TECHNICAL UNIVERSITY OF DENMARK

BACHELOR THESIS

Smart City Traffic Management

Author:
Rasmus Alkestrup Eskesen (s133997) ,
Jacob Thinglev Grundahl (s133994)

Supervisor:
Christian D. Jensen

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Engineering (IT)

in the

-

Department of Applied Mathematics and Computer Science

23. January - 2017

Abstract

Rasmus Alkestrup Eskesen (s133997) , Jacob Thinglev Grundahl (s133994)

Smart City Traffic Management

The concept of smart cities involves a city or urban development, where multiple Information and Communication Technology (ICT) and Internet of Things (IoT) solutions are integrated in a secure way to manage the city's assets. We want to focus on transportation systems in this regard - more specifically, traffic management with special focus on traffic signals and the control thereof.

The ultimate goal of smart cities, is to improve quality of life using urban informatics and technology to improve the efficiency of the relevant services, while still satisfying the needs of the residents. We aim to benefit a given city (thus ultimately improving the quality of life) by providing a more efficient traffic flow, reducing overall pollution and other general environmental impacts, improving the utilization of the city's budget, and automating incident detection allowing for quicker responses from a relevant authority. All the while, citizens will ultimately benefit in the form of improved road safety, reduced congestion (and fuel costs), and generally a better driving and commuting experience.

We want to introduce a software-based solution that simulates a relevant environment (i.e. one or more traffic signals) and implements the necessary features to achieve the abovementioned visions. More specifically, the idea is to control the traffic signals in a more dynamic way than the result of using fixed timers and/or simple sensors that provide the green waves today. We want to simulate one or more traffic signals as a local system or a network of local systems respectively. This would require a general model of a traffic signal, where the actual signals are controlled locally based on a local policy and by using inputs from various sensors, and this policy would then be able to communicate with other parts of the network by sending relevant information.

Contents

1 Introduction

To give a proper introduction to this project, we are first going to answer the following questions:

- Why are we doing this project?
- What are we trying to accomplish?
- How are we going to accomplish it?

The project motivation is going to seek to answer *why* we are doing this, where the project description attempts to answer *what* we are actually doing, and finally the project scope should give an idea of *how* this is going to be done.

1.1 Project Motivation

When looking at urban developments today, the words traffic and congestion are often used in immediate succession. The invention of the car was a major game-changer when talking passenger transportation. During the 20th century, it quickly changed from being a toy for the rich to a standard tool for the everyday-citizen in most developed countries. When we think about the amount of cars today, and the fact that most of these cars drive around in the same small parts of the world, it comes as no surprise that a proper infrastructure is required in order to account for the inevitable congestions and general disturbances in the flow of traffic.

If you have ever stopped at a red light on a completely empty road, you probably know that it can be quite frustrating, since you might have to wait patiently until the light turns green for your lane, even though there are no other cars around, but depending on what type of vehicle you are driving as well as the kind of traffic light you are waiting at, it might not actually know you are there. Some traffic lights have fixed timers, whereas others use sensors to pinpoint where you are.

While many traffic lights still use the fixed timers to control the flow of traffic, methods such as the so-called green wave has proven to be somewhat successful. The green wave is essentially created on a busy road with multiple consecutive traffic lights, by turning the lights green one after another in a synchronized fashion. The idea behind this method, is that a driver should then be able to drive fairly continuously along the road, without the need to stop all the time. Another method involves reacting to different preset situations. An example of these situations could be: rush hour, normal traffic and night time. In this case the traffic light would be instructed to alter its policy

depending on the current situation; during rush hour, two of the lanes might tend to be especially congested compared to the rest, while night time might only require a short burst of green in each direction.

Traffic lights using sensors do however seem to have the greatest impact on the improvement of the traffic flow, as it makes the system reactive and much more dynamic. Some typical kinds of sensors might include cameras/motions sensors and underground electronics such as induction loops or even weight sensors. Other types of hardware (other than actual sensors) working as an interface could also include buttons for pedestrian crossings for instance.

Sensors are great for letting the system know when an object is approaching, where it is approaching from, and maybe even where it is bound to go, depending on the types of sensors, as well as the structure and layout of the traffic light. With all this information available to the system, the control unit can make a better decision than the pretty much random guessing of a fixed timer.

Regardless of the chosen current technologies and methods, there is still much to be done in the area, to improve the overall traffic flow in urban developments and ultimately improve the quality of life for the road users.

1.2 Project Description

Our vision involves improving the current traffic control systems, by utilising modern technology to allow for communication between control units in a network. We aim to prove that the efficiency of the current systems can be improved significantly, by establishing forms of communication in a traffic network, to share traffic information between nodes in the system.

By sharing traffic information between multiple traffic lights in a traffic network, each individual traffic light will know about all (applicable) incoming traffic, thus ultimately make a more informed decision upon deciding the most ideal light-configuration for a particular situation.

More specifically we want to design and implement a traffic simulator that we can use to test different traffic control algorithms, based on different types of information inputs. We ultimately want to compare an algorithm utilising information from neighbouring traffic lights with commonly used algorithms that either use fixed-timers or local sensory inputs.

1.3 Project Scope

In order to satisfy our goals for this project, we need to prove that traffic lights that implements an algorithm that utilises information shared between multiple independent traffic lights, shows significant improvements to the overall traffic flow. By showing a decrease in average wait time for all cars traversing one or more traffic lights, in a number of different scenarios, we

will be able to conclude that extended knowledge about traffic flows in the network for each traffic light correlates to an improved traffic flow through the traffic lights and the network as a whole. An overall goal would be to improve the quality of life for the users of a traffic network, by providing a more efficient traffic flow. An improvement to the traffic flow could ultimately result in shorter travel times through the network, less pollution, quicker response times for emergency services, higher road safety and more value for the money for the users (less fuel cost).

1.4 Report Outline

Now that we have given a short introduction to the project at hand, let us establish a quick outline of what the rest of this report will bring.

After this **introduction** to the project, we are going to have a look at the **state of the art**; the current relevant technologies and what has already been done in the area in which we are going to be working.

After establishing what we want to do, and what has already been done, we will introduce an **analysis**, where we begin to analyse the problem in order to determine a suitable solution to the problem. As part of the analysis, we first describe the overall system and then put the system into perspective by looking at it as a user. This part includes a few use cases that aim to ensure an understanding of the systems we have to work with, as well as how the new system will affect the users. Still as part of the analysis, we then analyse the product functions by breaking the project into three primary parts to separate the theory; the traffic simulation model, the traffic control algorithm and general communication. As the last part of the analysis, we then establish some specific requirements for the system, to ensure we have a complete problem before we start to actually design the system.

After analysing the entire system, we then proceed to **design and implementation**, where we describe how the system has been developed. This section first outlines how the system is structured, and then proceeds into more technical details about how it is build.

This is then followed up by an **evaluation**, where the system is first tested using a few different scenarios. After the tests, the results of them are then discussed relative to the scope of the project. Lastly a few ideas for future development is outlined to give an idea of where this project could be headed in the future.

As a roundup of the report, the **conclusion** will then summarise the work that has been done, and conclude on whether the project as a whole was successful or not and to what extend. At the very end of the report the **appendix** is located, where diagrams, calculations and test data is stored for reference throughout the report.

2 State of the Art

Before analysing the problem further, let us now establish the recent stage in the development of what we aim to do; what are the newest ideas and features regarding smart city traffic management. The purpose of this is to outline the base of operation; the foundation upon which we are going to build our model.

2.1 Traffic simulators

The complexity of traffic stream behaviour and difficulties involved in performing experiments with real world traffic has made the use of simulators important when it comes to analysing and testing out new ideas involving traffic management. With the use of simulators and different traffic simulation models one can simulate real world traffic problems and situations in great detail. In a broad sense, traffic simulation models can be categorized into 2 types: continuous and discrete, depending on how the elements describing a system change state. Discrete can then again be classified into time and event based models. Discrete time models divides time into a fixed small interval and then within each interval computes changes within the selected system elements. Whereas discrete event based models computes based on abrupt changes in the state of the system (events). Discrete event based models use less computational power compared to discrete time based models, however discret time based models are more realistic and detailed.

Depending on the level of detail, traffic simulators are classified into 3 different categories: macroscopic, mesoscopic, and microscopic. The least detailed: macroscopics view the traffic flow as a whole. While microscopic models give attention to individual vehicles and their interactions. Mesoscopic is in the middle between the two. Simulators thus enable the evaluation of infrastructure changes and traffic policy before they are implemented in the real world, thus enabling it so the changes can be optimized before they are implemented on the road, and through this hopefully avoid the implementation of bad traffic policies.

Although the ideal goal would be to implement our model onto a real-life network of traffic lights, simulators are a great medium for testing the model before actually playing around with real human lives. A ton of open source traffic simulators already exists, but we are now going to focus on three of the ones that brought the most interest to us. The three simulators that we have primarily looked at are: SUMO (Simulation of urban Mobility), Movsim, and OTSim (OpenTrafficSim).

SUMO is an open source traffic simulation suite developed in 2001. It is implemented in C++ and has a lot of supporting features and provides various APIs to remotely control the simulation.

Movsim is an open source, microscopic vehicular traffic simulator developed in java.

OTSim is an open source software initiative to support research and development of multiscale and multi modal traffic models. It is also implemented in java and provides free to use knowledge and utilities.

2.2 Vehicular networks

Mobile ad hoc networks (also known as MANETs) are wireless networks created spontaneously with the primary intent of data exchange. Vehicular ad hoc networks (henceforth referred to as VANETs) are created by applying the principles of MANETs to the domain of vehicles. While they share a great portion of their high level design and concerns of interest, the details differ vastly between the two. Whereas MANETs are build upon the idea that the entities move around in a relatively random fashion, VANETs are focused on vehicles, which tend to move in an organised fashion, as the vehicles are usually restricted to a certain path (i.e. a road) when traversing through an area where a need for these technologies are present. This means that communication and other general interaction between stationary entities, such as road-side units (RSUs) become fairly accurate and predictable. Contrary to common belief, VANETs - or MANETs for that matter, are not synonymous with inter-vehicle communication (IVC), which is a much more generic field, focussing less on spontaneous networking, and more on the use of infrastructure, such as cellular networks or even RSUs.

VANETs can technically make use of any wireless networking technology as their basis, however some are more prominent than others. The most used technology used in VANETs is short range radio technologies such as WLAN. A common implementation of this would involve standard Wi-Fi and then some sort of high-level communication protocol (such as ZigBee) in order to establish a wireless personal area network (WPAN) with small, low-power digital radios. This is a very common design architecture in regards to VANETs as it is both cheap, small, reliable, fast and very supportable (scalable, maintainable etc.). Other possible wireless networking technologies to be used for VANETs include cellular or even LTE (telecommunication) networks.

VANETs support a wide range of applications. An example of these applications is traffic information systems. These use VANET communication to provide real-time traffic reports to the vehicle (e.g. via vehicle's satellite navigation system). If you have a navigation system in your car or even your phone, chances are it is using VANET communication to update you in case of traffic congestion or the like. Another example application is electronic brake lights. This is a way to inform a driver that other vehicles are breaking, even though the actual view of the physical brake lights might be obstructed.

Lastly VANETs can be utilized in the field platooning. This is when a continuous line of cars are electronically coupled like a train, and it allows for the leading car to control the speed and direction, whereas the rest of the train will just follow autonomously.

Intelligent vehicular ad hoc networks (InVANETs) are VANETs that use Wi-Fi (WAVE standard) and WiMAX for easy and effective communication between vehicles with dynamic mobility. This is primarily used for tracking vehicles and for media communication between different vehicles. By using a Wi-Fi based navigation system, the localisation of vehicles become faster and supports localisation in cramped or even underground areas (e.g. cities with tall buildings or tunnels respectively). InVANET can be used as part of automotive electronics, which has to identify an optimal path for navigation with minimal traffic intensity. Thus InVANETs serves as a modernised alternative to standard VANETs in regards to vehicular navigation.

Regardless of the network type, the aforementioned network technologies both enables vehicles to communicate - either with each other (vehicle to vehicle (V2V)), or with RSUs (V2R). This is a very interesting feature when talking intelligent transport systems in general, as it potentially contributes to safer and more effective roads by providing drivers with relevant information in real-time.

2.3 Urban Traffic Management Control (UTMC)

The Urban Traffic Management Control programme is a British initiative for the implementation of Intelligent Transport Systems (ITS) in urban developments. The programme is managed by a community forum, represented by local transport authorities as well as the UK systems industry. The idea behind the programme is to implement ways of communication and sharing of information between different applications within a modern traffic management system. The applications which are able to send and/or receive information, include anything from cameras (such as Automatic Number Plate Recognition (ANPR)) and dynamic digital signs (such as Variable Message Signs (VMS)) to weather stations and traffic signals. The (perhaps obvious) aim of the programme is to maximize road network potential, thus creating a more robust and intelligent system to prepare for future management requirements and general ability to handle traffic flow.

UTMC is build upon a centralised data network, where everything is controlled from a single point (including a number of substations). This means that all information has to be relayed back to the center of control, before being analysed and ultimately acted upon by other parts of the system.

While the UTMC programme might seem as a huge source of inspiration at first glance, the different choice of network structure makes a big difference in the implementation of such a system, as one of the fundamental pillars of our idea involves a more distributed structure, rather than a centralised one, for reasons we are going to analyse later.

3 Analysis

We have now outlined the problem as well as determined where we currently are in regards to this project; what already exists and what we can potentially use. To start analysing the project at hand in further details, let us first describe the overall problem/task, analyse a few different solutions and then specify the requirements of the system, as this will serve as a basis for what to expect of the final product, and hopefully provide a better understanding of the assignment by establishing some tangible goals.

The purpose of this project is ultimately to improve the quality of life for as many citizens in a limited area as possible, by improving the logical control of traffic signals in said area, utilising sharing of information between the different traffic signals.

Up until this point we have briefly discussed why we want to do this, but exactly how we are going to fulfill that purpose, we will get into later on. For now let us instead focus on the details of what we want to do precisely.

3.1 System overview and overall description

Traffic lights, traffic signals, signaled intersections or even traffic control signals, are used to signal users of said system at road intersections, pedestrian crossings and other places where there is a need to control the flow of traffic. A standard traffic light is programmed (hard-coded) to change which of its lights are lit at preset intervals; the light will be red for a set amount of time, before turning green for a set amount of time (disregarding the, in this context redundant, yellow light).

There exists a wide variety of different types of traffic lights and two general factors have a big impact on how the system works.

Firstly, the roads, pedestrian crossing, etc. coming into and going out from the traffic lights area of effect; are we working with three small roads intersecting at a T-junction, or are we looking at four major roads - each with several lanes of different types (cars, buses, taxis, etc.)?

Secondly, the equipment the system has to work with; does each individual light just have its own local control unit based on a simple timer, or is the entire system controlled based on inputs from various different sensors around the traffic light - such as cameras and induction loops. The point is that traffic lights are not always structured the same way - in fact there can be

anywhere from tens to thousands of different types of traffic lights depending on how many factors you use to distinguish them. This is why there is a need for a dynamic, reactive and adaptive solution.

To give a detailed overview of a traffic light, we ought to think about the actual lights (including types, positions and colors), the lanes (universal lanes, turning lanes, bus lanes etc.), the general traffic rules and regulations, the available technology, along with many other areas of interest. Depending on the country and even state or district of said country, the rules and overall pattern of the traffic lights can vary greatly, as traffic regulations are national- or even state-specific; the rules of traffic are not the same in Denmark as they are in the United State of America for instance. If nothing else is explicitly specified, our model will be based on Danish traffic regulations.

Consider the following visual representations of common traffic light layouts, as these will act as a base of definition for future reference.

Single-lane quad-directional traffic light (standard)
This is the most common layout of a traffic light, as it intersect two roads without any special lanes or lights. When the light i green you are allowed to go all three other directions (relative to the one you came from).

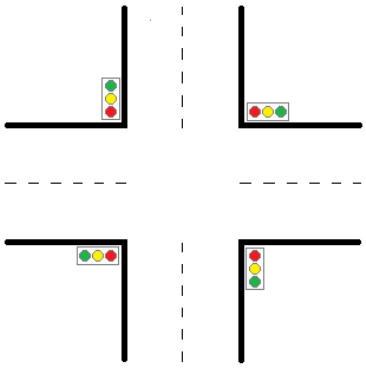


FIGURE 3.1: Standard four-way traffic light

Quad-directional traffic light with explicit right-turning lanes
Turning right is usually the safest option at an intersection like this - not only do you take the shortest path through the intersection, but it is also often possible/safe to turn right even though the main light is red. This is why using an additional lane along with an associated light, can improve the efficiency of the intersection, as it allows cars to turn right even though it is not possible to go straight or left. In theory, all four direction could allow for turning right simultaneously.

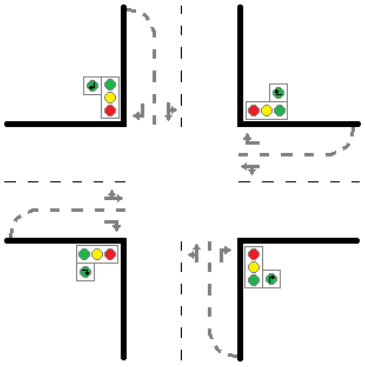


FIGURE 3.2: four-way traffic light with turn-lanes

Three-way junction (a.k.a. T junction or T intersection) with explicit right-turning lanes

This scenario is similar to the one above, however due to the three roads rather than four, the arrows have been adjusted, since you can't go the direction that doesn't have a road. An alternative layout to the three-way junction/intersection, is the Y junction/intersection. There are no function differences between the T junction and the Y junction - only a visual difference, as the two roads forming the top part of the T-shape with a 180 degree angle between them, are rotated upwards forming a sharper angle, thus a Y-shape instead.

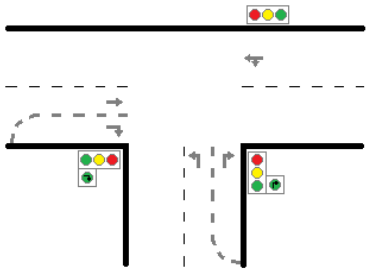


FIGURE 3.3: T-junction with turn-lanes

3.2 Product perspective and user characteristics

Now, let us briefly reiterate the overall description of the problem we are attempting to solve. We aim to improve the overall road infrastructure of a certain area by making its traffic signals more intelligent, thus allowing for a more efficient traffic flow. From the user's perspective, there might not be a noticeable difference from what they have gotten used to, as the actual user interface will not see much of a change at all; the physical interactions between the different parts of the system and the users will not change much. For instance the electric lights of the traffic signal will work as they did before and remain the same as usual - only the logic controlling the lights will essentially be altered.

The users interact with the system automatically by approaching the traffic light, as it will detect an incoming relevant object using an array of different sensors. These sensors will send a signal to the system, letting it know that an object of a certain type has arrived at a certain position/location, so it can include that object in its calculations, and more specifically use it when running the algorithm that determines the next state of the traffic signal. In a network of more than one traffic signal, each node of the network should be able to relay information to other nodes in the network. Just like each node may gain necessary information via its own sensors, it might also receive information from other nodes in the network. For instance, imagine a straight road segmented by two individual traffic signals. When a car approaches the first one, the sensors of the traffic signal will detect it and pass the information onto the system itself. Once leaving the first traffic signal, the system will be aware of its direction, and might let the second node know of the incoming car. That way the second traffic signal will have more time analysing the situation and determining a suitable solution in advance. Once the user has entered the network of traffic signals' area of effect, they will be an actor in the system until they leave the area completely. As long as the user remain in the particular network, it will be a subject of anonymous monitoring, as the system will know (or have a really good qualified guess upon) where the user is at each tick separated by a specific time interval, due to the discrete nature of the system.

Regarding a potential user of the system, we are likely looking at the driver of a car (henceforth referred to as a driver, a car or simply a user depending on the context). As road-specific traffic lights usually provide its primary service to motorised vehicles as well as bicycles and pedestrians, we will be looking at these actors as individual generalisations of their type and functionality (e.g. the actor/object car implies that it contains a person driving said car etc.). All users will be interacting with the system indirectly/passively, meaning that their primary goal likely is not to interact with the traffic signals, but rather to get from point A to point B, where the traffic signals merely acts as interruptions on the path. Therefore it is crucial that a certain standard is achieved regarding availability, reliability, general performance and so on, all whilst also holding up to the topical safety standards - but more on that later.

3.3 Use Cases

3.3.1 Turning right in a quad-directional traffic light with specific turning-lanes and signals

The goal of this use case, is to establish a fundamental knowledge of a general traffic light. (for reference, see fig. 3.2)

Primary actor:

- Car; A standard car including a driver.

Stakeholders and interests:

- Car: wants to turn right at the intersection with as little wait time as possible, whilst feeling safe throughout the entire process.

Preconditions:

- The car is driving towards the intersection, thus approaching the traffic light.

Success guarantee (postconditions):

- The car is going the desired direction, and knows that any relevant traffic regulations has been adhered to.

Main success scenario:

1. The car selects the correct lane for the intended purpose; the right-turning lane.
2. The car drives up to the stop line. *[Alt1]*
3. The car stops and waits for the light to change. *[Alt2]*
4. Assuming that the light follows a red-red and yellow-green-yellow-red cycle, the yellow light lights up along with the red, indicating that the light will soon turn green. The car prepares for driving.
5. The light turns green, and the car begins to drive out into the intersection itself, waiting for any other potential cars to move first.
6. The car enters the intersection and begins to turn right unto an appropriate lane on the new road. *[Alt3]*
7. The car leaves the intersection, thus the traffic light.

Alternative flows:

- *Alt1: There are other cars waiting in the right-turning lane in front of the user.*
  - The car drives up to a position just behind the current backmost car.

- *Alt2: The light is not red to begin with, but instead either red and yellow, or green.*
  - The car continues at a respective pace into the intersection, taking heed of any other cars before skipping to step 6.
- *Alt3: Something is obstructing the way the car is headed (e.g. a pedestrian crossing the road)*
  - The car waits at an appropriate position until the path is clear, before continuing.

Exceptions:
If at any time in the use case before step 7 of the main success scenario, the traffic light malfunctions, the intersections becomes a subject to standard road-crossing rules and regulations (e.g. priority to the right in Denmark), assuming there are no other indications of rules - such as signs or a directing officer.

3.3.2 Turning left in a tri-directional traffic light (T-junction) with specific turning-lanes and signals

The goal of this use case, is to establish a fundamental knowledge of a general traffic light. (for reference, see fig. 3.3)

Primary actor:

- Car; A standard car including a driver.

Stakeholders and interests:

- Car: wants to turn left at the intersection with as little wait time as possible, whilst feeling safe throughout the entire process.

Preconditions:

- The car is driving towards the intersection, thus approaching the traffic light.

Success guarantee (postconditions):

- The car is going the desired direction, and knows that any relevant traffic regulations has been adhered to.

Main success scenario:

1. The car selects the correct lane for the intended purpose; the left-turning lane.
2. The car drives up to the stop line. *[Alt1]*
3. The car stops and waits for the light to change. *[Alt2]*
4. Assuming that the light follows a red-red and yellow-green-yellow-red cycle, the yellow light lights up along with the red, indicating that the light will soon turn green. The car prepares for driving.
5. The light turns green, and the car begins to drive out into the intersection itself, waiting for any other potential cars to move first (when turning left, it is possible that cars are parked in the intersection waiting for a clear path)
6. The car enters the intersection and begins to turn left unto an appropriate lane on the new road; in this case just the single lane available. *[Alt3]*
7. The car leaves the intersection, thus the traffic light.

Alternative flows:

- *Alt1: There are other cars waiting in the left-turning lane in front of the user.*
  - The car drives up to a position just behind the current backmost car.

- *Alt2: The light is not red to begin with, but instead either red and yellow, or green.*
  - The car continues at a respective pace into the intersection, taking heed of any other cars before skipping to step 6.
- *Alt3: Something is obstructing the way the car is headed (e.g. a pedestrian crossing the road)*
  - The car waits at an appropriate position until the path is clear, before continuing.

Exceptions:
If at any time in the use case before step 7 of the main success scenario, the traffic light malfunctions, the intersections becomes a subject to standard road-crossing rules and regulations (e.g. priority to the right in Denmark), assuming there are no other indications of rules - such as signs or a directing officer.

3.3.3 Traversing straight through a network of two traffic lights

The goal of this use case is to give an example of a common scenario using our model implemented on a network of traffic lights (thus there will be less focus on basic and fundamental actions, and instead more focus on the system itself).

Primary actor:

- Car; A standard car including a driver.

Stakeholders and interests:

- Car: wants to cross both traffic lights with as little wait time as possible, whilst feeling safe throughout the entire process.

Preconditions:

- The car is driving towards the first intersection, thus approaching said traffic light.

Success guarantee (postconditions):

- The car traverses both intersections, and knows that any relevant traffic regulations has been adhered to.

Main success scenario:

1. The car selects the correct lane for the intended purpose; in this case the straight lane.
2. The car is noticed by the system via one or more sensors.
3. The system incorporates the car in its calculations (along with any other cars in other incoming lanes), and determines a fair wait time, based on all available inputs.
4. The car waits for the light to turn green.
5. Once the light has turned green, the car traverses the first intersection and enters the outgoing road, towards the next intersection.
6. The first traffic signal acknowledges that the car has traversed the intersection in the given direction, and sends a signal to the next traffic light in that direction.
7. The second traffic light receives the signal, and treats the car as an input to the local system. *[Alt1]*
8. When the car arrives to the second intersection, the traffic light has already prepared a suitable solution, given the input from the first traffic light as well as any standard inputs to the second.
9. The car awaits a green light, and then traverses the second intersection as well.

Alternative flows:

- *Alt1: There are other minor intersections between the two traffic lights.*
  - The second traffic light can not be 100% sure that the output from the previous traffic light will act as an input. Therefore it either uses an average success rate of output=input, or overlooks the data all together - depending on the situation.

Exceptions:
Same exceptions apply as for the previous use case. If a traffic light does not receive any data, even though it is supposed to, it may make use of its sensors instead. If the traffic signal has not received any data nor detected anything from its sensors, the car will not be included in the evaluation of the situation from the system, and might have a longer wait time than the ideal. As always, the traffic light will return to a simple timer control, in cases where no inputs have been detected for a certain amount of time.

3.4 Traffic Simulation Model

In order to demonstrate our final product on a simplified and clear platform, a traffic simulator could be used. A simulator is usually used for the imitation of the operation of a real-world process or system over time, which is exactly what we want in this case. When creating a traffic simulator, we ought to first define and develop the model which presents the characteristics and functions of the system and its processes. This model would present the system itself and is a key part of the simulation, as it is the basis for the imitation of the operation (the operation being a visual representation of traffic flow).

When using a traffic simulator to demonstrate our product, we automatically avoid most of the tedious and resource-consuming processes of working with active real-life systems - such as real traffic lights, and testing and altering becomes much simpler and quicker as a result. A simulator is also ideal for the evaluation and general demonstration of the model, as we can design it to show exactly what we want, thus avoiding too many (if not any) redundancies.

In order to create a suitable model for a traffic simulation, we must first analyse the scope of the assignment carefully. Normally, a statistical theory of traffic flow (preferably based on empirical data rather than theoretical and conceptual data) is needed in order to estimate traffic flow, delays etc., for a specific traffic light. When analysing the performance of a signaled intersection (often using the term Level of Service (LOS)), the model used for specifying delays for instance, are described using a deterministic and stochastic component to reflect the fluidness as well as the random aspects of the traffic flow.

A generally accepted observation of traffic lights, states that the cars traverse the intersection in groups separated by a time factor equal to the time the traffic light is red (also known as the platooning effect). Additionally, the number of cars traversing the intersection during one cycle of green light, never exceeds the throughput of the traffic light. After the intersection the group of cars slowly spreads out due to the fact that some cars drive faster than others - this effect is also known as platoon diffusion. We are not going to focus too much on that however.

Other than the general parts/objects of the simulation (cars, roads, traffic lights etc.), a possible addition to the model, could be the utilization of vehicular networks, so cars travelling in the network will have the option to themselves send data to the different nodes in the system (the traffic lights). This would enable cars travelling in the system to add themselves to a traffic lights inflow giving the traffic light info that it is coming towards it. This would enable the traffic light to adjust it so that once the car reaches the traffic light, the lighting is already green in the best case scenario. This would also enable cars who have a pre-planned route to tell the system its route, enabling the system to make more informed decisions based on known future traffic. This could also be used by public services such as ambulances and fire trucks, enabling them to get through traffic easier and thus faster.

3.5 Traffic Control Algorithm

In order to create an optimal algorithm for each individual signaled intersection, we must seek the perfect balance between safety and efficiency. Consider the quad-directional intersection with explicit right-turning lanes in fig. 3.2. All possible/valid routes a car can take through the intersection are marked in the figure below:

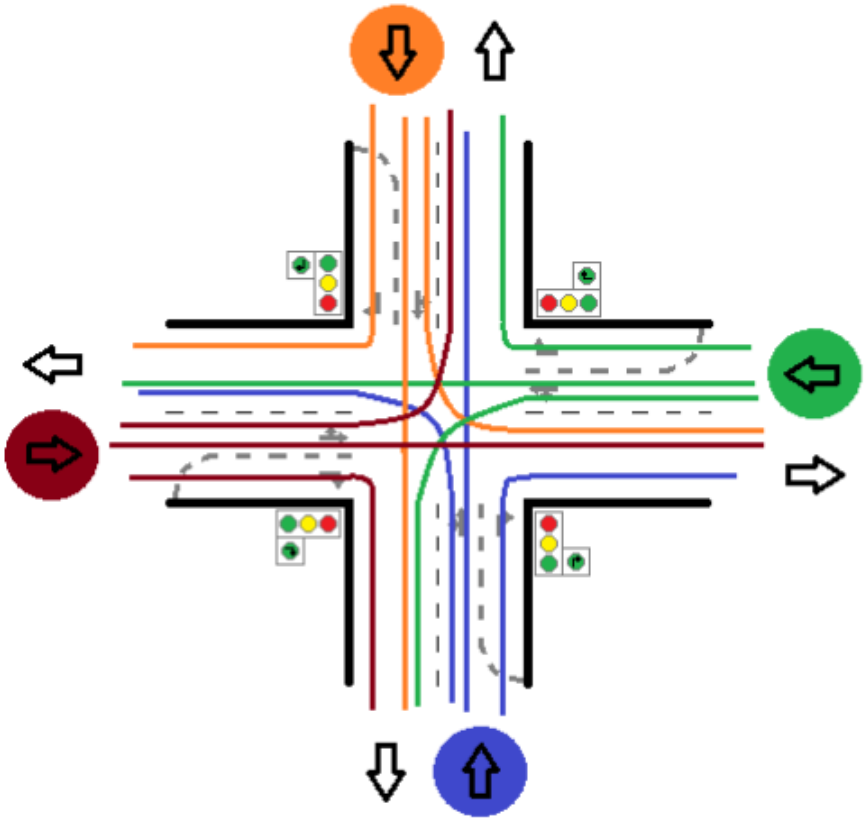


FIGURE 3.4: Valid routes through a four-way traffic light.

At an intersection where you can go all three ways from each of the four directions, there are a total of 12 valid routes that cars can follow, which corresponds to 12 situations we need to account for - and that is not even considering combinations of said situations. Since each place where the lines of different colors intersect means we have a conflict, we can infer that a set of routes are conflict-free if (and only if) none of the lines of the members of the set intersect (it is also important to note that routes also intersect if they arrive at the same lane). To adhere to the standard security requirements of signaled intersections, we must ensure that all set of routes are conflict-free, and to achieve the most efficient intersection, the aim is to find the maximum number of conflict-free sets of routes. There is a few exceptions to the conflict-free rule however, as we might run into some issues when using combined straight and left-turning lanes. The problem with these, are the fact that the left-turning lane intersects with all the other straight lanes as well as two of the other left-turning lanes. This means that while allowing for cars to

turn left from one direction, all the other three straight lanes must be held back. This would have a huge negative impact on the efficiency - so much so that it is commonly (at least in Denmark) accepted to allow for both left and straight lanes at opposite directions at the same time. While this often results in pile-ups in the middle of the intersection waiting to turn left, it has proven to be significantly more efficient as a platoon at least get through each time. While technically breaking the conflict-free rule, it has also proven to be a relatively safe solution, as we are dealing with opposite directions, increasing the chances of the driver acknowledging the oncoming traffic, more so than if the traffic was coming from either side.

Let us now look at some different algorithms and analyse which might be the better solution for what we are trying to achieve.

Randomness
The most common heuristic for signaled intersections, is randomness - as in fixed timers, based on no knowledge of the traffic flow or infrastructure. This method essentially provide a practically random timer, thus disregarding any information about the traffic flow or even the infrastructure itself. However since this method is so basic and simple, it is likewise really reliable, which is an extremely important quality for a traffic light, as we'll get more into later. While being a reliable and generally simple solution, it is very inefficient and far from optimal however, since it has no way of adapting to changes in the traffic flow.

Since randomness is a poor choice due to its lack of adaptability, let us instead consider some adaptive heuristics. Rather than simply deciding and hardcoding the timers into the traffic lights, we should realise that traffic lights are there to help its users, thus we should aim to find a more dynamic solution that continuously satisfies the users in real-time. The only way to do that however, is to get some form of data from the infrastructure.

Highest Flow / Most Pressure
One solution could be to adjust to the highest flow of traffic (e.g. the most pressure/cars). This is not a very common solution, even though it generally lets the most amount of cars through, which is a good thing, since it ultimately relieves the most congested lanes. When traffic lights has no form of communication between them, it can cause even more congestion, since one traffic light has no knowledge of how many cars there are on the output lane.

Longest Queue and Relative Longest Queue
Instead you could try to eliminate the longest queue of cars, as this would be a way to avoid filling up lanes like the solution above might. It does however not account for the length of the roads between the intersections, thus small roads might remain congested as a result. A way to avoid that could be to divide the length of the queue by the length of the road, ultimately getting the relatively longest queue.

While these are arguably better solutions than the randomness, they still only calculate one lane at a time and then compare them to each other.

Best First
A better alternative could be to sum up sets of lanes that are conflict-free and could potentially go at the same time. This way the efficiency of the intersection would be improved, as it would look at the situation as a whole, ultimately letting more cars through.

Other than these somewhat basic methods of traffic control, there are however more advanced heuristics out there, that has the potential to improve quality of life even further. One of them would be to give each car a value corresponding to its priority. This priority could be based on a number of different things depending on the specific situation, but a general one could be the number of passengers. Assuming that we want to move as many passengers through the intersection over time as possible, the car could acknowledge the number of passengers to the system (or the system could detect it via sensors). The traffic light could then add these values to one of the above mentioned methods, e.g. letting a car with three passengers count for three when calculating the longest queue. This solution also has some ethical benefits, as it would increase the attractiveness of carpooling, knowing that you would have a higher priority at each traffic light, there more people you have in the car.

3.5.1 Scheduling (computing)

In computing, scheduling is a way to assign work to resources. A common use of scheduling is Process Scheduling, where different tasks set by different processes are scheduled to execution by the CPU. This can very well be used as an analogy for signaled intersections and their control of traffic; where each lane of the intersection is a process, and its task is to relieve the congestion, thus a task of a certain process is being executed when the light for that particular lane is green, and it is waiting when the light is red. By using this analogy, we can draw inspiration from some of the relevant and more common scheduling algorithms used in computing. It is again important to note, that a set of conflict-free lanes can be processed simultaneously - meaning that multiple processes can be executed at the same time, as if we had multiple CPUs, or a shared CPU using multitasking.

Round Robin
If we look at the real world, and imagine each lane as a process, a very common algorithm used for traffic control, is Round Robin. Round Robin is great because it is simple, easy to implement and eliminates the risk of starvation; a lane with cars that doesn't get processed because the queue length is too short, the priority is too low etc. (depending on the method of control). Round Robin assigns time quanta to all processes in a system with equal/no priority - hence the overall time quanta is divided up in equal portions. In computing, Round Robin is a common algorithm used in process and network schedulers, since it is a so-called cyclic executive, which is an arguably good alternative to a real-time operating system. In computing Round Robin only handles one task at a time, but all processes gets their fair share of the CPU. This can be translated into traffic control, as each lane (or typically conflict-free sets of lanes) are being handled one by one, with all lanes being handled in each cycle. This means that no lanes will be skipped unless explicitly defined; once a task is completed (a lane is empty) the task (the lane) will be removed from the scheduler. Once a car enters the lane, it will then be created once again, to then be included in the scheduler for further cycles.

In a dynamic and adaptive traffic control system, this might prove to be a relatively counteractive strategy however, as there is no particular use of the information available to the system. Round Robin essentially disregards any information given to it, and sticks to the execution time for each cycle regardless of changes in the traffic flow (It can still change the timings, but it would affect all the lanes, rather than just the relevant ones).

First In, First Out (FIFO)
FIFO's are very simple scheduling algorithms, and due to their somewhat fair approach to distribution of resources, it is also quite common in computing. The essence of a FIFO is that is serves the first one to come; like a queue of people waiting to go on a rollercoaster - this queue is commonly know as a task queue. Because context switches can only occur when processes terminate, and there are no need to reorganise the task queue at any point, the algorithm maintains a minimal scheduling overhead. That being said, the actual efficiency (throughput) of the algorithm can be relatively low, due to the fact that processes that require long execution time, can hold the CPU from execution of other processes; it has to finish the process first in line before continuing with the next. This also means, that both waiting time, response time and potential turnaround time can be relatively high. Additionally there are no prioritisation, as every process gets executed in order of which came first. This means that meeting certain deadlines can be tricky, as there is no way to prioritise important/urgent processes over others. However, due to the equal or no prioritisation, there is no risk of starvation, assuming that the CPU eventually completes the task queue.

This is not the most viable option for a traffic light control algorithm, as there is no way of prioritising lanes (processes) with a lot of cars (long execution time), plus once a lane gets a green light, it would have to wait until all cars had been processed before changing state, meaning that though there is

no risk of starvation, some lanes might have to wait a long time for the light to turn green.

Shortest Remaining Time
This scheduling algorithm arranges processes so that the ones with the shortest execution time left gets put first in the task queue. This is also a fairly commonly known algorithm (however not commonly used in modern computing), since it maximises the throughput in most situations, as long as some forms of estimation of the execution time is present. While being decently quick, starvation is possible when a lot of small processes are being queued up; the larger processes run the risk of starving as it keeps getting interrupted whenever a smaller one comes along. This also creates some overhead, as the interrupted process has to be put back into the task queue. It is also bad at meeting deadlines, as the only mean of priority is the time it would take to execute its task. In a traffic light controller however, this strategy becomes a bit more interesting.

Consider a situation where a quad-directional signaled intersection is experiencing a constant flow of heavy traffic in both the northern and southern direction, while only experiencing the occasional car in either the eastern of western direction. The lights will then be green in the 'vertical' orientation most of the time, but when a car suddenly arrives at one of the 'horizontal' roads, it might be of interest to process that single car immediately, and then continue with the standard state again afterwards. Otherwise the car might have to wait a long time for more cars to turn up in the queue, until the point where either traffic light controller deemed it significant or a potential time limit was reached on the current state (to avoid starvation if a sensor failed to detect a car). Combined with a solution where each traffic light controller not only knows of the cars already at the traffic light, but also of the cars on the way from a previous one, it would be possible to plan whether to process the single car right then and there (ideally timing the green light, so it doesn't have to stop at all), or to wait for a few more potential cars incoming.

Fixed Priority Pre-emptive Scheduling
The idea behind this algorithm is to assign a fixed priority to each process and the let the scheduler arrange them in the task queue depending on their priority. The throughput of this algorithm is about the same as the FIFO, as each process is being executed - only the order is different; imagine a FIFO where we first order the processes according to their priority, and then let them enter the task queue one by one in the order of priority. If there are a limited number of priorities (rankings), it would essentially work as a number of FIFO's ordered after the priority of the processes they include.

Using this algorithm, both response time and wait time would depend on the priority of the processes, as higher priority processes would have a lower time than lower priority ones. This also means, that meeting deadlines become more possible, as higher priority can be given to important/urgent tasks. On the other hand, starvation of lower priority processes can occur, as they might be continuously shifted back in the queue if there are a lot of

higher priority tasks incoming.

In a traffic light controller (using our model), this would mean assigning each car a priority immediately when the car enters the area of influence for a specific traffic light, giving each lane a sum of priorities, thus a new overall priority. The lanes (or set of conflict-free lanes) would then be resolved in the order of their overall priority rating.

This could be an interesting strategy when working with multiple types of cars (e.g. standard cars vs emergency vehicles), or cars with an explicit priority (e.g. a car with 4 passengers including a pregnant woman vs a car with one single passenger).

Work-Conserving Schedulers
This is an overall description of a group of schedulers. These types of schedulers always tries to minimize the time a resource will be unused - in other words, it aims to keep the scheduled resourced busy.

This could correspond to a traffic light in a way where we try to always process cars whenever the light is green; ideally a lane should never have a green light if there are no cars in that particular lane (assuming that there are other cars waiting in other lanes which are not green). This is an important feature and ideal for traffic lights, as we ultimately want to process as many cars as possible as fast as possible. Looking for the perfect algorithm might therefore be of relevance in this category.

Multilevel Feedback Queue
This scheduling algorithm is a mix of some of the previously outlined, in an attempt to make a more diverse and dynamic algorithm. Although it utilises strategies and general ideas from three different scheduling algorithms, it is in fact still a commonly known one, as it is the one used in the Windows NT, Windows XP and Windows Vista operating systems. It is a combination of Fixed Priority Pre-emptive Scheduling, Round-Robin and FIFO. This allows the operating system to either increase or decrease the priority of different threads dynamically, depending on whether it has been waiting for a long time, or maybe already been serviced. Each rank/level of priority is represented by a different queue, and Round-Robin is then used on the high priority threads, whereas FIFO is used on the lower ones. The result is, that the response time is short for all threads, and that even very short threads gets executed very quickly. Starvation is however a risk for higher priority threads, as each thread can only use one cycle of the Round-Robin at a time, meaning that long threads with high priority run the risk of starving since only small parts of it gets serviced every time all the other high priority threads do.

3.6 Communication

While the use of network communication for sharing of information between nodes might not be necessary for a traffic simulator running locally, it would be a crucial part of a real life implementation, as the only way for nodes to share information would be over some form of connection. There are many different types of networks, as well as many different protocols, so let us take a look at some of the possible solutions for an implementation of our model onto a real life system.

As we are dealing with a system where all the components interact with each other to achieve a common goal, we can safely say that we are looking at a distributed system where the individual components communicate and coordinate their actions by passing messages of information. This is ideal due to the available, transparent and especially scalable nature of such systems.

3.6.1 System architecture

The architecture of a system involving multiple remote components can be categorised into three primary types; centralised, decentralised and fully distributed.

Centralised
In a centralised network, the core of the system is located in one central location. This means that the central server is the acting agent for all communication in the system; if two nodes of the network wants to communicate, it has to go via the central server. Depending on the function of the network, the central core can also have different functions.

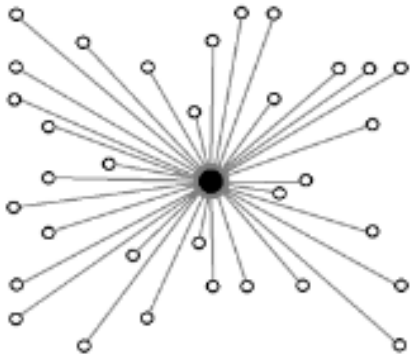


FIGURE 3.5: Centralised architecture

In the example above, the server would act as the postoffice for the messages. In an example where multiple nodes send and receive data to a main server, thus not actually communicating with each other, but rather just the central server, the server would act more like a bank where people can either deposit or withdraw money. A centralised architecture is easy to maintain, as there only exists one major point of failure, and is also relatively quick to implement, as only one major part has to be created, thus it can pretty much be applied anywhere without much effort. The architecture does have its drawbacks however. While it might be easy to maintain a single point of a system, it proves a huge stability risk as only that single part has to fail for the whole system to collapse. If you take out the central point, no points will be connected. Likewise the architecture

is not very scalable, as it has a finite capacity that usually maxes out for really expansive systems.

Consider a traffic network, where each node in the network graph would represent a traffic light (controller). The central node would then function as a primary controller, distributing the messages and relaying them to the intentional end-points. Whenever a traffic light would sent information to a neighbouring traffic light, the message would be sent to the central controller, with the message header specifying the intended receiver. The controller would first receive the message, and analyse the relevant content. Once determining the intended receiver, it would proceed to sent it to that particular traffic light. The two traffic lights would then be interacting indirectly, thus they would not necessarily know of the other traffic light's location and address in the network. While each message would have to be processed by the central control unit, thus running the risk of being delayed, assuming that the controller receives multiple requests continuously, the message would never have to travel on more (or less) than two links, thus only be intercepted once (by the central controller). In a scenario where the delay from the central control unit is minimal, and nodes have to communicate over long distances - beyond other nodes, this could prove to be more efficient, than the message having to traverse through multiple routers along the way. In a system like ours however, the nodes are relatively close together - both physically and regarding potential delay from the network layer. A centralised network architecture would generally improve processing delay, as the maximum (but also minimum) number of routers in the link is one. The transmission delay would likewise be decreased, as the message would only have to be transmitted twice. That being said, both the queuing and propagation delay would be increased (relative to a decentralised architecture), as several messages would have to go through the gap of the central control unit at the same time, and the distance from sender A and receiver C through the control unit B, would always be at least the distance from A to B even if A and C are right next to each other. While the delay and latency of the network can be crucial to the overall performance, the most prominent issue with a centralised traffic network, is the single point of failure; reliability is a key requirement for the system, and the risk of a single failure paralysing the entire system is too high.

Decentralised

A decentralised architecture is the opposite of a centralised one, since there is no longer a definitive single core of the system. In this type of architecture, a node in the network can act as both a client and a server, depending on the situation. Either some nodes are specifically chosen to be intermediate servers (e.g. so-called super peers in P2P), or each node has the option to do both (e.g. peers in P2P). What a decentralised architecture lacks in maintainability and ease of development, it makes up for in improved (relative to centralised architectures) stability, reliability, adaptability, scalability and diversity. A decentralised architecture is usually great for dynamic and volatile systems, as it easily adapts to the requirements, once it has been created to

start with.

In a traffic network, this architecture would involve a number of groups of traffic light controllers, that independently interact with the central control unit. When a node A wants to send a message to a node E, it would first send the message to the central router B of the local group in which the node A i residing. The message would include information about the receiver E, so that routers along the way would know where to relay the message. If we compare this architecture to a centralised one, the distance from the sender of the message to the receiver would be approximately the same for longer distances, but potentially a great deal shorter for nodes closer together, as they do not necessarily have to relay their message via the central control unit, if they are in the same group of nodes, with the same local router, thus the average distance between nodes are shorter than that of a centralised architecture. Consider our model of a traffic network; as the interest of each traffic light lies solely with its neighbours, it only cares about incoming traffic from said neighbours, connecting these in a cluster with a shared local central point would be desired, as a route via a global (within the network) central control unit, would be ridiculous given the distance the message should travel.

FIGURE 3.6: Decentralised architecture

Fully Distributed

A fully distributed architecture is a variant and ultimately an extreme example of a decentralised system. In a distributed network, each component knows about only their neighbour to start with. In the graph to the right, each node acts as a component in the system (e.g. a traffic light controller) and each link shows the connection between two nodes. Since each node only knows its immediate neighbours initially, they must exchange messages in order to explore the graph (i.e. the network). What a decentralised architecture masters over a centralised one, a distributed structure improves even further. This does however mean, that the disadvantages of a decentralised structure is even greater for a distributed one. The ultimate essence of this, is that a distributed system can be difficult to implement and maintain, but is otherwise brilliant for an ever-changing large network.

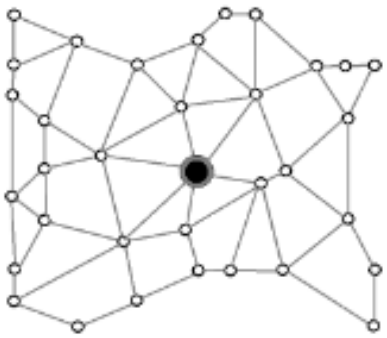


FIGURE 3.7: Distributed architecture

If we once again consider our model of a traffic network, a fully distributed architecture would involve all nodes to be connected with their nearest neighbours, ultimately creating a spider web-looking network, where all nodes have at least two links. A route from node A to node B, would then involve using a shortest path algorithm to determine the prefered route based on minimal total delay. By utilising this layout, the queuing delay would usually be relatively low, as the overall load of the system would be distributed out to all the different nodes - with the more central nodes carrying a heavier load than the outer ones, due to the natural routing between nodes (i.e. the probability of a receiving node to be located on the other side of the general centre is greater than on the same side - disregarding the function of the network). Assuming that the processing delay and transmission delay of each individual router is fairly low, the propagation delay for the message would likewise be decreased compared to the architectures described above. When determining the route for a message, dijkstra's algorithm is generally a good (if not perfect) choice - depending on the situation. It usually works extremely well for a network layout like this, as the essence of the algorithm is focused on finding the shortest path between two neighbours, and as we have multiple neighbours for each individual node, this often is the best choice. As the basic variant of Dijkstra's algorithm only finds the shortest path between two neighbours, a recursive variant where the shortest path between two nodes in a network connected be multiple other nodes, is often the better choice, as the path between A and C, might not always be the same as the shortest path from A to B plus the shortest path from B to C (mainly if something like the delay is significantly different for each link).

Now that we have established some of the primary types of architectures, let us take a look at some of the more common architectures in (general) distributed systems that could be relevant for this type of system - more specifically, there are generally four primary types of distributed architectures; layered/multitier, object-oriented, datacentric and eventbased/eventdriven. However, as the layered structure is a client-server architecture where each part of different responsibilities are separated into tiers (i.e. layers), this is not really a relevant architecture for a traffic network. We will therefore go over the latter of the three.

Object-Oriented
An object oriented architecture, is one where objects (just like in object-oriented programming) are distributed across different components (or address spaces) in a network. The objective of using this architecture, is to make the location of the objects transparent; ultimately making remote objects seems as though they are local. This means that each distributed object is able to access data and invoke methods on remote objects (Remote Method Invocation (RMI)) - just like remote procedure calls in object-oriented programming. The RMI protocol usually involves message-passing, where the caller-object sends a message to the remote object, asking for permission to access it. If permission is granted by the remote object, the result of the invocation is sent back

to the calling object.

In a traffic simulator, this is often how the communication architecture is structured, if the simulator is implemented following an object-oriented design pattern. Likewise this could work in a real world network of traffic lights, as each traffic light (i.e. node in the network) would function as an object, and whenever it needed a fresh portion of traffic information, it would invoke a method on the object which held the information - using RMI. This could be a decent solution when implementing the traffic control model for a local simulation first, as the transition onto a real world network would be very similar, in that each procedure call would simply be replaced by a remote procedure call.

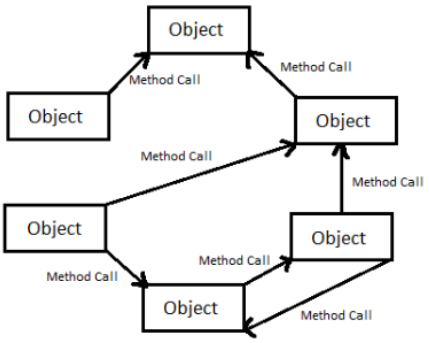


FIGURE 3.8: Object-Oriented architecture

Datacentric

A datacentric architecture, is a centralised network in which the persistent data is shared between a number of components. This means that all the eligible components are able to publish and/or request data depending on their privileges. It is centralised in nature, as the system data is gathered in one single place.

In a traffic network, this might be a suitable solution, as the individual components of the system has a lot of data to publish - just as they often want to receive new information about the traffic flow. Gathering the data in the cloud (so to say) might therefore simplify the network structure. Each traffic light (controller) would simply request a certain form of data from the cloud, that had previously been published by another traffic light, ultimately maintaining loosely coupled components.

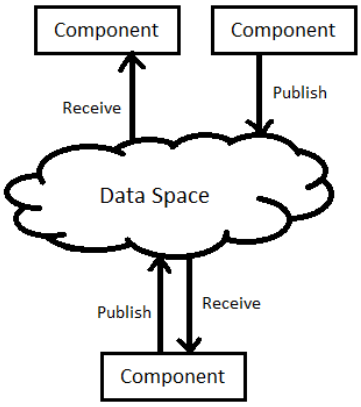


FIGURE 3.9: Datacentric architecture

Eventdriven
An eventdriven architecture focuses on interactions using events. An event is a sudden change of state for the system. Whether it being a component publishing some data, or another component requesting a delivery of some data, the state of the system is altered, and an event is triggered. The events are not actual objects, but rather a concept. The actual object being sent back and forth on the event bus, would usually be messages sent asynchronously.

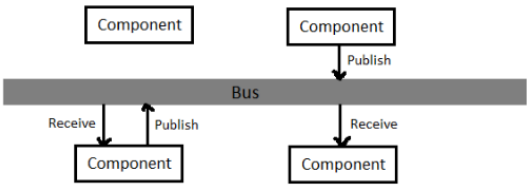


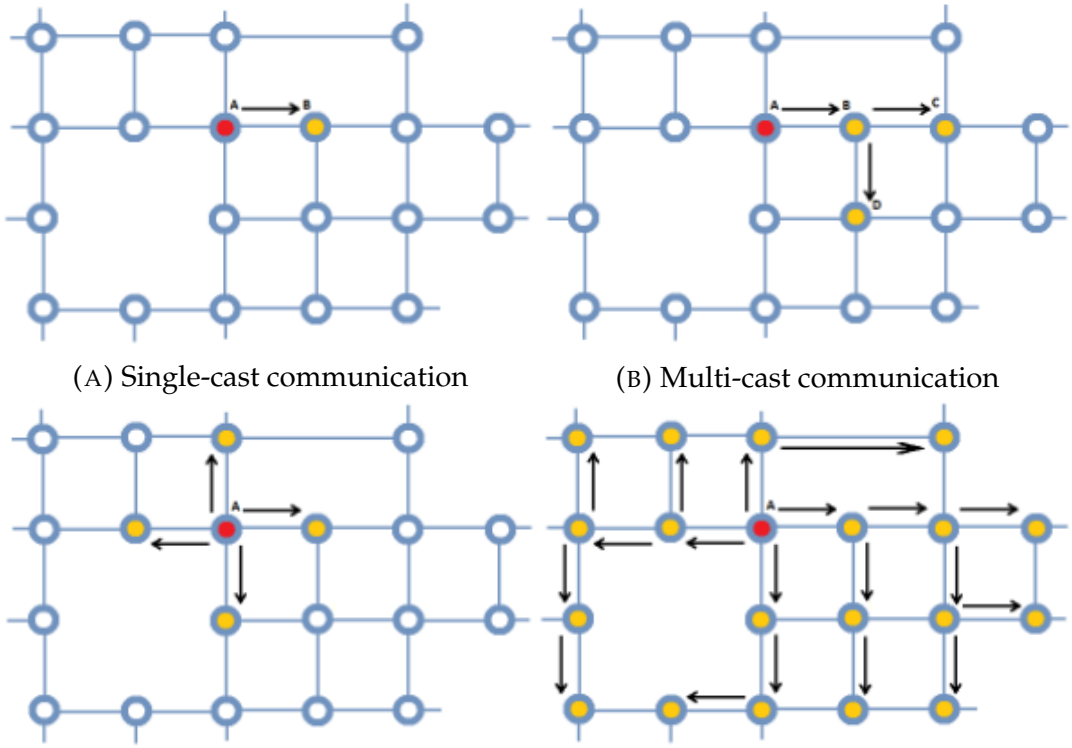
FIGURE 3.10: Eventdriven architecture

In a traffic network, an event could be triggered whenever a traffic light component experienced a flow of traffic traversing the intersection, and published this data. All other components subscribing to this type of data (e.g. physically neighbouring traffic light components) would then receive the data without technically knowing anything about the component that originally sent the message. This scenario will be analysed in further details in the subsection *publish and subscribe* under the next section.

3.6.2 Communication

Regarding the actual form of communication over the network, there are typically three primary methods for sending messages. If a message is sent directly from one component to another, the method of communication is called single-cast. Similarly, multi-casting would involve the message being sent to a specific group of components, rather than just a single one. The last option for sending a message over a distributed network, is broadcasting. By broadcasting a message, the message is sent out to all other components in the network. When using a publish-subscribe pattern, the receiving components could choose to subscribe to a certain component, or even a certain type of message. Whenever the sender publishes a message, all subscribing components will receive it, whereas the ones not subscribing will not. Another variation of this could be any-casting, where a message is sent to practically anyone - usually the nearest neighbours in the network.

Consider the following four examples - each of one of the abovementioned methods of communicating in a traffic network. In the graphs, each node represents a traffic light (controller) and each link represents a wired connection - possibly along with a physical road between the two relevant traffic lights.



(A) Single-cast communication (B) Multi-cast communication

(C) Any-cast communication (D) Broadcast communication

FIGURE 3.11: Different methods for communication

In the first graph (see fig. 3.11(A)), we see an example of using single-cast communication to send a message from traffic light A to traffic light B. By analysing the data, A realises that only B would show interest in this data, thus sends it directly to B and no one else in the network. If A has knowledge of the network structure, or even just its neighbours, and it knows the direction of its processed cars, it will know exactly which traffic light will receive the cars as input. This allows A to explicitly alert B that a certain amount of cars are incoming, giving B time to assess the situation and adjust its timers accordingly. If each node not only has knowledge of neighbouring nodes, but also of the exact distance and average speed limit between them, it would even be possible to not only alert the receiver about the amount of traffic incoming - but also at what exact time they will arrive, allowing B to make an even more precise decision.

In the second graph (see fig. 3.11(B)), we see an example of using multi-cast communication to send a message from traffic light A to the group of traffic lights B,C,D. Once again A analyses the data, and realises that not only B, but also C and D might have interest in it. If A knows at least two levels of neighbouring nodes (i.e. its own neighbours and their respective neighbours), if can send the message to all nodes that are up to two links away from itself. That way B get the information like with single-cast communication, but C and D also gets information that traffic is incoming to B. Based on a statistical analysis, both C and D might deduce that some (or all) of that traffic may comes from B in the future. This would give C and D more time to evaluate and though some of the data would be based on a qualified

guestimate based on either randomness or previous experience, the system would have a better overall knowledge about the traffic within the network - potentially letting each node make better decisions.

The third graph (see fig. 3.11(C)) shows an example of using any-cast communication, where a message from A is sent to practically anyone - in this case however the message is sent to its nearest neighbours. The message is essentially copied and sent out four times - each with a different receiver in mind. Consider a traffic network where each traffic light did not know where its output was going. Sending the message to all the nodes immediately connected to it, might be a good solution, as it would not have any way of knowing which specific node to send the information to. This solution would not be ideal however, as the system would start to overflow with redundant information, due to the fact that nodes to the north, west and south of A probably would not care much about how much traffic was going east.

An example of broadcast communication in a traffic network is shown in the fourth graph (see fig. 3.11(D)). When using broadcast, the message is sent from A to all other nodes in the network. Since any-casting might be an unsuitable solution to an intelligent traffic control system, broadcasting might at first glance seem as an even more extreme version of that, thus an even worse solution. Broadcasting a message to an entire system can however prove fairly useful in certain situations. Assuming that each node can handle/analyse all the received data without a significant disadvantage in performance, a system where all nodes knows everything about all other nodes, might be a great solution, as each node would have access to all available information in the system. Unless an extremely complicated statistical model was implemented (for a large network of nodes), this would likewise involve a ton of redundant information, which would mean a potential waste of performance.

Publish and Subscribe

Publish-subscribe is based on an eventdriven architecture, and refers to a message pattern where the components sending messages (publishers) does not know anything about the components receiving the messages (subscribers). When the publishers publish a message, they specify not the intended receiver of the message, but rather the topic or content of said message. The subscribers then subscribe to a certain topic or form of content, and thereby only receives data that is of interest to them, without necessarily knowing anything about who originally published it. The messages is initially sent to a so-called message broker or event bus (as the bus in fig. 3.10). It essentially receives all the published messages, translates them from the message protocol of the sender to the message protocol of the receiver, and relays them to the intended receiver.

The publish-subscribe paradigm using a message broker, is typically supported by a Message-Oriented Middleware (MOM), which functions as the infrastructure between the publishers and subscribers. The middleware is essentially what allows the messages to be distributed over (heterogeneous)

networks. MOM practically introduces a new layer to the OSI model, between the transport layer and the application layer, and basically functions as the "to" in peer-to-peer for instance.

There are three primary variations of the publish-subscribe pattern; list-based, broadcast-based and content-based.

In a list-based publish-subscribe pattern, a list of subscribers is maintained for each subject (e.g. topic) of the messages. Whenever an event occurs (i.e. when a message is published), it goes through the list of subscribers for that particular subject, and relays the information to those on the list. In the traffic network, the subject could refer to the position and direction of traffic flow, and each traffic light could then subscribe to the subjects of their respective interest. In practice, this would typically be the inputs to the traffic light, thus the outputs from neighbouring traffic lights with direction towards itself. In a system where each node has the knowledge of traffic beyond their immediate neighbours, this could quickly become a long list of subscriptions for each subscriber.

Let us say that traffic light A has detected some traffic traversing the intersection from north to south. It then publishes this information to the even bus, where the subject is evaluated (traffic moving southbound from node A). The message broker locates the list of subscribers for that particular subject (e.g. traffic light B that is located south of A in direct succession), translates the message (if needed) and sends it to B that process the message, thus getting the information about the incoming traffic.

In a broadcast-based publish-subscribe pattern, the message is instead broadcast onto the network when a message is published. All subscribers technically receive the message, but only if the subject of the message fits their subscription, the message is processed.

As previously mentioned, this could also work for a traffic network, as each traffic light publishing a message to the event bus, would trigger all subscribers to check the subject of the message and only process it if the subject matches its subscription. This could be done for groups of traffic lights in a local area networks rather than the entire network, to reduce redundant message being propagated.

Content-based publish-subscribe is different from the two previously mentioned variations, in that it focuses on the content rather than the topic of the messages. Where the distribution of messages in a broadcast-based approach is coupled to a multi-cast tree based on the Transmission Control Protocol (TCP), this variation overcomes this limitation by utilizing the actual content of the messages, thus allowing for a direct route to the subscriber that subscribes to that exact content.

In a simple traffic network the content of the messages being published would be relatively basic, but imagine if each message included information about how many passengers were in the cars, the type of vehicles etc. - it would then quickly become more relevant to use this variation of the publish-subscribe pattern.

3.7 Specific requirements

Up until now we have considered a general problem, for which we want to make a solution. Let us now specify the requirements of said solution to ensure that the goal will be met, whilst complying to any external rules. Since we are working with real life public traffic lights, which not only holds a risk of crippling an entire city, but is also a epicenter of risk for human safety, areas like safety as well as reliability and performance are especially important to address.

3.7.1 Functionality

Each node in a given network should be able to send and receive data, as well as utilize and interpret data from either local inputs or remote signals without any semantical differences.

The system should be able to handle large amounts of data, without any noticeable latency and the exchange of data over a network should be secure and reliable.

Given the relatively general problem we are attempting to solve, the system needs to be fairly generic and adaptable; the required effort to implement the model onto any given network of traffic lights should be as low as possible.

At any point in time, it should be possible to disable the system on either a local or global basis, in case of malfunctions or any other situation where it might be preferred to return to a simplified model temporarily; our model should be implemented as an addition to the already existing model, rather than a substitution.

In addition to previously mentioned functionality, it is to be assumed that each traffic light as a minimum acts and functions as a standard traffic light.

3.7.2 Usability

It is important that the users does not have to alter their routine in order to traverse the traffic lights. As previously mentioned, there does not need to be any changes to the interfaces that the users directly and consciously interact with - such as the actual lights and the lanes of the traffic signals. Other than the hopefully improved traffic flow, the users should not notice any game changing differences.

Other than that, it is to be expected that a certain standard is met when it comes to responsiveness and aesthetics.

3.7.3 Reliability

Traffic lights are critical when it comes to the enforcement of traffic laws and regulations, as well as general safety. Therefore it is very important to ensure the availability of the traffic lights. Certain backup/safety measures can be implemented to ensure acceptable availability during a technical failure. For

instance the traffic light could make use of a more simple form of control, in the case of a failure to the current control unit (e.g. make use of a set timer, if the otherwise prefered means of control are malfunctioning).

Another way to ensure as high an availability as possible, could be to alert the relevant authority of any failures. Letting the proper technician know if there is a technical malfunction, could bring the downtime down significantly, as the problem could be fixed right away. Regarding the physical composition of the traffic lights, we are going to assume they are as robust and durable as standard traffic lights.

In terms of accuracy and general correctness, it is of utmost importance that the users are not misguided, as that could lead to a severe break of safety. Frequent monitoring and general maintenance should prevent any malfunctions or at least detect them, so any appropriate fixes can be made in time.

3.7.4 Performance

As the overall goal of this project is to improve the efficiency of traffic lights, the performance of the actual traffic lights are also of utmost importance. Implementing a new measure of control for a network of traffic signals, would be for nothing if each traffic light is slow and sluggish, or if the communication between each node in the network is too delayed. Therefore it is crucial that the speed and general efficiency of the system is as high as possible.

Other than being fast and efficient, it is also important that the cost of the system is as low as possible. Just like it needs to be fast and efficient, it also needs to limit its resource consumption; an appropriate balance must be found between those two areas.

As an important quality of the system is its scalability, it must also be capable of upscaling the system without any major upgrades other than of course the upscaling itself.

3.7.5 Supportability

Serviceability, maintainability, sustainability, repair speed, testability, flexibility, modifiability, configurability, adaptability, extensibility, modularity, installability and localizability.

These are all very important quality attributes, when working with a generic and scalable solution to a dynamic problem. It makes sense that each and every one of these qualities are in the back of our head when implementing the solution. As mentioned briefly when discussing both reliability and performance, the ability to easily implement, maintain and configure the system is extremely important, as this will make the attractiveness of the system skyrocket.

4 Design and Implementation

4.1 Traffic Simulator

As mentioned earlier, the core of any simulation is the model on which it is build. Our model consists of, and primarily focuses on, the description of a traffic light, as this is the primary object of interest when working with traffic control and/or management. As we aim to demonstrate the effect of using multiple traffic lights that all share the same knowledge, our simulation likewise supports the existence of multiple traffic light objects within the system.

Throughout this section, it is recommended to refer to the design class diagram of the traffic simulator (including the algorithm) in appendix A, as this gives both a general overview of the structure and more detailed information including variables and functions.

The traffic simulator consists of a bunch of different objects, that for the most part can be categorised into two primary types; nodes and links. Nodes include objects like Intersections and Spawn Points, and is where there is an active form of control present. It is only within nodes, we can actively change the outcome of the simulation; if we alter the control algorithm of the traffic lights or decide to spawn more cars at a specific point, the result of the simulation will be different than if we had not changed anything. Links on the other hand include objects like Roads and parts of the roads like Lanes, and acts as connectors between nodes - just like roads are connectors between intersections in the real world. Links have a passive form of control, as they can only be interfered with indirectly; if we want to change the outcome of a situation on a link, we have to make a change to a connected node in a way the changes the outcome of that particular link.

With these two types of objects, it is possible to construct any traffic network from a real world scenario into the traffic simulator, as long as it consists of valid subcategories to either nodes or links - for instance the simulator supports intersections like three- and four-way traffic lights among others, but not roundabouts or traffic lights with more than four inputs and/or outputs. With this basic structure in mind, let us take a more indepth look at the model on which the simulator is build.

4.1.1 Traffic Simulation Model

A traffic light, in our model, acts as the base of operation for our traffic controllers. It consists of a number of road objects, a lightController, and a lineController. The lightController, as suggested by the name controls the lighting patterns in the traffic light, controlling for whom the light is green/red and for how long. As such it is in the lightController the traffic light algorithm is implemented. The lineController controls the actual flow of the traffic in the traffic light. It controls which lane incoming road traffic is added to and moves traffic through the traffic light adding outgoing traffic to the respective roads outputs. The roads represent the incoming and outgoing roads in the traffic light, these respectively leading to a new node in the system - typically another traffic light, and are used to determine the time it takes to travel to the next node by using the length and speed variables. Roads also contains the lanes which the trafficLight's lineController uses to control the direction of the traffic. Traffic lights and so-called Side Roads (T-junctions) are the main building nodes the simulation program uses to section off roads in the system. This is an important point, as one might think of a traffic network as a collection of roads and intersections. Instead we are using the intersections as the reference point; every object in the system exists relative to the traffic lights, rather than the whole network.

We once again refer to the design class diagram in appendix A, to give a visual presentation of the system.

The only thing a traffic light has other than a number of functions, on an abstract and physical level is an identifier, abbreviated id. Most of the objects in our model holds an id, as a way to reference them from all over the system. When dealing with a generic and expandable system, where multiple instances of the same object can exists, it is often a good idea to give these objects some form of uniqueness, e.g. in form of an identifier.

Within the network, multiple traffic lights can exist, and at least one must exist - otherwise the purpose of the network is defeated, as the traffic lights are the primary part of our model, as mentioned above; while a network of just roads can be interesting to look at, it is not really the level of complexity we are after, since an analysis of the network becomes extremely simple, and alterations become more or less obsolete.

Since the traffic lights serves as the centers of attention, roads are attached to the traffic lights - not the other way around. For a traffic light to have a general purpose, it must have at least three connected roads.

Each section of road has a name (which is an identifier indicating the position of the road relative to the traffic light - e.g. 'North' or 'South West'), a length and a speed limit. The length and the speed limit are both common attributes of a road and are used when calculating the time it takes a car to traverse through the section of road (from one intersection to another). A road also has both Ways and Lanes. These are both objects in the system, and acts as subdivisions of a road. A way is like a direction, and since a road is positioned relative to a traffic light, the two possible directions of a road

(aka. the two Ways) are in and out. If a road only has one associated way, it functions as a one-way road.

Technically a link between two traffic lights A and B, works as two individual roads; one that acts as the output for A and input for B, and one that acts as the output from B and input to A.

Lanes are what determines where the cars can go at the intersection. For instance, a road can have a lane to go left and straight, one to go just straight and one to go just right. This alleviates some of the pressure at the intersection, and allows for a more specific and detailed algorithm to be used for the traffic control - which is a good thing, since each car becomes part of a smaller group when analysing the flow/pressure, which in turn means more focus on each individual car.

The way, or direction, is what determines the flow of traffic; number of cars over time, whist the lane determines the line/queue of cars. The width of the lane indicates the number of lanes of the same type, e.g. if there are two right-turning lanes and one straight lane, there are two lanes where the right-turning one has a width of two. The lanes is also the object that holds the actual traffic light, as the light might be different for each type of lane. The light object only has a state, which indicates what color is currently shown.

Since the road is the overall pathing object, it is also here the spawn nodes are attached. The spawn nodes are what spawns the cars, since we need to actually create cars when working with a simulation. These spawn nodes are usually placed around the outside of the network, acting as the inputs to the system as a whole. The spawn node dictates the flow, which indicates how many cars are spawned over time.

4.1.2 In depth view

Now that we have established the general structure of the traffic simulation model, let us take a closer look at what it actually does. (See appendix A for a design class diagram over the system).

The Main class is what initiates and starts the simulation. It creates the traffic network by initialising the objects and connecting them, and sets the various parameters, such as which algorithm each traffic light should use, the length of the roads etc.. As the simulator runs locally simulating a distributed network, each node in the simulation (i.e. traffic light, side road or spawn node) is started as their own thread, so they are isolated from the rest of the objects.

Once the simulation has started, cars will begin to spawn at the declared spawn nodes. The spawn nodes makes use of the Random java class to generate traffic. It does this so as to simulate a degree of randomness to the traffic flow since real world traffic have a certain degree of uncertainty to it, and in this way try to mirror real world traffic a bit more. The Random class generates a stream of pseudorandom numbers (with the use of a 48-bit seed which is modified using a linear congruential formula) which is used to determine whether new traffic is spawned or not. If the number (which is between 0 and the max spawn value) is less than the set spawn value, then traffic is

spawned and driven into the system. A new number is drawn every spawn interval (msec) which controls how often a spawn node tries to generate traffic. By default the spawn value is sat so there is 50.01% possibility for traffic to spawn at each spawn interval. This means that if the spawn interval is sat to 200 msec, the probability of one or more traffic spawning in one second (five spawn intervals) is 96.88%. And the probability of five traffic spawning in one second is 3.13% (for calculations see appendix B).

Each instance of a traffic light, starts both a light controller and a line controller, each in a new thread, and then proceeds to run for the rest of the simulation continuously yielding for other threads; essentially giving itself lower priority for execution by the CPU, and allowing other threads to be executed until the end of the simulation.

The light controller then decide the light configuration based on the chosen algorithm for the associated traffic light, each so-called lightTime, which is essentially the tick/refresh rate for the light signal - thus also the minimum amount of time each light will remain on for.

The line controller essentially updates the lines on each incoming (to the associated traffic light) road by either incrementing or decrementing each individual lane coming into the associated traffic light, based on the arriving flow (at the edges of the network, this would be from spawn nodes exclusively). The line controller can basically be seen as a policeman in the middle of the intersection guiding the cars with his og her hand signals. The line controller controls where the traffic goes, by practically moving the cars around, whereas the light controller controls the light signals, by changing them according to the result of the algorithm.

4.2 Traffic Control Algorithm

The system supports three different traffic control algorithms; one using fixed timers, one using sensory inputs and one using data from neighbouring traffic lights. The first two are relatively simple and is basically simplifications of the latter. We are therefore going to describe the algorithm that was developed with communication between traffic lights in mind.

The traffic algorithm determines two things; which light is turned on and for how long. It does this by looking at the current lines at the traffic light; which light is currently turned on, and incoming traffic. All the traffic algorithm knows about the incoming traffic is their arrival time to the traffic light. With the data of the incoming traffic and the current lines, the traffic algorithm tries to determine how long it will be before there is a “hole”(a period of time with no traffic) in the traffic flow at the incoming roads, and how long these “holes” will be. For simplicity’s sake will we call the time before a “hole” appears on a road clear time and the duration of the hole, hole time. The traffic algorithm then tries to avoid having a light green in any direction where there is a hole, thus only having the light green in a direction for that direction’s clear time. When it becomes time to change the light to the other direction and the direction which is being changed to has a clear time of zero

(no traffic), the traffic algorithm keeps the light green in the current direction (doesn't change the light) for the duration of the other directions hole time (until there is traffic). The maximum hole time that a road can have is that road's travel time. Travel time is the time it takes for traffic to transverse the road to the traffic light. This is a calculation involving the roads length and speed. This is the maximum hole time since this is the maximum prediction range the traffic algorithm has (it only knows what traffic is currently travelling directly to it). This is possibly an area of further development. Light time is the time the traffic algorithm determines that the light should be green for. This is typically a directions clear time however it does have a max value that it's allowed to be. Lets call this max value; max light time. Max light time is determined by the traffic light to be the max time allowed to pass before the traffic algorithm has to recalculate and try to change the light. This is so as to prevent the traffic algorithm from determining that there should be green in one direction for a very long time, do to high traffic, and then not going back to recalculate in a long time. This would result in the traffic light not changing in a long time, ultimately resulting in starvation. Thus max light time is the max time before the traffic algorithm as to recalculate and thus give priority to the other directions. This however doesn't prevent the the traffic algorithm from not changing the light if there is no other traffic. It does however protect against long clear times.

One limitation with the current traffic algorithm is that it doesn't know which way incoming traffic wants to go, and since it calculates worst time clear times, meaning that it calculates the longest possible clear time of a road, the predicted clear times are sometimes longer than the actual clear times. This is primarily a deficit when incoming traffic arrives before the roads longest line is cleared. This causes the arrived traffic to be theoretically added to the longest line thus increasing the clear time. In practice however, the incoming traffic might not be going the way where the line is the longest but a way where the line is shorter, thus not actually increasing the overall clear time. However since the traffic algorithm doesn't know where incoming traffic wants to go, it makes predictions based on the worst case scenario.

5 Evaluation

Now that we have analysed the problem at hand and proposed a solution, let us evaluate the result of the chosen solution, by evaluating tests and discussing the results of them. The purpose of the implementation was to demonstrate our algorithm in an environment simulated to function as a real life traffic network. The evaluation will therefore focus on the differences between our algorithm - which supports communication between traffic lights, and already commonly used ones, that does not.

5.1 Test

In order to measure how our developed traffic control algorithm measures up against other implemented algorithms such as time switched based ones, and sensor based ones, a bunch of tests simulations with the different algorithms were run with the results measured against each other. These test simulations were run in a bunch of different scenarios with varying traffic flows so as to get an idea of how the different approaches handle different scenarios. Each test is run with 100 simulations of each algorithm where the average wait time for traffic is measured and the overall average wait time is computed for each algorithm. It has been decided that we measure the effectiveness of the algorithm on the user's(traffic's) average wait time in the system, since the less time that the users hold still for a red light the better the experience.

The time based algorithm has no knowledge of the traffic flow throughout the network, thus the light configuration is purely based on fixed static timers. The sensor algorithm is based on the time based algorithm, but additionally it utilises simulated sensors, that are able to detect cars when they arrive at the given traffic light. This means that it knows how many cars that are currently in the lanes for the traffic light, thus it can change the light configuration accordingly, so as to accommodate for lanes with higher traffic flow. Our developed algorithm, is somewhat similar to the sensor algorithm, in that it knows about any cars that are currently in line for the intersection. What makes it different from the other two however, is its ability to interpret traffic data from nearby traffic lights, and thereby plan the light configuration according to not only the cars already waiting, but also the cars the are approaching the intersection.

5.1.1 Test 1

The graph below shows the resulted average wait times of traffic in Test 1.

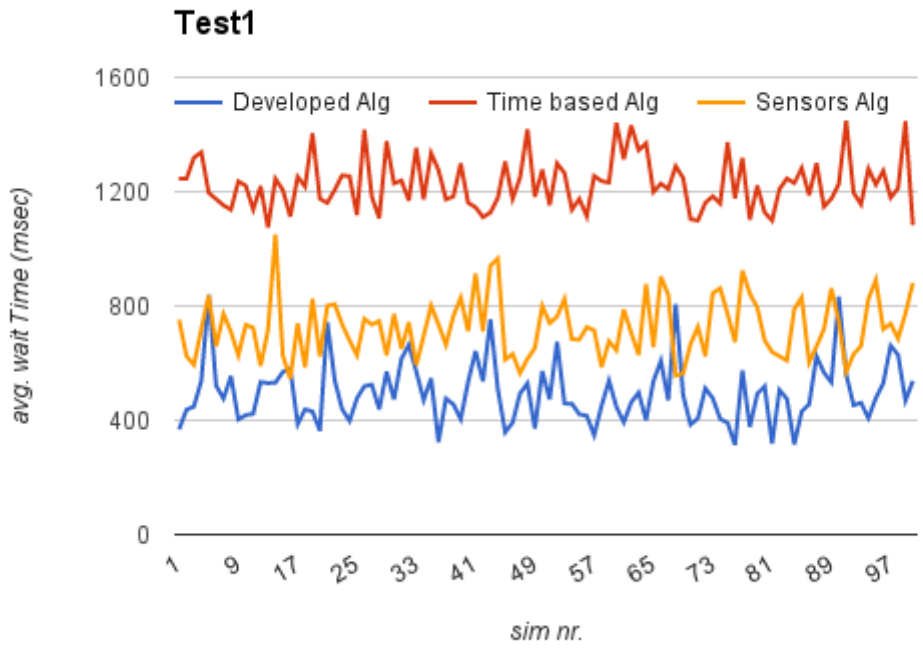


FIGURE 5.1: One traffic light, four roads, no turn lights and moderate traffic flow.

Test 1 is run with 100 simulations of each algorithm with a moderate traffic flow in each direction. The east and west direction has a slightly higher traffic flow. The test involves one standard traffic light (4 roads and no turn lights), where there is a spawn node attached to each incoming road. The roads have varying speeds, lengths, and turn chances to each other and are the same in all the simulations. For all the exact value specifications for the test please refer to appendix C. The overall average wait times computed for the three different algorithms are 495.83 msec for the developed algorithm, 1227.1 msec for the time based algorithm, and 723.6 msec for the sensor based algorithm. This clearly shows that the average wait time at the traffic light when a time based algorithm is implemented is nearly two and a half times greater than when our developed algorithm was used in a standard traffic light with moderate traffic. This indicates that with the developed algorithm the time users(traffic) spend waiting around for a red light is significantly lower and thus ensure a better overall user experience. The same can be said when comparing to the sensor based algorithm. With the sensor based algorithm the average wait time is nearly one and a half times greater than with the developed algorithm, again showing an improvement in the user experience when the developed algorithm is used.

5.1.2 Test 2

Test 2 is a test involving our developed traffic light algorithm against both a standard time based and a standard sensor type algorithm in a standard traffic light with spawn nodes connected to the four incoming roads. The test looks a lot like test 1 and the network is also the same. The only difference is that test 2 is overcrowded with traffic, so the spawn interval on the spawn nodes are half of what they were in test 1, resulting in a lot more traffic since traffic is being spawned more often. This is to test how the different algorithms compare when there is overcrowded traffic from all directions. The graph below shows the result of test 2:

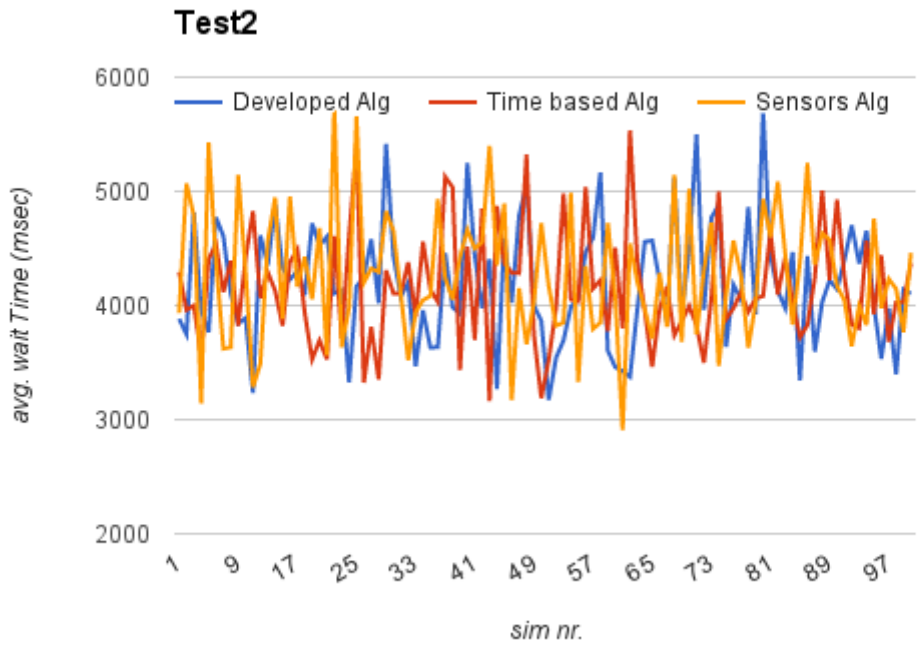


FIGURE 5.2: One traffic light, four roads, no turn lights and extreme traffic flow.

As seen above, the average wait times between the three algorithms are very similar. This is also as expected since the test involved there being an overflow of traffic from all directions. This results in there constantly being long lines in all directions, and since there is no priority on any specific thing (all traffic has the same priority), the developed algorithm will switch between the directions generating max light time, which is sat to be the same as light time in the time switching one. A similar thing happens with the sensory algorithm. This results in the average wait times being very similar which is also evident on the overall average wait times which is 4194.62 msec for the developed algorithm, 4160.62 msec for the time based one, and 4263.91 msec for the sensory algorithm . For the specific specifications for Test 2 see appendix D.

5.1.3 Test 3

Test 3 looks at how the developed algorithm stands up to the other algorithms in a traffic light with a bigger high traffic road and a smaller low traffic road. This is a typical example of a main road intersecting with a smaller road. The graph below shows the result of the test:

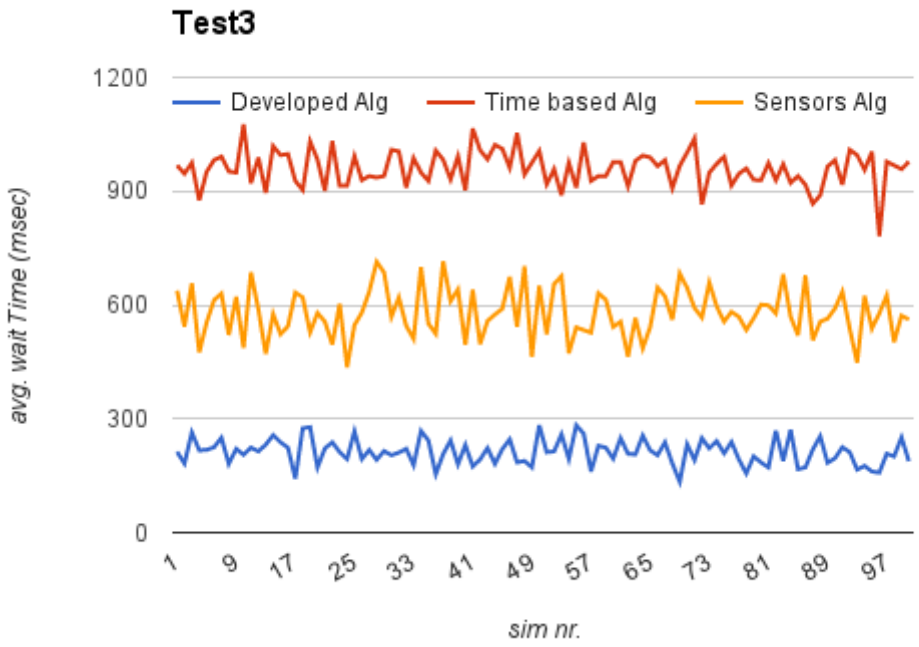


FIGURE 5.3: One traffic light, two big and two small roads, no turn lights and moderate and lopsided traffic flow.

It can clearly be seen that the average wait time is significantly lower with the developed algorithm when compared to the time based one and the sensory based algorithm. This indicates that the developed algorithm performs better in this scenario. The average wait times for the different algorithms in this scenario is 212.74 msec for the developed algorithm, 959.95 msec for the time based algorithm, and 578.88 msec for the sensor typed algorithm. The difference between the time based algorithm and the other two can primarily be accounted to the fact that the other two algorithms only changes the light to be green for the smaller roads when there is traffic and only for the short amount of time it would take the traffic to cross where as the time based one changes after a specified amount of time irrelevant of actual traffic. For more information on the specific test data and values look in appendix E.

5.1.4 Test 4

Test 4 looks at how the different algorithms perform when implemented in a traffic network with moderate traffic. It was selected to use a traffic network from the real world as the base for this test so as to better relate to the results of the test. The selected traffic network is from around Glostrup, Denmark. The network spans from the intersection O3 (Nordre Ringvej) - 156 (Hovedvejen) to 156 (Hovedvejen) - Nørre Alle, and some of the roads and intersections around the area. The traffic network is better described by the figure below and the test results by the graph. For additional information about the traffic network and test data please refer to appendix F.

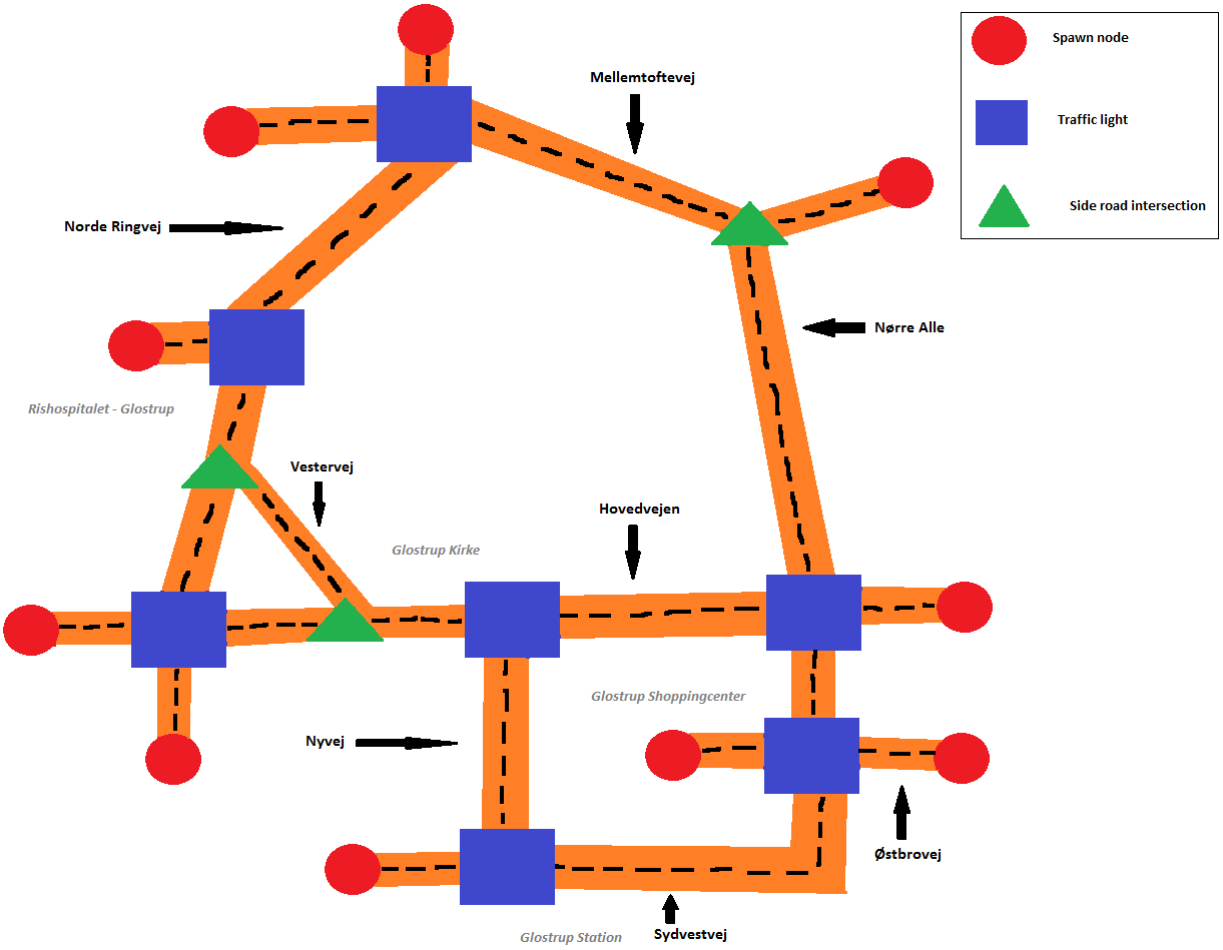


FIGURE 5.4: Traffic network used for test 4

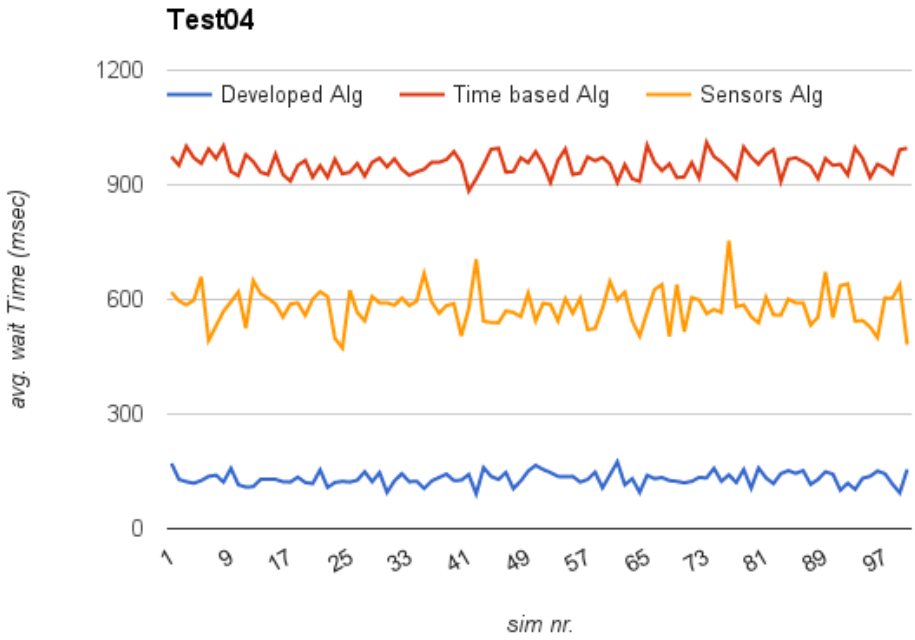


FIGURE 5.5: A network of multiple interconnected traffic lights of different types

The graph shows the overall average wait time for the whole traffic network per simulation for each algorithm. By looking at the resulting data it can be concluded that the developed algorithm far out performed both the time based algorithm which performed the worst and the sensor based algorithm which is in the middle. The total average wait time for the entire test per algorithm is 131,03 msec for the developed algorithm, 953.10 msec for the time based algorithm, and 580.64 msec for the sensor based algorithm.

5.2 Discussion

Let us now reflect on these test results and compare them to the project scope.

The shared purpose of the four tests above, was to check the developed algorithm against two more common traffic control algorithms, to see if the improvement of the average wait time for each car/user was significant. The hypothesis was tested in three different scenarios for a single traffic light, and then once for an elaborate network of traffic lights.

In the first test, the hypothesis was tested using a standard traffic light layout with four connected roads of equal size (throughput), no explicit turning lights and a moderate traffic flow from each direction. The purpose of the test was to measure the algorithms against each other, to ultimately test our hypothesis against a common traffic light layout. Just as expected, our developed algorithm showed to be significantly better at handling traffic flow, than the algorithm utilising only fixed timers and slightly (yet consistently

and significantly) better than the sensor-utilising one. The sensor-utilising algorithm showed to be an improvement over fixed timers, which makes sense as the sensor algorithm is basically an upgrade from the the time based algorithm.

In the second test, the hypothesis was then tested using the same layout of the traffic light, but this time with an extreme amount of traffic. The purpose of this test, was to show that even though our developed algorithm might be a big improvement over the other two, it does not really matter when dealing with a scenario where every single lane is crowded with cars. When there is a continuous stream of traffic in each direction, the light configuration does not matter, as the same amount of cars would go through regardless of which direction was open. It is also due to a scenario like this, that our developed algorithm tries to limit the average wait time. Had it tried to get as many cars through as possible in a scenario like this, the light would never change unless there was a defined maximum for how long one light configuration can remain unchanged. As expected, all three algorithms perform very similarly, as there is no real benefit to having increase knowledge about the traffic flow in a situation like this.

In the third test. The hypothesis was tested in a realistic situation different from the first two. The purpose of this test was to further confirm the improved efficiency from implementation of our developed algorithm compared to either sensors and/or fixed timers. The aim of the test was to simulate a common traffic layout, where a relatively small road intersects with a larger one. Our developed algorithm did not see much of a change in superiority over the time based one compared to the first test, but a slight increase in improvement over the sensor-based one. The test thereby confirmed the result that we had gotten previously. An interesting added result of this test, was that the sensor-utilising algorithm showing an even more significant (very slight but consistent) improvement over the one only using fixed timers compared to the first test. This is likely due to the nature of the traffic layout, as fixed timers tend to fall short, when the two intersecting roads are of different size - thus throughput. Fixed timers are more comfortable when each road has the same more or less constant throughout - which was not the case in this example.

As each test has its own specific purpose and interesting result alongside, the fourth test was arguably the most interesting, as it tested the hypothesis not just for a single traffic light, but rather an elaborate network of different types of traffic lights. The average wait time in this scenario is a result of the average wait time for each traffic light (i.e. an average of the averages), which is also reflected in the graph, as each curve is slightly more consistent compared to the previous tests.

As mentioned previously, the network itself was based on a real world example, however the actual traffic data was simulated completely. As we have been unable to acquire any form of traffic data, the data used in the test have been chosen carefully based on general knowledge about the area in the real world and general logic, so as to get as close to a real life case study as possible. It is however important to note that the average wait times are not perfect

reflections of wait times in the real world, as countless realism-factors could have been added - such as yellow signal light, gradual acceleration from the cars etc.. These are not implemented in the simulator since they do not add to the accuracy of the tests, which is the point of the simulation; adding a yellow light for instance, would increase the average wait times, but that would be the case for all three algorithms, so the difference would barely be noticeable.

Overall the tests that were performed confirms our hypothesis by showing a significant improvement in our developed algorithm over the more commonly used ones. This ultimately proves that traffic lights that implements an algorithm that utilises information shared between multiple independent traffic lights, significantly improves the overall traffic flow. The proof is based on the fact that the relevant tests show a decrease in average wait time consistently over the 100 times we ran each test. Test 2 is the only one that does not show a significantly consistent change in the traffic flow, but as mentioned above, this was to be expected from the test scenario, which purpose was to simply confirm that expectation.

It is worth noting that the addition of various realism-factors probably would have given a different result - possibly even a less significant difference. However these factors generally does not affect the outcome of the scenarios that much and due to the massive improvement to efficiency in the traffic networks seen above, a few minor changes would likely not have changed the results in a degree that would have resulted in a different conclusion.

5.3 Potential Project Extensions

This project was done with potential extensions in mind. Both the traffic simulation model and the traffic control algorithm could potentially be development upon even further, to improve realism, accuracy, efficiency and performance. We are now (in no particular order) going to list the subjects which could be improved upon.

Further reach
In the current version, each traffic light is only capable of communicating with its immediate neighbours. As discussed in the section regarding communication, this might limit the efficiency, as the ability to gather information from nodes beyond the first level of neighbours could improve the overall efficiency of the control algorithm. The ability to utilize data from beyond the first level of connections, would however require some measures of probability and statistics to estimate the incoming traffic based on how much incoming traffic the node outputting the traffic was experiencing.

Imagine a network of three traffic lights in a row - let us call them A, B and C respectively. A experiences a flow of traffic going in the direction of B. Rather than just alerting B about the incoming traffic, A also alerts C that B is receiving an incoming flow of traffic. C now knows that some traffic might be coming in the near future, but it has no way to be sure how much of it

will actually turn up when the time comes. C then has to make a qualified guess based on empirical data from previous experience, a predefined static probability, or complete randomness. While this decreases the accuracy of the system, it has the potential to increase the efficiency, if used correctly, as each traffic light controller, gets more time to evaluate more data.

More user types
Currently the model only supports cars (aka. the object Automobile) as users in the system. Possible extensions to the system, could be the support of more types of users - as in users with different types; different attributes, priority, size etc.. Examples of other user types could be pedestrians, trucks, buses, emergency vehicles and bikes. While the latter four example types would probably just alter the simple values, like priority and size, a new user type like pedestrians would require a bit more implementation to completely support. Support for something like pedestrians, would probably involve a new link object in the model (acting as a sidewalk), giving each four-way traffic light an additional four-eight inputs and outputs depending on the defined traffic rules. Regardless of the design, this would greatly complicate the scenarios, thus the traffic control algorithm.

Yellow Light
Right now the traffic lights only has two states; red and green. When the light is red the cars are not allowed to traverse the intersection, and has to wait in a line, until the light turns green, at which point they are encouraged to drive. A possible addition to the state machine, could be a yellow light; as a state in between the red and green. The light would then change from green to yellow and only then to red. Likewise the light would change from red to both red and yellow at the same time (to differentiate from the pre-red light) and only then to green. This would improve the realism factor, as changing the light would become slightly less desirable for the algorithm. It does not make a very noticeable difference however, all used algorithms would be affected the same way, with the same factor - the only difference would be an increased overall clear time - thus average wait time. In the real world, the yellow light is mostly present due to increased safety and general quality of life in the form of additional information for the users - both things that has no real impact in a simulation, which purpose is to test different traffic light algorithms.

Model diversity
In the current version of the system, only the most basic forms of nodes and links are supported, to make it possible to test algorithms on simple (yet somewhat realistic) traffic networks. A possible extension to the model, could be to add support for more types of both nodes and links in the model. Examples of more types of nodes, could be five-way intersections and roundabouts with traffic lights. Examples of more types of links, could be highways and bus-lanes. Generally additions of objects to the model would increase its

diversity and allow for more complex traffic networks.

Priority
Currently each car in the network has the same priority (i.e. none). This means that all cars are valued equally and decisions on whether to change the light or not is made without any differentiation between users. A possible extension to the system, could be a priority attribute on each car; either a fixed one, or a dynamic one. Imagine if each vehicle was able to send data to a traffic light (possibly using a VANET or InVANET), including relevant information about the content of the vehicle. This could for instance include the number of passengers, or if it was transporting a pregnant woman. The controller would then give the vehicle a priority based on the data relative to the other vehicles in the network, making vehicles with a higher priority weigh heavier when deciding which lane to clear next. Additionally this could be combined with more user types; where emergency vehicles had a higher priority than buses that in turn had a higher priority than cars for instance.

Global Positioning System
Right now cars are detected when they enter the network by the sensors most traffic lights in urban areas use in the real world today. The position of a car is known by the system relative to a traffic light. To make the positioning more precise, one could utilize the GPS that most people have active either on their smartphone or build into their car. This would give the traffic controllers more detailed information about each car's position in real time, ultimately improving the accuracy, thus the efficiency of the system.

Likewise tools like Google Maps and Bing Maps' API's for traffic coverage and analytics could potentially be utilised to gather additional information about general traffic flow.

Communicating cars
As of right now, the only way to determine the current direction of a car, is by reading the output from another traffic light in the system that is an immediate neighbour. Consider a scenario where each car was able to communicate with the traffic lights (e.g. using VANETs or InVANETs). The cars would then be able to tell the entire system where it was going from the entering of the traffic network, and the system would then be able to plan the entire route from start to finish. This would greatly improve the efficiency of the system, as each traffic light in theory would know when to expect any car in the network. It would however require that the cars supported this feature. Combined with cooperation with the navigation system of the car or a passenger's smartphone, this could mean greatly improved traffic flow throughout the network, as cars could be routed based on the current traffic situation and traffic lights would know the entire traffic situation throughout the network in real time.

6 Conclusion

The result of this project is a traffic simulator that can be used to test different traffic control algorithms, and an algorithm that is created with the idea of data communication between traffic lights in mind. The algorithm serves to prove that utilising traffic information from neighbouring traffic lights can significantly improve the overall flow in a traffic network, whereas the sole purpose of the simulator itself is to test different algorithms against each other, using a model that supports registration of traffic flow.

If we recall the scope of the project from the introduction, the goal of this project was to prove that traffic lights implementing an algorithm that uses traffic information gathered from neighbouring traffic lights, would experience an improved traffic flow - specifically classified as the average wait time for each user.

By performing multiple tests on different scenarios, that all consistently confirmed our hypothesis by a significant amount, we are hereby comfortable concluding, that by utilising information about the traffic flow from neighbouring traffic lights in a network using our algorithm, the overall traffic flow of the network is improved, by decreasing the average wait time for users. As a result of decreased average wait times, the users of the network would experience an increase in the quality of life. As a consequence of a more efficient traffic flow, users might experience shorter travel times, less fuel consumption (thus cost), less pollution, better road safety, better response times for emergency services and generally a better driving and commuting experience.

Working on this project has been an interesting and educational journey. We both share an interest for traffic management, and working with the improvement thereof has been rather interesting. The design and implementation of both the traffic simulator and the algorithms (especially our own) has also proven to be relatively exciting. The model of the traffic simulation is the perfect subject of an object-oriented approach, of which our background and general knowledge as software developers is based, and programming in Java is something we enjoy more so than in any other programming language. The development of the algorithm proved to be more challenging and entertaining than we had initially expected, but by working together and utilising the knowledge and experience we have from our time under education, we managed to end up with a solution that satisfy our goals and that we are proud of.

All in all we believe that we have provided a solution to the problem at hand, that satisfy the goals we set at the beginning of the project.

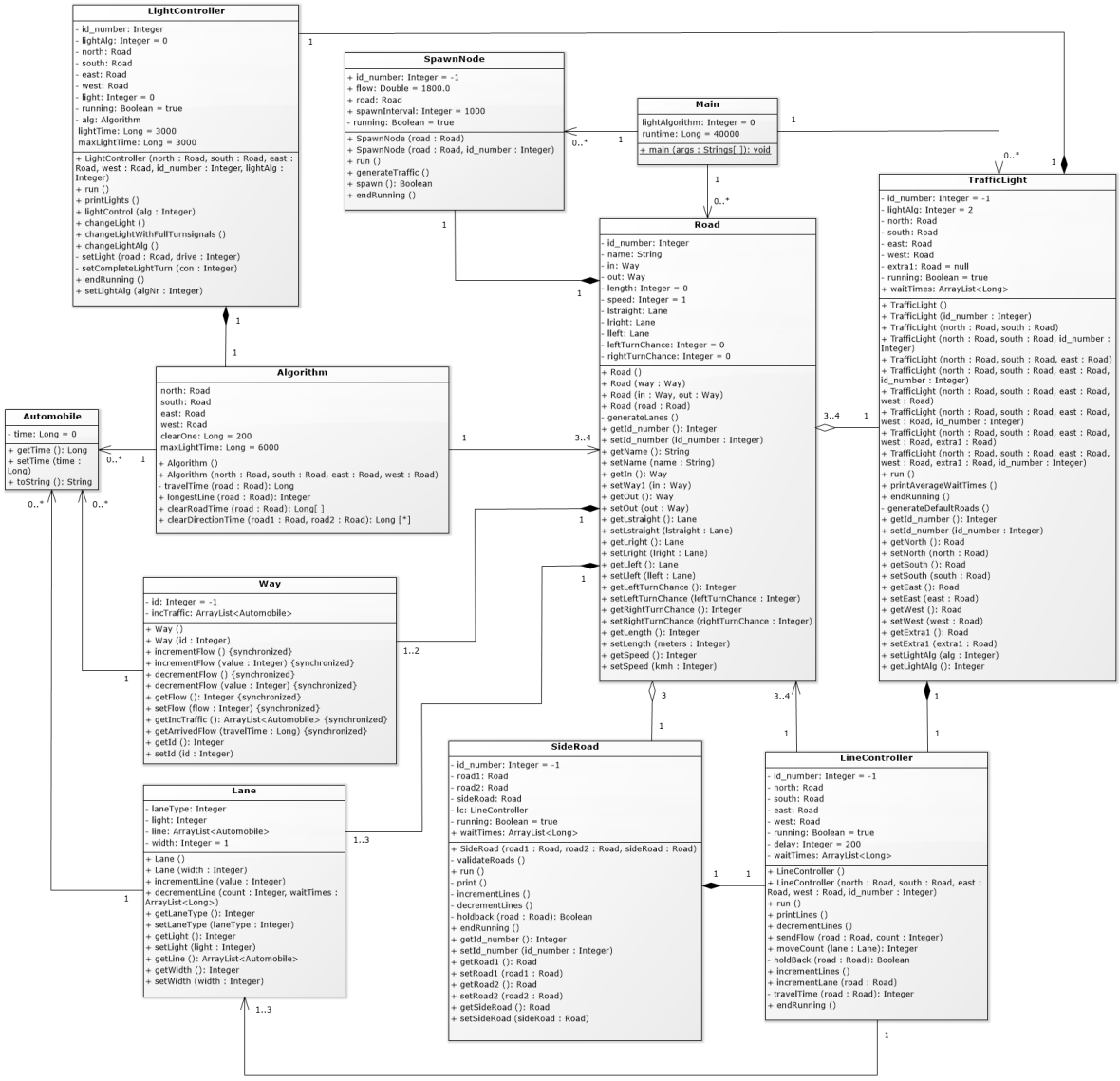
7 References

- http://www.movsim.org
- http://www.opentrafficsim.org
- http://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931_read-41000/
- https://en.wikipedia.org/wiki/Vehicular_ad_hoc_network
- https://en.wikipedia.org/wiki/Urban_Traffic_Management_and_Control

Other than the references above and our general competences, we have drawn inspiration from course material from various related courses offered at DTU that we have both previously attended. This includes text and figures from lecture slides as well as course nodes, all of which has been gained through the filesharing of https://www.campusnet.dtu.dk.

8 Appendix

A - Design Class Diagram (traffic simulator)



B - Probability calculations

Calculation for probability of at least one traffic spawn in one second when spawn interval is 200 msec, spawn value is 1800, and max spawn value is 3600: 1-(17993599)50.9688 = 96.88 %

Calculation for the probability of five traffic spawns in one second when spawn interval is 200 msec, spawn value is 1800, and max spawn value is 3600. (A spawn at each interval): (18003599)50.0313=3.13%

C - Test 1 info and data

Info				
spawnNode	Spawn Interval (msec)		Run Time (msec)	SimTimes
snn	210		60000	100
sns	220			
sne	200			
snw	200			
road	length (m)	speed (km/h)	L-turn chance (%)	R-turn chance (%)
n	100	80	20	20
s	100	110	20	10
e	30	40	40	4
w	50	80	25	20
Algorithm	lighttime (msec)	maxlighttime (msec)		
our alg	changes	3000		
time based alg	3000	3000		

Data:

sim Nr	Developed Alg	Time based Alg	Sensors Alg
1	368	1245	752
2	437	1245	625
3	448	1319	594
4	538	1338	718
5	839	1196	837
6	521	1174	659
7	476	1152	772
8	555	1137	710
9	403	1236	627
10	418	1221	734
11	423	1138	724

12	534	1219	590
13	530	1074	716
14	532	1245	1051
15	569	1206	627
16	581	1113	545
17	387	1252	739
18	438	1218	585
19	431	1404	825
20	363	1175	624
21	743	1161	801
22	540	1206	806
23	439	1257	736
24	398	1254	679
25	477	1118	628
26	519	1417	755
27	525	1183	735
28	439	1107	747
29	570	1376	627
30	473	1229	772
31	615	1239	650
32	664	1169	743
33	571	1354	592
34	471	1173	695
35	547	1335	801
36	324	1276	736
37	476	1173	664
38	456	1183	764
39	404	1299	829
40	535	1161	711
41	642	1145	914
42	536	1111	711
43	753	1127	941
44	510	1178	966
45	358	1306	612

46	392	1175	632
47	494	1249	565
48	529	1418	615
49	371	1182	653
50	572	1277	800
51	473	1153	740
52	675	1299	763
53	459	1265	825
54	458	1137	6[illegible]
55	421	1175	682
56	416	1114	726
57	348	1255	715
58	451	1237	586
59	539	1232	677
60	447	1441	645
61	393	1313	788
62	463	1432	706
63	497	1345	628
64	400	1370	877
65	537	1199	656
66	608	1228	903
67	470	1209	839
68	806	1289	556
69	488	1248	564
70	385	1104	667
71	407	1099	727
72	512	1161	624
73	477	1184	845
74	405	1158	862
75	391	1372	772
76	314	1177	674
77	575	1318	924
78	376	1102	843
79	492	1221	795

80	519	1127	680
81	318	1099	639
82	506	1209	624
83	475	1246	609
84	316	1230	789
85	431	1282	830
86	456	1187	602
87	624	1300	659
88	567	1148	719
89	533	1175	862
90	833	1226	745
91	561	1449	570
92	453	1196	631
93	461	1156	660
94	408	1280	825
95	478	1225	891
96	530	1273	719
97	661	1179	738
98	629	1212	686
99	469	1446	774
100	538	1083	881
Overall Average	495.83	1227.1	723.6

D - Test 2 info and data

Info				
spawnNode	spawnInterval		run time (msec)	sim Times
snn	105		60000	100
sns	110			
sne	100			
snw	100			
road	length (m)	speed (km/h)	L-turn chance (%)	R-turn chance (%)
n	100	80	20	20
s	100	110	20	10
e	30	40	40	4
w	50	80	25	20
Algorithm	lighttime (msec)	maxlighttime (msec)		
our alg	changes	3000		
time based alg	3000	3000		

Data:

sim Nr	Developed Alg	Time based Alg	Sensors Alg
1	3883	4294	3934
2	3748	3961	5069
3	4815	4000	4786
4	3955	3557	3141
5	3766	4407	5427
6	4772	4535	4378
7	4607	4117	3622
8	4099	4393	3634
9	3843	3821	5144

10	3890	4416	4303
11	3233	4826	3283
12	4613	4067	3491
13	4352	4292	4563
14	4886	4131	4947
15	4329	3822	3882
16	4228	4381	4955
17	4302	4490	4165
18	4090	3937	4427
19	4720	3528	4055
20	4527	3694	4679
21	4602	3529	3561
22	4106	4609	5768
23	4134	3712	3634
24	3328	4609	3985
25	4168	5383	5659
26	4242	3324	4199
27	4579	3812	4325
28	4021	3351	4294
29	5416	4308	4826
30	4433	4106	4654
31	4078	4107	4098
32	4202	4379	3518
33	3467	3901	3942
34	3958	4559	4047
35	3632	4139	4083
36	3635	4032	4934
37	4460	5127	4242
38	3985	5035	4053
39	3928	3436	4421
40	5249	4518	4663
41	4435	3699	4499
42	3978	4852	4555
43	4407	3162	5396

44	3269	4872	4353
45	4592	4369	4894
46	4025	4286	3170
47	4789	4283	4150
48	5056	5323	3660
49	4001	3748	4024
50	3865	3189	4720
51	3172	3522	4173
52	3542	3982	3823
53	3693	4982	3850
54	3987	4053	4984
55	4058	4035	3330
56	4468	5039	4347
57	4593	4149	3794
58	5166	4222	3851
59	3605	3776	4724
60	3460	4508	4069
61	3425	3798	2907
62	3375	5534	4544
63	3971	4378	4203
64	4558	3953	3968
65	4569	3463	3710
66	4218	3970	4285
67	3859	4165	3813
68	5116	3740	5146
69	3925	3865	3681
70	4482	3996	5020
71	5499	3826	3747
72	3961	3500	4307
73	4769	4055	4726
74	4880	4999	3470
75	3637	3879	4091
76	4181	3976	4569
77	4056	4114	4265

78	4861	3944	3629
79	3921	4062	4023
80	5717	4082	4933
81	4433	4641	4597
82	4127	4096	5085
83	3977	4424	4449
84	4469	4032	3833
85	3342	3721	4389
86	4431	3828	5248
87	3596	4244	4357
88	4031	5007	4632
89	4221	4221	4576
90	4143	4928	4172
91	4396	4250	4045
92	4702	3831	3643
93	4369	3802	3997
94	4656	4567	3829
95	3995	3924	4763
96	3534	4439	3976
97	3977	3683	4233
98	3398	4028	4145
99	4139	4030	3766
100	4104	4371	4462
Overall Average	4194.62	4160.62	4263.91

E - Test 3 info and data

Info				
spawnNode	spawnInterval		run time (msec)	sim Times
snn	160		60000	100
sns	170			
sne	400			
snw	380			
road	length (m)	speed (km/h)	L-turn chance (%)	R-turn chance (%)
n	100	80	10	10
s	100	110	10	10
e	30	40	35	45
w	50	80	45	45
road	S-lane width			
n	2			
s	2			
Algorithm	lighttime (msec)	maxlighttime (msec)		
our alg	changes	3000		
time based alg	3000	3000		

Data:

sim Nr	Developed Alg	Time based Alg	Sensors Alg
1	213	968	638
2	182	946	543
3	264	975	657
4	217	876	475
5	218	951	554
6	225	982	613
7	250	991	630
8	181	952	521

9	221	949	622
10	205	1076	487
11	224	921	686
12	214	989	589
13	232	896	471
14	257	1019	576
15	238	995	523
16	224	998	544
17	142	927	633
18	275	903	620
19	278	1031	528
20	172	983	579
21	222	901	556
22	238	1032	496
23	212	914	603
24	194	914	436
25	267	992	547
26	194	929	580
27	218	940	635
28	192	936	714
29	215	940	686
30	204	1009	569
31	211	1005	620
32	221	909	545
33	178	987	511
34	268	949	700
35	243	927	551
36	154	1007	524
37	207	983	715
38	243	931	610
39	180	992	641
40	231	902	495
41	174	1065	640
42	193	1008	497

43	223	984	558
44	181	1022	575
45	220	1012	590
46	246	962	674
47	185	1053	542
48	189	944	702
49	172	975	463
50	283	1006	651
51	212	918	523
52	214	958	655
53	259	889	676
54	194	975	473
55	283	909	541
56	261	1029	534
57	161	927	527
58	229	940	631
59	224	940	614
60	195	977	542
61	249	977	556
62	208	913	464
63	206	980	566
64	256	994	486
65	217	989	542
66	204	966	646
67	238	982	623
68	182	908	561
69	134	967	683
70	233	1002	645
71	191	1039	592
72	249	865	567
73	222	949	659
74	241	972	596
75	209	991	556
76	239	915	582

77	193	946	569
78	156	960	534
79	201	930	565
80	185	929	601
81	172	974	599
82	269	929	578
83	189	971	681
84	271	922	569
85	167	940	520
86	172	917	678
87	219	867	507
88	254	889	556
89	184	965	564
90	196	982	591
91	225	917	635
92	212	1009	537
93	166	995	448
94	176	956	624
95	161	1004	538
96	158	782	578
97	208	978	625
98	201	968	502
99	251	958	572
100	188	978	562
Overall Average	212.74	959.95	578.88

F - Test 4 info and data

Network setup			
	Algorithm	lighttime (msec)	maxlighttime (msec)
	our alg	changes	3000
	time based alg	3000	3000

Nodes						
	spawnNodes nr.	node ID	spawnInterval	road		
	1	1	190	1		
	2	2	250	3		
	3	4	800	11		
	4	7	1000	8		
	5	9	150	19		
	6	14	150	32		
	7	15	190	22		
	8	16	1000	35		
	9	18	210	38		
	10	19	400	43		
	TrafficLight nr	node ID	road1(n)	road2(s)	road3(s)	road4(w)
	1	3	2	9	5	4
	2	5	10	13		12
	3	10	18	21	23	20
	4	12		41	27	26
	5	13	29	33	31	28
	6	17	34	39	37	36
	7	20	42		40	44
	SideRoads nr.	node ID	road1(n)	road2(s)	road3(e)	
	1	6	30	6	7	
	2	8	17	14	15	
	3	11	25	24	16	

links									
	Roads nr	road id	length (m)	speed (kph)	L-Turn Chance	R-Turn Chance	L-lanewidth	S-lanewidth	R-lanewidth
	1	1	100	70	0	0	1	1	1
	1	2	100	70	10	10	1	2	1
	2	3	100	60	0	0	1	1	1
	2	4	100	60	40	30	1	1	1
	3	5	200	50	40	30	1	1	1
	3	6	200	50	5	0	1	1	1
	4	7	385	70	60	40	1	1	1
	4	8	385	70	0	0	1	1	1
	5	9	50	40	20	10	1	2	1
	5	10	50	40	0	10	1	2	1
	6	11	50	50	0	0	1	1	1
	6	12	50	50	45	55	1	1	1
	7	13	400	70	20	0	1	2	1
	7	14	400	70	0	0	1	2	1
	8	15	720	50	0	100	1	1	1
	8	16	720	50	0	100	1	1	1
	9	17	350	70	0	5	1	2	1
	9	18	350	70	30	30	1	2	1
	10	19	330	50	0	0	1	1	1
	10	20	330	50	20	20	1	2	1
	11	21	100	70	30	30	1	2	1
	11	22	100	70	0	0	1	1	1
	12	23	165	50	20	20	1	2	1
	12	24	165	50	0	0	1	2	1
	13	25	100	70	0	5	1	2	1
	13	26	100	70	0	15	1	2	1
	14	27	275	50	10	0	1	2	1
	14	28	275	50	20	25	1	2	1
	15	29	320	50	40	40	1	1	1
	15	30	320	50	0	5	1	1	1

	16	31	100	50	15	10	1	2	1
	16	32	100	50	0	0	1	1	1
	17	33	250	50	40	45	1	1	1
	17	34	250	50	50	30	2	1	1
	18	35	20	30	0	0	1	1	1
	18	36	20	30	50	15	1	1	1
	19	37	160	50	10	50	1	1	1
	19	38	160	50	0	0	1	1	1
	20	39	100	60	20	40	1	1	1
	20	40	100	60	0	40	1	1	1
	21	41	360	50	60	40	1	1	1
	21	42	360	50	70	30	1	1	1
	22	43	50	40	0	0	1	1	1
	22	44	50	40	40	0	1	1	1

sim Nr	Developed Alg	Time based Alg	Sensors Alg
1	171	974	619
2	129	951	596
3	123	1001	586
4	119	971	598
5	126	956	659
6	137	994	493
7	140	969	531
8	122	1002	569
9	158	935	594
10	115	924	620
11	109	979	525
12	110	961	649
13	129	933	615
14	129	927	603
15	129	980	588

16	123	927	554
17	122	910	588
18	135	951	591
19	121	964	558
20	118	920	601
21	154	950	620
22	108	920	608
23	121	967	498
24	124	929	473
25	122	933	624
26	127	956	566
27	149	923	544
28	123	959	608
29	147	970	591
30	95	947	591
31	125	968	585
32	144	941	604
33	123	925	584
34	124	934	596
35	105	941	668
36	125	959	594
37	134	959	564
38	143	966	584
39	126	987	589
40	127	958	505
41	142	884	576
42	91	916	705
43	160	952	543
44	137	993	540
45	129	996	539
46	147	933	570
47	105	935	566
48	125	971	555
49	151	958	617

50	166	987	544
51	155	954	590
52	147	906	587
53	137	964	544
54	136	994	602
55	137	927	564
56	122	931	60
57	129	973	521
58	148	963	524
59	107	972	577
60	141	955	647
61	175	906	599
62	115	953	619
63	131	916	544
64	95	909	504
65	140	1004	567
66	131	959	626
67	134	937	639
68	126	955	503
69	124	919	639
70	120	921	516
71	124	958	605
72	134	918	599
73	133	1011	563
74	158	974	574
75	124	960	566
76	141	940	754
77	121	916	581
78	155	999	586
79	106	973	555
80	159	954	539
81	133	979	606
82	118	992	560
83	144	909	559

84	152	967	601
85	145	971	591
86	152	961	591
87	116	949	533
88	129	916	554
89	149	969	672
90	143	951	552
91	101	954	636
92	119	926	641
93	103	997	543
94	132	969	545
95	137	919	528
96	151	954	500
97	144	944	604
98	117	928	603
99	94	992	638
100	155	996	482
Overall Average	131.03	953.1	580.64