore, previous units will receive warning messages from the isolated unit when appropriate, and these warning messages should get the highest priority. Another advantage of the use of accelerometers above the use of GPS is that the accelerometer is inside of the tablet, which means that it needs a lot less battery power than external services like GPS.

The accelerometer can also serve another purpose. It is useful to identify seperate car drivers, by assigning them a unique identifier. These identifiers can be used to indicate the direction the car is driving at (see section 6.3), but also allow for storage of data sorted by the unique identifier of a car. The accelerometer has noise of 8 digits behind the comma, this can be used for a unique identifier with 8 digits, that is (almost) purely random. There are build-in functions in Java that try to make random identifiers, but they are pseudo-random, meaning they use the same algorithm to come to a value on all tablets and smartphones.

Another timer that is used, is the KillTimer class, extending the TimerTask class (just like MyTimer does). This class makes sure that if no data is sent to units for five minutes, it resets all data currently available, since it is no longer relevant for the current traffic situation. Also, the identifier is deleted and a new one is created for privacy purposes, so that cars cannot be tracked by others based on their identifier.

Finally, an AsynchronousHTTPCient was created, by extending the class AsyncTask. AsyncTask allows the programmer to execute a thread outside of the MainActivity class. In this case an HTTP client was made, that will send the data to the collecting unit. More about the collecting unit, and how data is sent to this unit, will be explained in section 6.1.

6 Collecting and Processing Data

6.1 Collecting Unit

To gather data from all seperate car drivers, collecting units are needed. To do so, there are three basic functions they should take care of: Set up a wireless network to which clients can connect (the car drivers with their tablets), create a webserver to which the data can be sent, and make a connection to the processing unit (possible via other collecting units). This connection is described in section 6.3.

To do the task of letting clients connect to a wireless network, a wireless access point would be sufficient that reaches up to about half a kilometre. This means that the connection would have a range of 1 kilometre (half a kilometre on both sides of the access point) to connect to for the clients. One thing to keep in mind is that the data transfer speed and the availability of a connection should be sufficient along the full kilometre, so that car drivers can always make contact with these access points in a reasonably small time. On a highway, when driving with 120 kilometres per hour for example, just 1 kilometre range means that the car driver has 30 seconds to connect to the access point, and send its data, which is relatively short. Note that the application needs to use DHCP (Dynamic Host Configuration Protocol) to get the IP-addresses dynamically, so that a connection can be made with the access point.

Wireless repeaters (often called ”range extenders”) can be used to extend the network of the wireless access point. By placing the repeater just within the range of the network, the repeater will amplify the signal so that others can connect to it outside of the range of the original network. It should be noted that using only repeaters and one network source (router,access point, etc.) is not a good idea: repeaters do give good connections, but the data transfer speed drops significantly after repeating a signal several times. The webserver can also be connected to this network, which means that the network should not only be sufficient as to give every car driver access to it, but it should also be quick and stable enough to give the webserver space to operate.

The webserver itself needs a solid connection to operate. To keep the reference name to this webserver the same, it is possible to use the Dynamic Domain Name System (Dynamic DNS or simply DDNS), while still assigning IP-addresses with DHCP. This DDNS protocol updates the IP-address behind the reference name to the webserver, so that the URL stays the same over time. In that way, the URLs can be hardcoded within the application or downloaded when someone enters a road. Another way is to

simply download the currently active IP-addresses of the webserver when entering a road, and directly refer to them without a reference name.

Since we want to transfer short messages of data, this can simply be done by including them in the URL, for example:

$http://xxx.xx.xx.xxx?i = 12345678\&p = 025\&v = 22025\&a = 52365\&o = 06282$

In this URL, the x's represent a specific IP-adress (or a reference name that refers to an IP-address assigned using DHCP), unique for the webserver linked to an access point. The question mark means that the IP-address is over, and the information starts to be given to the server. The letters represent: ID (i), Priority (p), Velocity in $\frac{m}{s}$ (v), Latitude in degrees (a), and Longitude in degrees (o). Note that velocities often contain digits behind the comma. To avoid those in the URLs, the first two digits of the URL after "v=" are before the comma, and the last three digits are after the comma (the maximum speed is 99.999 $\frac{m}{s}$, which is about 360 $\frac{km}{h}$). The same holds for the latitudes and longitudes. The ID and priority are both already integers, and therefore do not possess this property. Note that numbers have a predefined length. This means that in the case of the longitude for example, there should be a 0 before the 6 to indicate that there is a 0 followed by a 6 before the comma. If this 0 would be ommitted, it would say 62.82 for example, and it would give an error, since the longitude is not 5 digits long anymore.

Finally, the connection between collecting units and other collecting units, or collecting units and processing units can be done via radio communication. To be able to place the collecting units 1 kilometre apart, as suggested earlier in this section, these radios have to overcome this distance. The radios are there to seperate the data transfer from the clients (car drivers, which go via HTTP) and the data transfer between units (the radio connection).

6.2 Processing Unit

The processing unit processes incoming data and executes tasks based on this incoming data. If needed, it can store this data for later processing. The processing unit changes the current conditions of the traffic lights on a crossroad, or changes the current maximum speed that should be driven on a road, based on the current traffic density, and velocities. A processing unit in general does not have the capability of sending data to another unit. Some applications however need combined capabilities of both a processing unit and the routing property of a collecting unit. In some applications, like

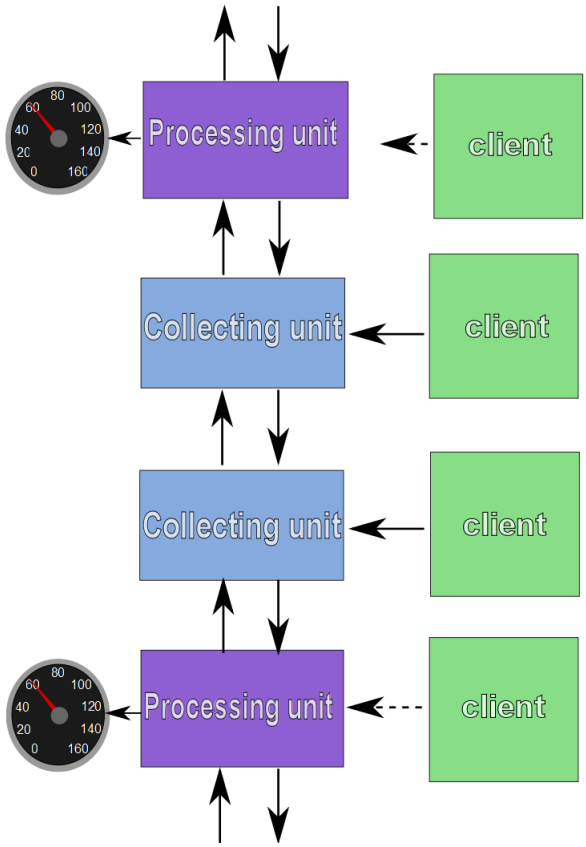

Figure 3: The use of processing units as combined processing and collecting units. The processing units have routing capabilities, and if needed also collecting capabilities. The protocol shown in figure 4 is left out for simplicity, but nevertheless still applies.

the highway application in section 7.2, these processing units even need the full capabilities of a collecting unit(both routing and collecting). In those cases, the processing unit, and (part of) the functionality of the collecting unit can be combined into one unit, as is shown in figure 3. In this figure, the possible collecting option is shown by a dashed arrow. Also note that processing units are still the only one that can process data into an action like switching the light of a traffic light, or giving a speed indication on a signal board along a road.

6.3 Data Transfer between Units

In figure 4 a processing scheme is shown. It shows the basic functionality of the collecting unit upon communication with another collecting unit or with a processing unit, it also shows the data transfer from smartphones or tablets to the collecting unit.

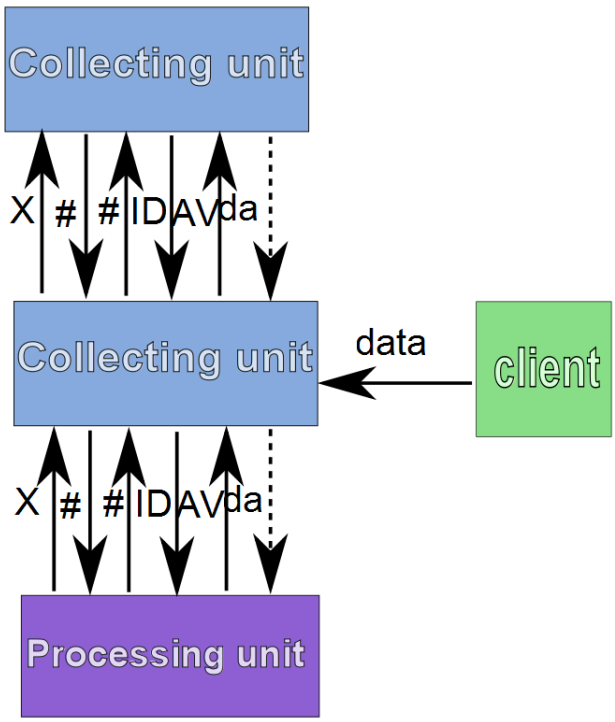

Figure 4: A block diagram that shows the relation between a collecting unit and another collecting unit, a processing unit, and a client. The solid arrow shows steps that always have to happen when a connection between two units is established, a dashed arrow shows a possible step, but not necessary to complete the process. The symbols are explained in section 6.3.

The relation between the client and the collecting unit is quite straight forward. As was mentioned in section 5.2, the tablet of the car driver has the build-in feature of an HTTPClient via which it can send data to a webserver,

which is part of the collecting unit. A URL was used to transfer data, as shown in section 6.1. This data communication from client to the webserver in a collecting unit is one-way traffic. The client can only send data to the webserver, but the webserver will not respond to these data transfers. From a commercial perspective such a feedback relation might still be established, so that information can be sent back to the user about current road conditions. This would give him an advantage over others for just installing the application, but for this report it would go against the idea of creating a public display, which means the display should be accessible to everyone.

The relation between a collecting unit and another collecting unit or a collecting unit and a processing unit is about the same. The main difference is however that a processing unit does not allow for two-way data traffic, whereas data transfer between two collecting units does allow for two-way data transfer. This is since the processing unit in general can only receive data, and not transmit data. If two-way traffic is needed, a solution is given in section 3, which was explained in section 6.2 in more depth. Another difference is that a collecting unit directly gets data from car drivers, whereas processing units do not establish a direct connection between themselves and the car drivers, they get their data completely from other collecting units. The steps for data transfer between units are as follows:

1. If a unit, unit A, is not transferring data already, and it can receive data, then it is transmitting a character X. X can be any character except for a number due to the confusion with data that is sent, which consists out of numbers. Units close to each other should not have the same character, so that they can be distinguished locally.

2. If another unit, unit B, needs to send data to unit A, it will give its own unique identifier (given as "#" in the figure) to unit A upon seeing the character X.

3. If the unique identifier of unit B reaches unit A before the identifier of other units reaches unit A, then data transfer is set up between unit A and unit B. To confirm this, unit A sends out the unique identifier of unit B.

4. Unit B sends out the identifier of the car driver he has information about (so this is the identifier obtained from the tablet, indicated as "ID" in the figure).

5. Unit A checks if it has this identifier already in its system.

6. If unit A has this identifier also in its system (i.e. it is available, in the figure indicated as "AV"), it sends out a "true" boolean, which

means the car passed first unit A, and then unit B. If the identifier is not in the system of unit A, it means it still has to pass unit A, and a boolean "false" is sent. This boolean gives an indication about the direction the car is driving at.

7. If needed, data (indicated as "da") is now sent about the velocity, position, and priority from unit B to unit A. In the case of a crossroad with traffic lights, it is no longer necessary to store data about the position and velocity of a car driver, once it passed the traffic light. It might be convenient to use this data on a next crossroad with traffic lights in the vicinity of the old traffic light, but otherwise this data is no longer needed, and does not have to be processed. This possible, but not necessary connection is indicated in the figure with a dashed arrow.

7 Applications

7.1 Traffic Lights

7.1.1 One Crossroad

The application of a crossroad with traffic lights was already mentioned several times as example in the previous chapters. This application is also used for the prototype that will be explained in chapter 8. This application will be slightly extended in this paragraph. Besides just monitoring traffic lights, an additional feature will be introduced, which is unique for a system that can monitor over a longer range. This feature is a signal board at 200 metres from the traffic light, that indicates how fast you should drive to still be able to reach the traffic light when it is green. The target of this application is to maximise throughput based on a local monitoring scheme. An overview of this system is shown in figure 5.

For one crossroad, a single processing unit is needed, that does not have to deal with any other processing units. Therefore, this processing unit only receives data, and controls the traffic lights with that data. The collecting units are along the four roads leading to this intersection. Monitoring about two kilometres in all four directions for a road on which the maximum speed is 80 $\frac{km}{h}$ would mean that someone driving at maximum speed would be monitored for 1.5 minutes. For monitoring in four directions this would make up for 8 collecting units positioned at 500 and 1500 metres away from the intersection in every direction (the solid-connection range of a collecting unit was set to 0.5 kilometres, which means in both directions that this is 1 kilometre in total that the unit is able to cover, as was explained in section 6.1). Only the collecting units closest to the traffic light (those at 500 metres distance) are shown in the figure. Note also from the figure that traffic that already passed the crossroad is no longer monitored, since the information of those cars is no longer needed for controlling the traffic light.

It has to be kept in mind that there are several regulations that have to be implemented to maximise throughput:

- Traffic that is still driving at maximum speed gets priority over traffic that already is waiting for a traffic light, since stopping the driving traffic would cost more time then to let them continue and let the stopped traffic wait a little longer. Note however that this might cause starvation, and therefore the next bullet point is introduced in the system.
- No road should always have red light, even if no traffic is driving there. The reason for this is that (as was discussed in section 5.1) not

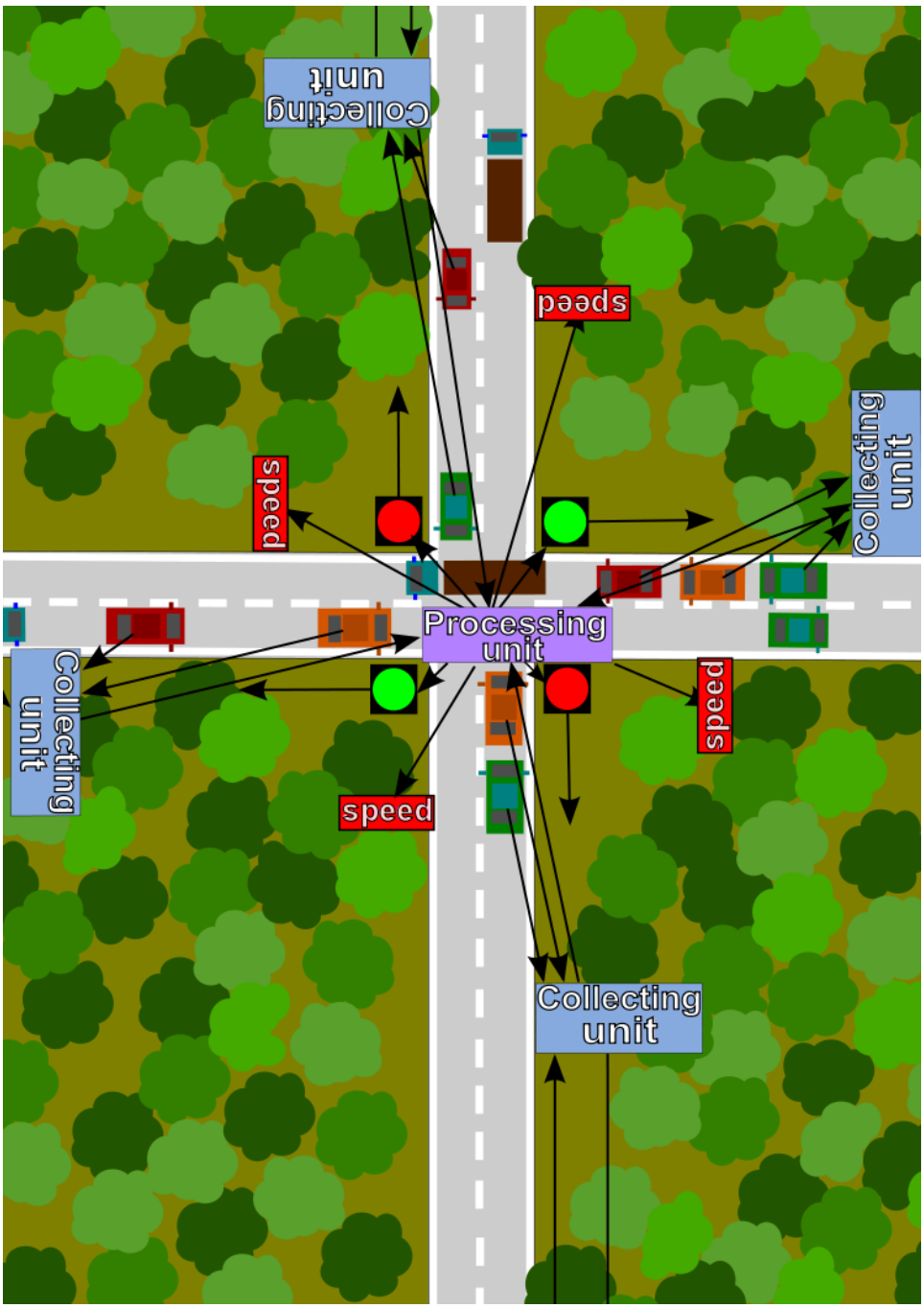

Figure 5: Crossroad that shows an example of the use of a real-time monitoring system in a traffic light application. Red "Speed" units are signal boards that indicate how fast to drive to still get a green light at the traffic light. Blue "collecting" units (only the closest collecting units are given), and a purple "processing" unit are given as well. The arrows indicate the direction of signals. The figure is not on scale.

everyone owns a smartphone that is capable of sending data. If the traffic lights would never get green, those people would have to wait for a long time.

- Groups of cars should get priority over individual cars, since there are more people in a group, so that means that less car drivers have to wait for a traffic light, and the throughput is increased.

7.1.2 Multiple Crossroads

With several crossroads that are linked, there are some issues that have to be accounted for. First of all, processing units are no longer stand-alone devices that only receive data. They have to communicate with each other, so that the optimal solution for turning traffic lights green can be found. The use of collecting units from outside of the cluster of linked crossroads is now the input data to this group of linked crossroads. Based on this data, an accurate estimation should be made about the timing scheme of the traffic lights on all these crossroads at the same time. Only by being able to control all of these traffic lights on all these crossroads at once, one is able to make the best processing scheme. In this section, it is tried to make a processing scheme that gives all power to the in- and outgoing roads of this series of crossroads, so that a system can be made that functions still based upon intensity of cars on the several in- and outgoing roads, and not on traffic that is stuck within the series of crossroads.

A system like this, that uses data from outside of the series of linked crossroads, wants to have full control about the traffic within this series of crossroads. An optimal solution would be to see the series of crossroads as a black box, with traffic coming in and going out. Those roads with in- and outgoing traffic are from now on called ”independent roads”, since they send in traffic independent of what happens on the other roads in the series of crossroads. To give back control to these independent roads, the black box itself should be managed as such that everything that goes in with a green light, can travel through this black box at once, without having to wait within the series of crossroads somewhere for a traffic light. If this would succeed, then the system would look again a lot like the single crossroad system, only with more inputs and outputs (so more independent roads). The advantage of such an approach is that it is fully controlled by the collecting units that gather data, so that traffic lights can again be controlled by the amount of cars coming in, and thus by the collecting units along the independent roads.

One of the techniques that can be used to get cars from one side of this blackbox (input) to the other side (output) would be via a so-called

"green wave".[9] The green wave allows cars from independent roads to pass a full series of traffic lights straight ahead of them by making all these traffic lights green once the cars reach these traffic lights. By using the green wave technique, in combination with the black box approach, one has to overcome the following obstacles:

- Traffic that is stuck within the blackbox with crossroads should be cleared as soon as possible. The processing units should take care of monitoring these cars, so that additional time is added to a green wave to allow the traffic that was stuck, as well as the throughtraffic, to make it to the end of the green wave.

- Another point is to see what road to give priority. In the case of a single crossroad, the road that increases the throughput the most, is the one that gets green light. In the case of several crossroads, it is possible to encounter the problem shown in figure 6. Would the blue roads be preferred to get green light, since they allow 28 cars to pass at once, or would the orange roads be preferred, since there are 20 cars waiting for one road, which might cause other traffic on that road to get stuck as well. In this case, a solution should be made that suits the conditions the best. To keep the original target, the largest throughput would be the best, but on crossroads with roads that cannot get that much traffic stuck on them, it is useful to give priority to the road with the more cars on it. Note that once the blue roads allowed enough cars to pass, the orange roads would get a larger amount of cars waiting, which means they get the green light afterwards. This also means that, whenever possible, the blue road should get preference, since the problem with the stuck traffic on the orange road will solve itself naturally (or if not, it will get green light by the fact that every light should get green once every so many minutes).

- Again, one should keep in mind that there are cars driving there that are not monitored, since they do not have a working version of the application. This means that any traffic light has to become green every so many minutes, and that by calculating the time for a green wave, one has to use probability theory to determine how many cars are stuck in the streets based on the amount of signals received from cars that do have the application, and how many cars are actually waiting for a traffic light. Another advantage of giving green light to every traffic light at least once every so many minutes, is that it avoids starvation of roads that do not get access to the green light based on the implemented algorithm.

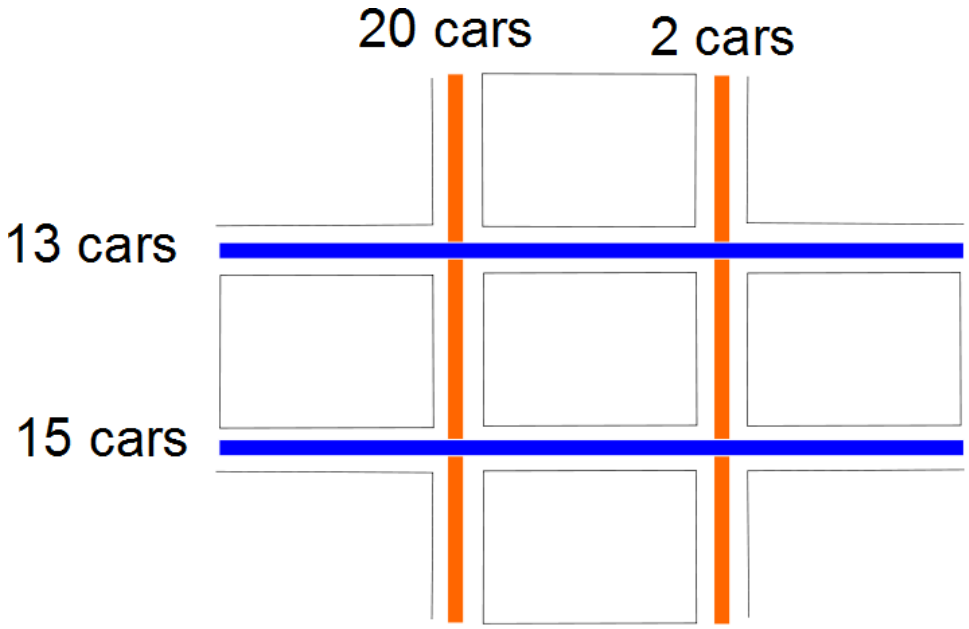

Figure 6: Four crossroads that show a dilemma. Two horizontal or two vertical roads can drive at the same time. Choose for largest throughput (blue) or largest build-up of cars (orange)?

7.1.3 Turning Left or Right

So far, all cars have travelled straight over crossroads. While often most cars will continue on the road they were driving at, there are also several cars that will turn left or right. Turning right is often not a real problem, since you are not crossing the opposite traffic that has green light as well. Turning left however is more difficult. The car will have to go over the lane of the opposite traffic, which means that it either should get a green light for turning left, meaning the opposite traffic is stopped, or it should take the chance and go left on its own responsibility, with the opposite traffic still allowed to continue.

The first option, giving the cars an additional green light for turning left, is useful when the road is very busy. If the main road has traffic in two directions, this means that all throughtraffic gets priority. Once all throughtraffic has been processed, the traffic lights for turning left are getting processed. First one direction of the traffic gets all green lights for turning left, then the other direction.

If the road is not that busy, it is possible to consider a ”flashing-yellow arrow light”.[10] The flashing yellow arrow light is especially made for left-turning traffic and means that most of the opposite traffic has passed, but

that still caution has to be paid to the fact that there might be more opposite traffic. Also note that it is possible that opposite left-turning traffic gets a similar sign.

To keep the green wave example, and therefore still use a blackbox of a series of crossroads, one has to keep in mind that left or right turning traffic is also traffic that potentially gets stuck in the series of crossroads. When taking the example of figure 6 once more, if someone enters the series of crossroads via the blue road labelled by "13 cars", and would turn to the right, it would encounter a red light, since both blue roads have a green wave, and therefore both orange roads do not allow traffic through. This traffic has to be accounted for once the green wave over the orange roads occurs, so that the green wave is long enough to clear away these cars, and the throughtraffic over the orange roads.

7.1.4 Police, Ambulance, and Fire Department

An additional feature of this system can be to give priority to emergency services like police, ambulance, and the fire department. This can for example be done by altering their identifier, such that it can be identified as an emergency service, once an emergency occurs. Since this system can monitor traffic over a distance of 2 kilometres, it can set its priority first to clearing out any cars that are stuck between crossroads, then clearing out any cars that are already on that way, and finally allowing the emergency service car to pass the series of crossroads with all green lights, without having to brake.

7.1.5 Bikes and Scooters

Once a crossroad is encountered that is also used by bikes and scooters, these have to be accounted for as well. One of the solutions is to give green light to all cycling paths at once, without giving green light to any of the roads for cars. Another solution would be to give the people on the bikes green light together with the cars that go in the same direction, but then it is needed to monitor right-turning cars as well, and then it is not possible for bikes to make a left turn on a crossroad at once (they have to wait twice for a traffic light).

7.2 Traffic jams

7.2.1 Monitoring and Processing

Another application for a real-time monitoring system is to avoid traffic jams from happening. Figure 7 refers to a system that might be used to try to avoid collissions caused by traffic jams on high-speed roads. The system itself can be extended to also give alternative routes to avoid the traffic jam at all, but then more roads should be monitored then just the current road. A change in velocity can be an indication for a traffic jam that is about to happen (the change in velocity is negative) or that is fading (the change in velocity is positive). The figure just mentioned, gives an indication via a colour scheme from green (not dangerous to drive at maximum speed) to red (very dangerous to drive at maximum speed). Collecting units are used again, just like for the traffic lights case, to retrieve information from smartphones from the car driver. Processing units are used once more to process this data to come to a conclusion about the chance at a traffic jam. In the case of this application, the processing and collecting units are one, the units should directly gather and process the information. In the figure, rectangles are these combined processing and collecting units. The circles in between are warning lights, that represent the colour scheme just discussed. There are also warning lights close to the processing/collecting units, which are indicated by the colour of these units, instead of a circle. The warning lights can be attached to markers that are placed along for example a highway (”hectometerpaaltjes” in Dutch, or mile markers in English).

The collecting/processing units again have to cooperate. When a car is almost out of reach of a certain unit, information has to be sent about this car to the next unit. In that way, this new unit can continue to retrieve accurate information about the speed and location of the car. Another reason for cooperation, is that car drivers should be warned in advance for upcoming decreases or increases in speed. This means that previous processing units should know what is going to happen in the next part, covered by the next unit. Therefore a feedback loop of information is needed to guarantee both information forwarding, and returning warning signals to previous units.

7.2.2 Priority Level

If there are cars that have a large deceleration, this might imply that a sudden traffic jam is occurring or at least that cars that come towards that place on the road should be more cautious. This avoids that they have to brake very fast. If several cars send out a high deceleration, this indicates that there should be taken immediate action to avoid more cars from braking very fast, since that causes accidents and on the long term also traffic jams.

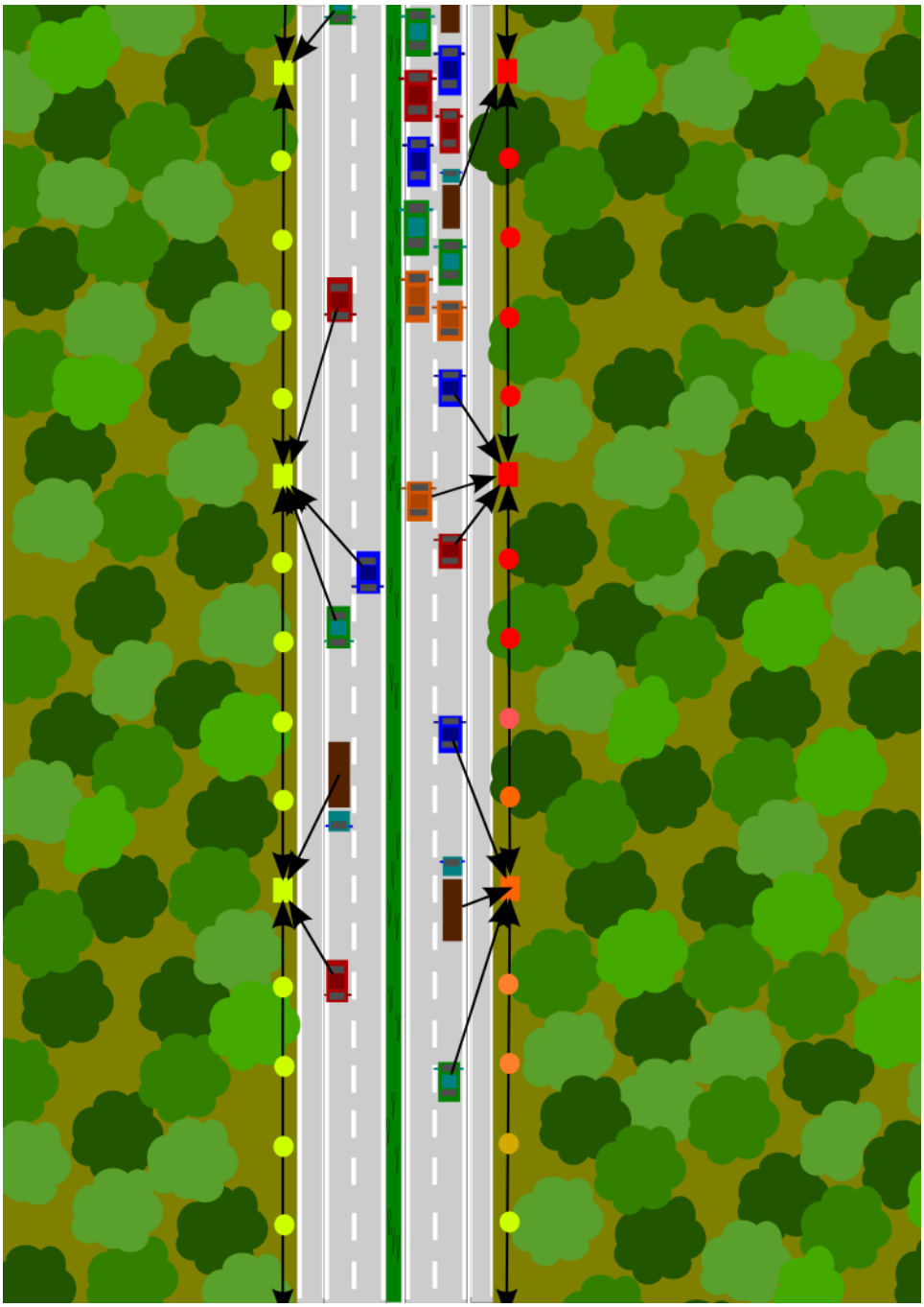

Figure 7: Piece of road that shows an example of the use of a real-time monitoring system. On the left lane there is no reason to decrease speed, on the right lane a traffic jam is happening. Squares are processing/collecting units, and circles are warning lights that indicate how fast you should drive based on the current road conditions (red=very slow, green=maximum speed). The lights at the positions of the units are represented by the colours of the (square) units.

The priority level is created by measuring the acceleration (or actually deceleration) using the internal accelerometer of the smartphone. The outcome is three floats in all three translational coordinates (x, y, and z-direction). These floats give a net acceleration by using the following formula:

$$a_{net} = \sqrt{a_x^2 + a_y^2 + a_z^2}$$

The acceleration is now casted from a float to an integer, which is the final priority level. Casting to an integer makes sure that the priority level is a rounded number, and therefore does not have digits behind the comma. Note that there is no longer a direction to this acceleration. A car driver can have its phone lying on the chair next to him, he can have it in his pocket, he can even have it upside down in his pocket. In that way, there is no real way to tell what the forward acceleration or deceleration is from a car. It is just assumed that if a large deceleration occurs, that this is due to deceleration (or acceleration) in the forward direction of the car. This is also the reason why the priority level is only used for an indication. No changes in the display to the car driver (i.e. the mile markers for example) can be made solely based on the priority level. The indication can however be verified using the GPS data, and if this data also shows that the indication was correct, then the display can be adapted.

8 Prototype

8.1 Assumptions

In this section, a simplified prototype system will be made from the traffic light application (as was shown in section 7.1). Assumptions that are made in this prototype are:

- The prototype will be made around one crossroad, so no links to other crossroads are made. This also means one processing unit, that can only receive data, and not send data.
- Traffic that wants to turn left or right, can do so freely. There are no additional traffic lights for turning.
- Traffic can come from two roads. The first road is hardcoded such that every thirty seconds a car passes over this road (with a maximum of 10 cars). The second road allows traffic to be inserted by using the app on the smartphone, as will be explained in section 8.3.
- The app should still be useful in case that the system is extended with more units.

8.2 Setup

While the basic setup was already sketched in figure 1, it still does not tell anything about the implementation of the prototype of this system. The tablet that the app was tested on was a Nexus 7 Tablet with Android 4.4 (API 19). The app can go as low as tablets and smartphones with Android 3.0 (API 11), due to a lack of support for multiple threading in lower API levels. These threads are needed for asynchronous HTTP clients to prevent that those will be running on the main UI thread of the application and block the entire application when no server is available to send data to. Note that as long as GPS data is acquired, and the application is not shut down, that these threads will be occupied.

In modern Android devices there is a maximum of 1 thread occupied per app. This is introduced since Android 3.0. By using the method executeOnExecutor with the variable "THREAD_POOL_EXECUTOR" it is possible to increase this amount to 5 threads, which was already introduced to versions of Android between Android 1.6 and Android 3.0. One danger of this multiple threading, were Android also warns for, is that the order of execution is not determined up front. For the application this does not matter, since in general only one thread is running when an application can send data to an available server. When there is no server, it is possible to get multiple threads, but they will not send any data, since there is no

server available to send data to, and therefore these threads will just time out after 1.5 minutes. The order of execution is not important in that case.[11]

The collecting unit is implemented with an arduino on which an ethernet shield is stacked. An ethernet shield allows for connecting the arduino to the internet via a UTP cable and network port. Using this combination of an arduino with an ethernet shield gives the option to create a webserver. This webserver can be addressed via its IP-address using a simple HTTP Client (using a URL). In the URL of this HTTP Client to the webserver also data can be sent, as explained in section 6.1.

The processing unit was first a laptop running C-code, but is finally replaced by another arduino due to a lack of time of exploiting a solid connection between the Arduino and the C-code. The idea was that the arduino would receive data via an XBee over the serial port, that was sent via the serial port of the collecting unit (also an arduino). XBees are small radios that can send data via a radio communication between two units. The only thing these units need for this radio communication is a serial port over which this data can be sent. These serial ports can be both hardware and software serial ports. The arduino makes use of its digital pins to send signals to the traffic lights about if they should be on or not.

The full setup is shown in figure 8

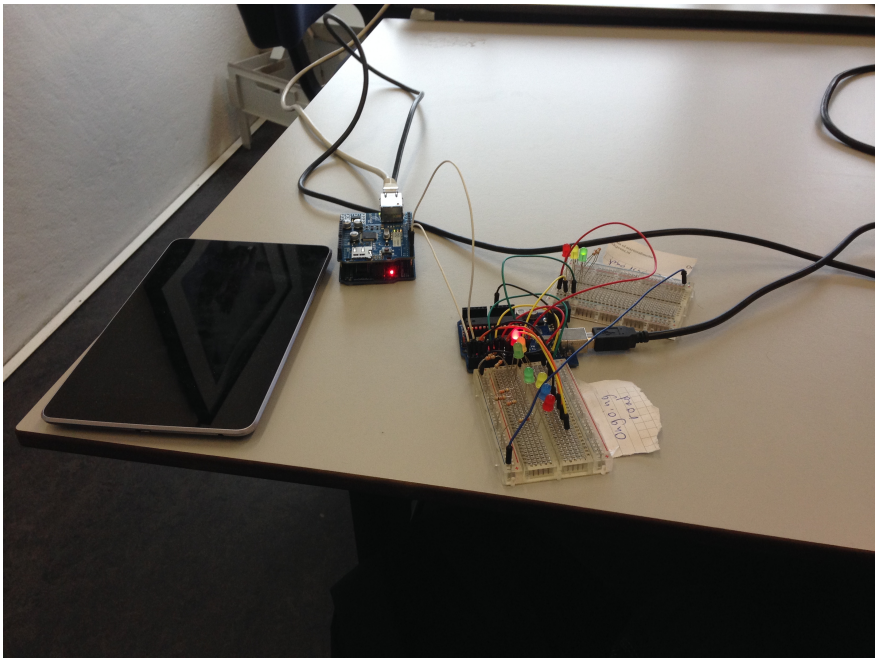

Figure 8: The setup of the prototype: A tablet with application, a collecting unit (arduino + ethernet shield) and a processing unit (arduino with LEDs).

8.3 The App

The app was already discussed in section 5.2. In this section, also a UML diagram was made (see figure 2). This UML diagram was also the basics of the application that was created for this prototype. The application performs the following steps:

- It activates a timer of 15 seconds.
- It listens to the accelerometer sensor, and tries to get a unique identifier based on the 8-digit long noise behind the comma of any measurement with the internal accelerometer.
- The accelerometer is constantly used throughout the timer cycle of 15 seconds, to determine a priority level. This priority level is the maximum acceleration measured during this cycle of 15 seconds.
- A GPS location is requested, this GPS location is saved, and then compared with the previous GPS location. A distance will be calculated between both locations in the unit metres (whereas the locations themselves are given in degrees latitude and degrees longitude).
- The time between two location data measurements is taken by using the SystemClock, and a speed is calculated, by dividing the distance

between the two locations, by the elapsed time between both location measurements.

- The data is placed in a String with the right address (dependent on what the ethernet shield gives out as IP-address, as explained in section 8.4).
- At the end of this address, the data is added by using a specified letter (as explained in section 6.1).
- An HTTP-client is made, which sends the String with the address and data as a URL to the server that will be explained in the coming section (section 8.4).

Note that some of the steps described above are not relevant for the traffic light application, like the calculation of a priority level based on deceleration. However, as was discussed in section 8.1, the app should be made as such, that the system is easily extendible, and therefore it should be able to calculate this priority, even though it might not be needed for the prototype as it is now. This would mean that if someone would come from a traffic light, and go onto the highway, the same data send to the collecting units should suffice to give enough information to the processing units along the highways as well.

The prototype is able to do these measurements of location, velocity, and priority level, but for the prototype that is described in this chapter these features are not needed. Since this prototype is used for demonstration purposes, a timer is included that counts down from 200 seconds to 0 seconds. Instead of sending all data about location, velocity and priority level, the prototype for now only sends out the unique identifier of the tablet, and the time it takes till it reaches the traffic light (which is this timer counting down from 200 seconds). The other data however is still calculated within the app, and can be shown on the display of the tablet if needed, or it can be shown during execution via a the serial port of a computer (using System.out.println for example). Note that, to simulate reality a little more, the URL will not be created until a new location is acquired. This means that, even though the location is not sent, still GPS is used to acquire a location, and then send the data (identifier, and time) away. This simulation therefore includes the delay for acquiring GPS data, which is a delay that will be there in the real-life application as well.

8.4 The Collecting Unit

The collecting unit is a simple combination of an arduino with an ethernet shield stacked on top of it. This is shown in figure 9. Using DHCP, an IP-address is generated dynamically, this IP-address can be read serially, and

used within the application. In real-life applications, one has to use static IP-addresses, or dynamic DNS. These concepts were explained in section 6.1.

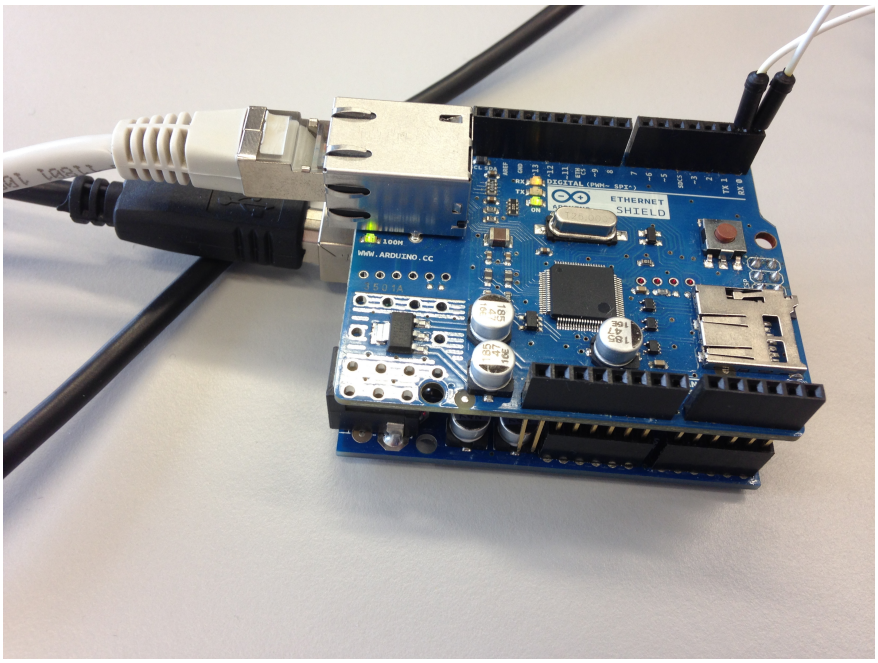

Figure 9: The collecting unit of the prototype: An arduino with ethernetshield stacked on top.

The collecting unit splits up the URL it gets from the application of the tablet into pieces that are relevant. The first part is its own address, which is not useful for processing. Afterwards, it selects per letter in the URL what number it should contain. For example, the identifier of a unique car driver, is represented in the URL given in section 6.1 as an "i". The processing unit knows that the 8 digits behind the "="-sign are the numbers that it needs for the ID. It casts these numbers to integers one by one, and stores them in the corresponding variable, in this case called "ID".

The data transfer between the collecting unit and processing unit originally was done by XBees, unfortunately one broke during the project, so no transmission per XBee was possible anymore. Instead, the communication between both units was done via a wire over the serial ports of both arduinos. XBees though also use this serial connection between both Arduinos. The only difference is that XBees can send this data wireless, whereass the current connection is wired.

8.5 The Processing Unit

The processing unit contains the algorithm to determine what road gets a green light, and what road gets a red light. The processing unit of the prototype is shown in figure 10. It also determines for the road that can be controlled with the tablet, how fast the car should drive when it is a distance of 200 metres from the traffic light, and still wants to make it through the green light. Here is for short how it works:

- It determines who should get green (either the cross road or straight road) by counting the amount of cars that are reaching the traffic light within 30 seconds. The road with the largest amount of cars gets green preference.
- If the amount of cars is the same within these 30 seconds, then the average time for all these cars till they reach the traffic light is calculated for both roads. The road with the lowest average time till it reaches the traffic light is given green preference.
- If still no conclusion can be drawn, since the amount of cars, as well as the average time till the traffic light is the same, the time sample is increased by 10 seconds, and the first two steps are repeated
- If there is after a time sample of 50 seconds still no preference found, then the crossing road is given preference.
- Now that the road with green preference is known, it has to be checked if that traffic light is already green. If so, nothing happens, if not, the currently green traffic light has to go via orange to red, and the new green preference can be applied.
- It will take exactly $\frac{1}{3}$ of the sample time before the algorithm will check again to determine the next green preference.
- On the other hand, it should be known if the traffic light stays green for the next period. So a look in the future has to be done. This will be done by simply applying the same algorithm once more, but this time with a delay of the sample time. If the green preference in the future changes from straight road (which is controlled by the app) to cross road, then the time till this change is known, and a speed over the distance of 200 metres can be simply calculated and shown to the car driver at that distance away from the traffic light. This process is repeated as long as $\frac{1}{3}$ of the time sample has not been elapsed.
- In the meantime, also during this waiting period, new data is gathered from the serial port for changes in time till it reaches the traffic light. These values are stored, sorted by identifier, in an array. If an identifier

is already known, only the value of the updated time till it reaches the traffic light is inserted.

- Note: There is an internal timer in this algorithm that makes sure to keep track of the fact that the times in the array are processed. It will count up, and if the time in the array minus the time currently on this timer is 0 or lower, the car has passed the traffic light. This car is now deleted from the array that stores all times ordered by their identifier. Since this timer has an offset (since the timer is no longer 0) once a time is updated in the array or newly inserted in this array, this offset has to be added to this newly gained time, such that upon subtraction you really see the time that the car still needs to reach the traffic light.

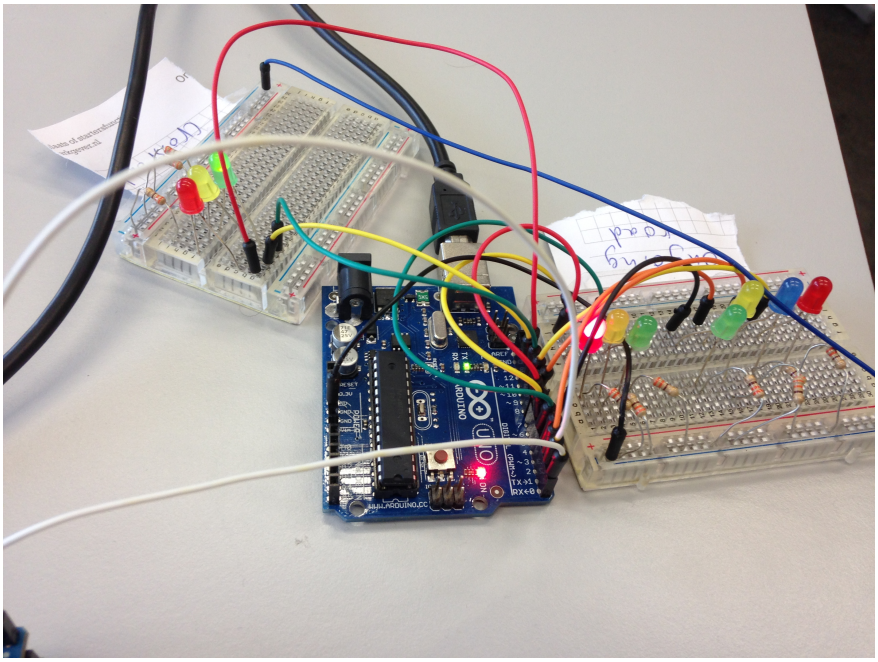

Figure 10: The processing unit of the prototype: An arduino with several LEDs indicating the traffic lights and a speed indication at a distance of 200 metres away from the traffic light (in binary by four LEDs on the breadboard at the right of the photo, indicating the digits of a speed, while neglecting the last digit, and rounding up the result, so 93 $\frac{km}{h}$ is rounded up 100 $\frac{km}{h}$ and that would give binary 10 on the display).

8.6 Measurements

In an application that is claimed to be real-time, it is important to see what the delays are in the prototype. Note that these times are measured via a

serial connection using "Dalvik Debug Monitor Server (DDMS)", which is the default monitor in the IDE "Android Studio". This means that, due to the use of a serial connection, additional delays are introduced that are not accounted for. First of all, just starting up the application takes on average about 215 milli seconds. Since a timer is used, a run() method has to be used that is executed once the timer starts to run again. This timer, called MyTimer (as explained in section 5.2), executes this method run() once every 15 seconds. From the moment that the method is called, till the moment that a URL is sent to the HTPP Client this takes on average 16.6 seconds. After creating this URL, it takes on average still 60 milli seconds to send this data to the collecting unit. This short period of time is due to the fact that the HTTP Client is completely handled by another thread then the UI thread, which runs asynchronously. If no server is available to answer the request, the threads keeps on trying for 1.5 minutes, and afterwards throw an exception about the lack of an HTTP server. This was tested by running the application without running the webserver, so that the application tried to make connection to a server that did not exist. The timer builds up an error on average of 1.28 seconds over its full range of 200 seconds. In between two time samples this error is on average 256 milli seconds. Since the timer is not used in real-life applications, also tests were performed by simply watching the speed changes. When leaving the tablet resting on the table, there was an average velocity of $0.4\frac{m}{s}$ without moving the tablet. The maximum velocity measured was about $1\frac{m}{s}$ while leaving the tablet on the table without moving it.

To evaluate the efficiency of the system, it is possible to look at the amount of cars that can directly pass, since they have green light once they reach the traffic light. All raw data that the rest of this section is based on, is shown in appendix A. In figure 11 a measure for this is shown, that is called "Hit rate" in this report. A time till the car reaches the traffic light is sent to the server, and if the car has green light once it reaches the traffic light, it is a hit. The average time till the traffic light actually gets green versus the time that was requested till the car reaches the traffic light is given in figure 12. Also note the (sampled) standard deviation of the mean that is indicated in this figure by the bars. The average green light that is given to a car versus the time that is given to the server till the car reaches the traffic light is given in figure 13. Again, the (sampled) standard deviation of the mean is given by the bars in this figure.

Finally, it is useful to test what happens when a car first sends data that it is 60 seconds away from the traffic light, and then a little over or under halfway send that he is still 30 seconds underway until he reaches the traffic light, such that a small error was introduced in the 60 seconds given at the start, and a new calculation has to be done to account for this small error.

The system is able to overcome these small deviations, since still a hit rate of 100% was reached. One thing that was different though was the time till a green light was given. The green light, on average, came about 9 seconds later. The period that a car got green light also decreased by, on average, about 7 seconds.

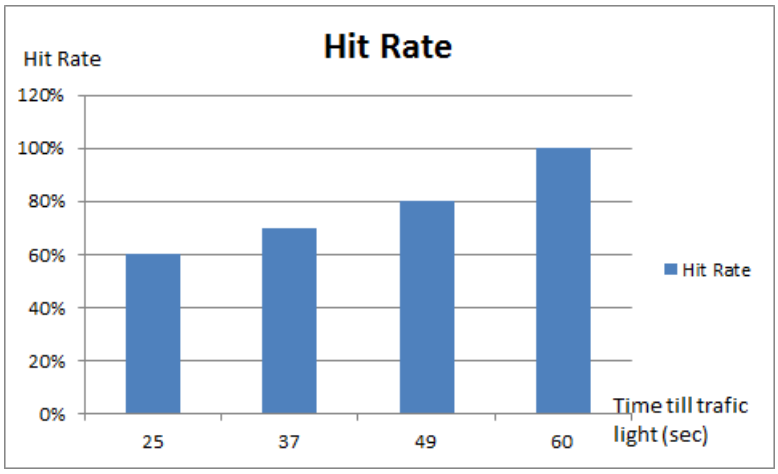

Figure 11: The amount of cars that directly gets green light once they reach the traffic light, versus the time that was given to the server about how long it still takes to the traffic light.

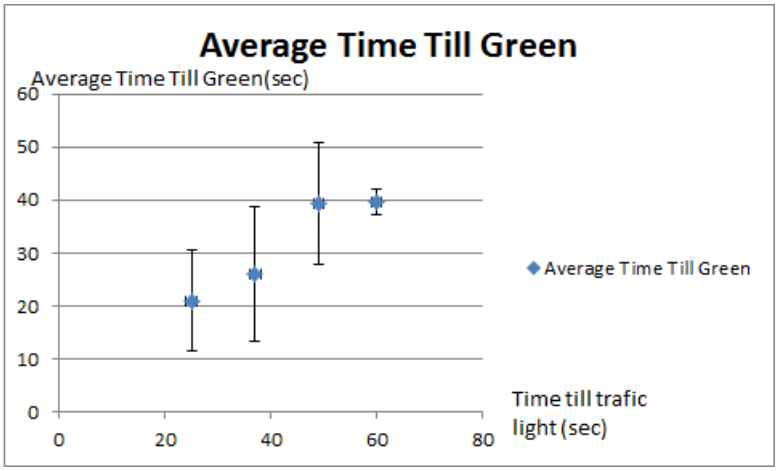

Figure 12: The time till green light is given versus the time that is given to the server till the car reaches the traffic light. The bars indicate the sampled standard deviation of the mean.

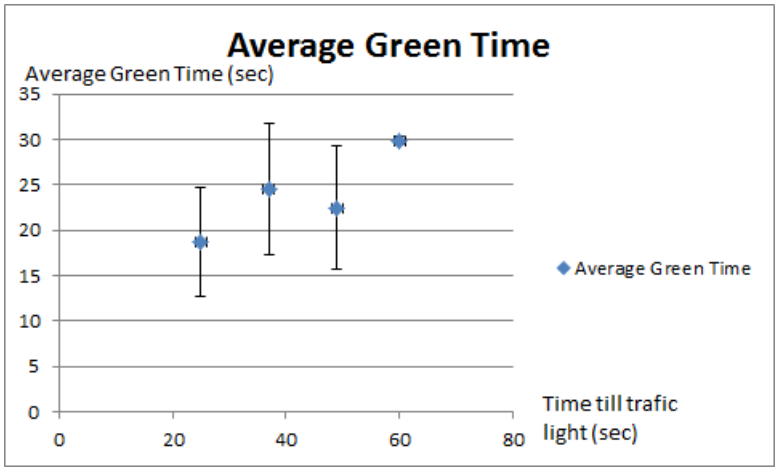

Figure 13: The time that the traffic light is green versus the time that is given to the server till the car reaches the traffic light. The bars indicate the sampled standard deviation of the mean.

9 Discussion

During the report, the prototype was made that consists of three main parts: The app for an Android tablet, the collecting unit, and the processing unit. In all three parts, there were simplifications made that might have been solved in a more neat way. This chapter discusses what could have been done better if I would do this assignment again.

9.1 The app

The app itself uses a lot of asynchronous tasks. While it looks neat to have seperate classes for all timers, accelerometers, location listeners, and HTTP clients, it brings with it several disadvantages, since you cannot inherit methods from the parent class of the main activity, in which all system services of a regular Android application are stored. At the other hand, the advantage of using asynchronous threading is that you have 5 threads at your proposal, instead of the regular UI thread that is used by the main activity. This can speed up the application significantly, especially when no server is available, and the HTTPClient draws the attention of the thread for 1.5 minutes, such that the application cannot do anything else in the meantime. These threads are however, as just mentioned, only used by the HTTP Client, all other mentioned asynchronous classes still have to be operated on the UI thread, but now cannot directly do so. They have to ask for the system services from the application that are stored in the MainActivity class.

Another point about the app was that it had to sent to the collecting unit a URL with the velocity, priority level, identifier, latitude, and longitude. In practice, I only sent the identifier, and a time it takes for the car to reach the traffic light. The reason for this was that for the demonstration purpose of the prototype, it was not possible to do a decent simulation of a car approaching a traffic light when there was almost no change in location. A possible solution could have been to fake the position data, but I chose to leave the app as it is (so that it is unaltered when it is needed), and implement a timer in the Android application that counts down from 200 seconds to simulate a car approaching the traffic light.

9.2 The collecting unit

The collecting unit in the prototype uses DHCP to get an IP-address for the webserver. In a real-life application the IP-address could be replaced by a system that uses DDNS to update the IP-address behind a reference name,

or to give a downloaded list with all new IP-addresses when someone enters a road. This could have been implemented during the bachelor assignment. However a static IP-address was tried, but did not seem to work on the University network as well as using DHCP. For some reason the connection did not always work, whereas just using a dynamically assigned IP-address always worked.

Another thing is that the collecting unit does not contain a wireless access point at this moment. Since the tablet that was used could connect to the wireless network that is available on the University of Twente, there was no urgent need for a wireless access point in the collecting unit. The only thing why a wireless access point could be handy, was for doing measurements with the system based on the connectivity range, and the time it takes to connect to the network, and send data. In that way it could have been evaluated if a car has enough time on a highway application to connect and send data to the collecting units on the road, when the car only has 30 seconds to do so.

9.3 The processing unit

The processing unit was from the start on never intended to be an arduino. In the prototype an arduino was used after several other options were tried. The processing unit was intended to be a stand-alone device running on C-code (just a laptop, or maybe a raspberry pi) that would neatly allocate and deallocate memory spots for data that would come in. Some threading would be used to maximise data processing efficiency. This code was actually written, and it worked on a laptop, however the problem was in combining the C-code with the collecting unit (in specific the data communication of the laptop with the arduino itself). Several options were tried, among which reading out the HTML code from the webserver page, and using a serial connection between the Arduino and C-code. Both seemed promising at first, but were just to difficult to implement in the short time slot of 10 weeks, without getting behind on schedule too much. An arduino was used, since communication between an arduino and another arduino is easier, and XBees could be used. These were later on replaced by a jumper, since one of the XBees started to malfunction.

In the processing unit, the algorithm for determining what traffic light gets green preference gives always preference to the crossing road when no cars are simulated. While for the demonstration this is pretty handy, so that the viewers can indeed see that the system sends data to this unit, and processes it so that the traffic light changes to green preference for the straight road (the one that is controlled by the tablet), it should not be used

in real-life since then starvation would occur.

9.4 Results

A lot of delay in the application is caused by acquiring a URL. For this URL, GPS is needed. The GPS service is often the one that causes this long time delay. Unfortunately, there is no real way to speed up this process. Taking a GPS location just needs this amount of time. The choice for an asynchronous thread for the HTTP client seems to not very much pay off once the application is working (the process of letting the collecting unit collect data via the asynchronous thread only takes 60 milli seconds). The danger however is that there is no server available at some point. By using asynchronous threads, the application is able to still function and try to look for new servers, since the threads are handled in the background and expire after 1.5 minutes. If these HTTP clients would be executed in the main UI thread, this would mean that nothing could be done for 1.5 minutes, while waiting for the HTTP client to finish. The timer causes a delay of 1.28 seconds for a full cycle of 200 seconds. It seems like a lot, but if this system is used in practice, then this data is not coming from a timer, but rather from changing locations on the tablet, so this error will not be there. A maximum error in velocity of $1\frac{m}{s}$ would be $3.6\frac{km}{h}$, which will almost not be noticed on roads where the speed is at least 5 times as high. Besides, this is the maximum possible error, the average error of $0.4\frac{m}{s}$ would give $1.44\frac{km}{h}$, which is even more neglegible, and might even be subtracted from the velocity on default as an offset for the sensor. This value though changes per device, and should then be calibrated per device before using this offset.

The results from the response to the server gave a clear conclusion: The longer the time till the car reaches the traffic light, the larger the hit rate. This is also what is exptected, since the algorithm has more time to make a scheme, since it has this information earlier. It should be noted that the hit rate of 100% is quite high, in practice this will never be reached, since it is possible that two cars come at the traffic light at the same time, and then one has to wait. Also note that in reality there will not be a constant stream of a car once every 30 seconds, that was hardcoded into the prototype.

Another thing to note, is that the average time till green light seems to increase at first, but seems to converge to about 40 seconds for the larger values. It should be noted that there are cars that are hardcoded every 30 seconds over the crossing road of the intersection. When a server request is made for a car on the ongoing road of the intersection, and the time till it reaches the traffic light is 30 seconds, then there is (almost) always a car of the crossing road that is getting first at the intersection. If this car is at the intersection at that moment, and the processing unit is recalculating

currently the green light scheme, then this takes 0 seconds. If the car on the crossing road has just started at 30 seconds away from the traffic light, and it is a little bit earlier then the car that made the request to the server, then it can take up to 40 seconds before the car that made the request to the server can pass. This means that the average time for a car that makes a request at 30 seconds from the traffic light away is about 20 seconds, the real value was on average 20.996 seconds for 25 second requests, which is quite close. The difference can be explained by the fact that only 10 samples were used.

If a time request is made to the server for more than 30 seconds till the car reaches the traffic light, then there is at least one (hardcoded) car that passes the intersection before this car will be processed. It was just discussed that this car could take in between 0 and 40 seconds of time, before another car could be seen by the green light scheme. If the request was made for for example 37 seconds, then at least 7 seconds have to be waited before the green light can turn green, and at max 47 seconds. This is also proven by the results obtained, since none of the results were above 47 seconds or below 7 seconds for a requested time of 37 seconds.

For a time sample of 60 seconds, this means that at least twice a car passes. The second car can pass at 30 seconds, which means that the green light will be given at 30 seconds to the road that contains the car that made the request. It is also possible that this happens at 70 seconds at maximum, which would mean that the first hardcoded car would pass 30 seconds after the request, and the second one 60 seconds after the request, and a time sample of 10 seconds before recalculating can let this go up to 70 seconds before the car that made the request can pass the intersection. The average waiting time would therefore be around 45 seconds for a 60 seconds request. The real average is 39.653. The diffeence can also be caused by the little amount of samples, namely 10 samples again.

The slight increase in average waiting time for green light in the case that two requests were made (one of 60 seconds, and then a little over or under halfway one of 30 seconds) was caused by the fact that in some of the cases this meant that another car (on the hardcoded crossing road) was just a little faster, and caused the traffic light to stay red a little longer.

10 Conclusion

At the start of this report, a research question was formulated. This research question was: "Is it possible to design a public system that can maximise throughput of cars in traffic situations, based on local, real-time, flexible monitoring using smartphones?". In this research question the main focus lies on throughput of cars in traffic situations, by using a public display. This should be done using a system that is at the same time locally controllable, real-time, and flexible. The following three paragraphs go over these three properties of the system and how they can increase the throughput of cars in traffic situations.

Controllability on a local scale was achieved by directly making a connection between the car driver his tablet or smartphone, and the collecting units. Collecting units were used to gather data from all car drivers, and these collecting units sent the data to a processing unit that could change the public display (a traffic light, warning lights next to mile markers on a highway, and so on). In this way, local data was used to increase the throughput of cars in traffic.

There was a delay between acquiring data on the tablet, and processing it in the collecting and processing units. While this means that the system is not perfectly real-time, it can still monitor approximately real-time with a slight delay. If enough cars are available that send out data, then this delay will eventually fade away, since there is plenty of data available to process. The almost real-time approach though does increase throughput, since decisions can be made (almost) real-time about what would be the best to allow as much traffic to continue their car trip as possible.

Finally, monitoring individual car drivers is already a great advantage over currently available, public display systems, but this system can be used for several applications at the same time. Connecting a highway (or other road with possibility of a traffic jam) with a crossroad with traffic lights can easily be done via adding additional collecting units to connect the processing units of both situations. Also, emergency services do need to be fast, and with a system like this they can be noticed already 2 kilometres up front, and the traffic can be cleared before they arrive at that point in traffic. All these flexible options also increase throughput once more, since they allow for a central processing scheme (while still monitoring locally) that can be optimised for throughput.

11 Recommendations

For any further development of this system, my recommendations would be:

- Work out the prototype to work for more situations. For example, add more crossroads with traffic lights, or try to establish a connection between a highway, and a crossroad with traffic lights. In that way, the insights in the system will be increased.
- Try out the app on several devices, and make sure that the prototype gets access points mounted on the collecting units. In that way it is possible to evaluate the feasability of such a system in real-life. Also do some measurements about the maximum distance covered per unit, and the speed of data transfer at the outer edges of the range of the access point to see if the system still works.
- Try to develop a system that can be used on all roads, and that can give alternative routes via public displays when the system indicates that a traffic jam is about to happen on that road.
- Try to find ways to increase the amount of car drivers that can be monitored. Also keep in mind that duplicate measurements are taken away (for example two car drivers in one car that have the app on their smartphone).
- Research should still be done to if the display is understandable to car drivers. Can in one glance the purpose of for example the speed indicator at crossroads with traffic lights be clear?
- Another research topic is to see if the newly implemented system described in this report is more efficient than the currently implemented system. Extensive simulation and modelling with traffic simulators can be done to see which of both systems is more efficient. These simulations can also be used to optimise the system described in this report (for example: how much margin should be used to make sure a car passes the green light in time?).
- A comparison can be made between the system described in this report, and a similar system based on opportunistic sensing. This comparison should then lead to a conclusion about which system is better and more efficient to use in the traffic application described in this report.

12 References

References

[1] *Sensors Overview*. developer.android.com.

[2] *Google Play Services*. developer.android.com, May 2014.

[3] *TomTom expands Availability of Live Road Condition Information with HD Flow and HD Route Times Products*. TomTom, 7 april 2010.

[4] Nicholas D. Lane, Shane B. Eisenman, Mirco Musolesi, Emiliano Miluzzo, Andrew T. Campbell *Urban Sensing Systems: Opportunistic or Participatory?*

[5] Hans Scholten, Pascal Bakker *Opportunistic Sensing in Train Safety Systems*. International Journal on Advanced in Networks and Services, Volume 4 Issue 3 and 4, 2011.

[6] *How are Vehicles Detected at Traffic Signals?* Traffic Signal Design (Sanderson Associates).

[7] Ramon Llamas, Ryan Reith, Kathy Nagamine *Smartphone OS Market Share, Q1 2014*. International Data Corporation (IDC), 2014.

[8] *Voor het eerst meer mensen met smartphone dan pc*. Volkskrant, 13 december 2013.

[9] Anna Salleh, *Traffic lights make their own green wave*. ABC Science, 2 november 2006.

[10] Michigan Department of Transportation, *MDOT explains flashing yellow left-turn signal*. Youtube, 20 october 2010.

[11] *AsyncTask*. developer.android.com.

A Data samples

In this appendix, the raw data samples from the results section of the prototype chapter (chapter 8 are placed for reference.

25 seconds:			
Sample number:	Time till green light:	Duration of green light:	Hit (1) or miss (0)
#1	28,64	15,91	0
#2	24,53	16,15	1
#3	18,55	14,93	1
#4	28,63	15,92	0
#5	15,63	15,96	1
#6	21,53	15,98	1
#7	25,51	16,09	0
#8	8,46	30,08	1
#9	34,6	16,06	0
#10	3,88	29,79	1
37 seconds:			
Sample number:	Time till green light:	Duration of green light:	Hit (1) or miss (0)
#1	19,56	30,08	1
#2	14,46	30,13	1
#3	45,66	15,98	0
#4	15,56	30,11	1
#5	24,2	29,68	1
#6	16,46	30,31	1
#7	40,6	16,36	0
#8	44,75	15,91	0
#9	25,48	16,16	1
#10	13,53	30,11	1

Figure 14: Data samples for 25 seconds and 37 seconds left until the traffic light is reached.

49 seconds:			
Sample number:	Time till green light:	Duration of green light:	Hit (1) or miss (0)
#1	29,6	30,13	1
#2	42,56	16,06	1
#3	30,55	30,75	1
#4	59,71	15,94	0
#5	55,86	17,78	0
#6	44,55	16,04	1
#7	30,81	22,46	1
#8	30,18	29,54	1
#9	41,49	16,1	1
#10	28,58	29,99	1
60 seconds:			
Sample number:	Time till green light:	Duration of green light:	Hit (1) or miss (0)
#1	39,21	28,63	1
#2	38,55	30,11	1
#3	42,56	30,14	1
#4	37,58	30,16	1
#5	39	29,68	1
#6	36,81	29,83	1
#7	40,98	29,78	1
#8	36,68	29,96	1
#9	41,53	30,15	1
#10	43,63	30,18	1

Figure 15: Data samples for 49 seconds and 60 seconds left until the traffic light is reached.

60 seconds (with additional +/- 30 seconds request):			
Sample number:	Time till green light:	Duration of green light:	Hit (1) or miss (0)
#1	37,62	29,31	1
#2	45,54	30,2	1
#3	54,63	15,18	1
#4	59,11	15,53	1
#5	57,8	16,1	1
#6	34,01	30,2	1
#7	42,25	29,91	1
#8	53,56	16,24	1
#9	54,25	15,96	1
#10	45,66	29,18	1

Figure 16: Data samples for 60 seconds left (including additional about 30 seconds step) until the traffic light is reached.