TU/e - Department of Mechanical Engineering
Sweco Nederland - Mobility and Infrastructure
De Bilt, February 2017

Anticipative flexible traffic light controller design and network stability

Supervisor: prof. Dr. H. Nijmeijer
Dr. Ir. A.A.J. Lefeber
Ir. B. van der Bijl
Authors: T.C.G. van Hout idnr. 0725389
Document nummer DC 2017.033

Eindhoven University of Technology
Dynamics & Control

Abstract

Traffic lights are used to regulate traffic flows within towns and cities and improve traffic safety across intersections. Controlling these lights is done by using sensors, which provide the position of traffic. Each direction of an intersection receives green until the queue has dissolved or the maximum green duration for the lane is reached, after which the next direction is served. However, due to increased detecting methods and connectivity much more information is available about the traffic situation. At Sweco a project is underway which, by detecting the current traffic situation, is also predicting the future traffic situation. A microscopic simulation which tracks every car is able to measure the effect traffic lights have on the throughput and performance of the traffic. By choosing the traffic light settings providing the least amount of disturbance, traffic flows faster across an intersection or even throughout an entire city. Directing the traffic flows to such detail requires a better control strategy for the traffic lights. This better, or smarter, control and the predicting microscopic simulation leads to the development of Smart Traffic. In this thesis the Smart Traffic approach is explained and tested. The goal of this thesis is to determine the performance of Smart Traffic opposed to current traffic light controllers. The performance is tested on a single intersection as well as on a network of intersections. In a network, traffic flows to and from intersections causing direct feedback between them. This feedback may cause undesirable behaviour causing too much or too little traffic to arrive at an intersection. To test Smart Traffic a model is created which simulates traffic dynamics around traffic lights. By applying Smart Traffic onto a local intersection and testing it under different traffic intensities its performance is tested. The Smart Traffic control strategy can cause the waiting time to decrease by 30% as opposed to current traffic light controllers. Additionally, cars that arrive close to one another in so called platoons, receive a green light more often eliminating their waiting time altogether. A network is controlled by multiple Smart Traffic instances, sending and receiving traffic from each other. A single intersection does not know what its neighbours are doing. It is shown that this may lead to one intersection having no traffic to process while the other intersection has too much traffic. This in turn leads to the amount of cars to start increasing as well as the waiting time. A supervisor is proposed which detects the amount of work being processed by the network. This is a measure for the efficiency of the entire network; the higher the amount of work being processed the faster cars can leave. When the supervisor detects a decrease in efficiency it overwrites the local traffic light controllers. It redistributes the cars across the network to ensure all intersections are efficient. Therefore, the supervisor prevents undesired behaviour while Smart Traffic minimizes the waiting time of the person on the road.

List of Symbols

μ_i	Maximal departure rate for lane i, also called saturation flow.
λ_i	Mean arrival rate for lane i.
ρ	Utilization, fraction of time a lane minimally needs to be served by a green light.
$x_i(t)$	Number of cars waiting in queue, buffer size.
σ_{ij}	Clearance time from lane i to j. During this time both lanes experience a red light.
τ_{ij}	Switching time losses between lane i and j. Consisting of σ_{ij}, driver reaction speed and acceleration loss
V	Maximum driving speed.
s	length of lane.
$Q_j^i(t)$	Stochastic flow of individual cars over time. Each lane j has an arriving and departing flow which consist of individual cars.
$wt_i(t)$	Total waiting time over all cars in lane i, integral of $x_i(t)$.
$g_i(t)$	Duration of green phase for lane i until queue is cleared.
$\hat{g}_i(t)$	Total duration of a green phase of lane i.
$u_i(t)$	Process rate of lane i.
$w(t)$	Amount of work of the system. The total remaining processing time of all jobs (cars) in the system.

List of Abbreviations

MPC	Model Predictive Control
FTS	Fixed Time Schedule
VAC	Vehicle Actuated Control
VISSIM	Visual Simulation
FIFO	First in First out

Table of Contents

Chapter 1

Introduction

According to a recent study [1] within five years the Dutch cities suffer from complete grid lock. The maximum capacity of the road network is reached while the amount of traffic keeps increasing [2]. The Dutch government launched several projects to keep the cities accessible; from better public transport to flexible working hours as to avoid rush-hours. Amongst the possible solutions is the option to make better, or smarter, use of the current road network. A common sight in cities are the long queues for traffic lights. Cars are often waiting to cross one intersection just to join in the queue for the next intersection. In general all traffic lights are controlled in a similar way; forcing the traffic to adapt, going when the lights are green and waiting when red. At Sweco a project is underway that does just the opposite: Traffic lights adapt to the current traffic situation. This new traffic light control is called Smart Traffic and by using realtime data it aims to decrease waiting time for drivers and increase the throughput of the road network.

Traffic lights have existed since the end of the 19th century and regulate the upcoming traffic, be it cars, pedestrians, boats or trains, to ensure safe crossing. As traffic intensities continued to grow, the need for traffic light control manifested himself. An explanation of traffic light controllers currently employed is given in Chapter 2. Due to the increased connectivity of the traffic (road-sensors, camera's, GPS, Smart phones) more input is available for a traffic light controller. This increased connectivity combined with increased computing power means a traffic light controller can handle more complex algorithms to decide which direction to serve. These algorithms work as an optimization strategy to minimize an objective function. The objective function of the traffic light controller Smart Traffic aims to decrease delays. By detecting and responding to changing arrivals during the day, see Figure 1.1, Smart Traffic aims to find the best available light control scheme for each situation allowing road users to travel as quickly as possible.

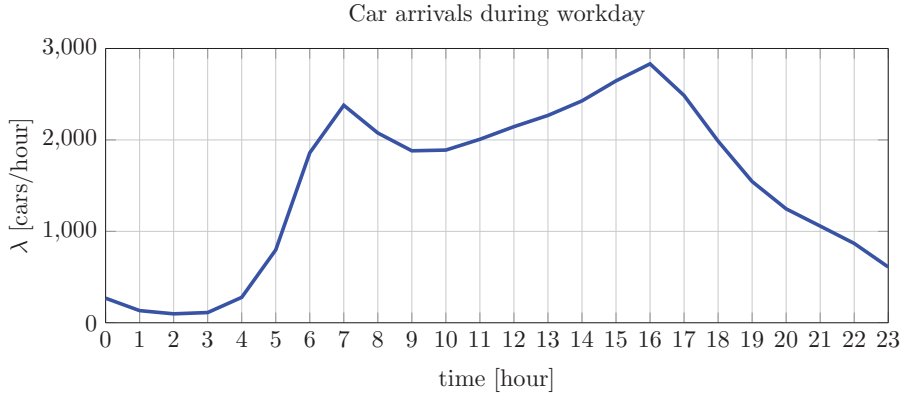

Figure 1.1: Average arrivals during a workday for the Insulindelaan in Eindhoven over the period 13-07-2006 till 31-12-2006.

This research is done in cooperation with Sweco, an international engineering consultancy agency which originates from Sweden. In 2016 it entered a partnership with the Dutch engineering consultancy bureau Grontmij B.V. Focusing on mobility, area development and energy it is a multidisciplinair company that works closely with local and national governments. The division hosting this project is Transportation and Mobility, boosting plenty of in-house knowledge and experience when

it comes to traffic light control as it has been one of their expertises over the past decade. Historically a consultancy agency, the development of Smart Traffic is a new and unknown step for the company. To support this new step they turn to universities to provide innovative knowledge and tools which assist their development.

We are interested in the performance of Smart Traffic in comparison with current available traffic light controllers, as the control strategy applied by Smart Traffic is a new approach for traffic management. To test the performance of Smart Traffic we need a model which: handles traffic behaviour around intersections, measures the performance of individual cars and is able to execute Smart Traffic as well as current traffic light controllers. The simulated driving behaviour is identical over all traffic light controllers allowing for a good comparison between the different traffic light controllers.

Another part of traffic management is controlling networks of intersections to ensure throughput across the network. For example traffic lights leading to and from a highway should never lead to a queue that spreads across the highway (backlash). Different traffic light scenarios are activated across the network to ensure backlash does not happen, with different scenarios for morning or evening rush-hours, events or other situations. These scenarios often prioritize a direction to ensure a certain level of throughput. In contrast, Smart Traffic explores high flexibility per intersection with each intersection being able to perform its optimal traffic light sequence. Applying a scenario would limit this flexibility and thus decrease performance. However, to prevent backlash or other undesirable behaviour some network control is needed.

We research the Kumar-Seidman Network [3], which is known to exhibit undesirable behaviour, and apply Smart Traffic to it as traffic light controller. The resulting instable behaviour is explained. By defining what causes the instability, additional constraints are created. These constraints serve to detect upcoming undesirable behaviour. Additionally, a solution is presented to prevent the instability from occurring.

The goal of the research is twofold:

- Determine and compare the performance of the proposed Smart Traffic control to current available traffic light controllers.
- Design and apply a controller that stabilizes the performance of networks of intersections controlled by Smart Traffic.

The first research question can be answered by creating a traffic model that simulates traffic behaviour around traffic lights and is able to execute multiple traffic light controllers. This model is expanded to simulate multiple, interlinked intersections to recreate a network providing information about what causes instable or undesired behaviour. Based on this information a network controller is designed and integrated into the model to create stability.

The structure of the report is as follows: in the Chapter 2 background information is provided about current traffic light control and the outline of Smart Traffic is shown. An overview is given about the basic terminology used throughout this report by using simple intersection as an example. Chapter 3 focusses on the model that is created to validate the working of Smart Traffic. The assumptions made in the model are explained as well as the influence of these assumptions between model and the real world. The model is thereafter validated to determine its performance by comparing it to VISSIM and theoretical approaches. When the model is a good representation of the real world it is used to test Smart Traffic. This testing is done in Chapter 4, providing in-depth information about Smart Traffic. It gives a step-by-step approach of the optimization that Smart Traffic performs and its use under different conditions is explained. It is tested and its performance is measured on single intersections and networks of intersections. This provides the first results of Smart Traffic for different traffic intensities. In Chapter 5 it is shown that the performance in a network is not always optimal, a method is defined to predict these shortcomings. By knowing in which cases a local controlled network fails we can act in advance to prevent these cases altogether. In Chapter 6 this method is implemented and executed to prove its viability and results are shown. Finally, conclusions and recommendations are presented.

Chapter 2

Background

This chapter describes and explains the basic methodology used throughout this report. It explains the basic notations and shows the currently traffic light controllers that are in use. Additionally it explains the general working of Smart Traffic and shows the key differences between the currently available traffic light controllers. Finally the challenges are shown of traffic light control in a network.

2.1 System description

Consider a simple three-way intersection as given in Figure 2.1

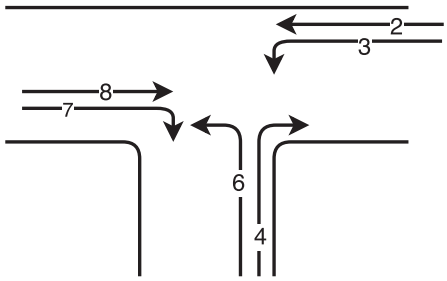

Figure 2.1: T-junction with traffic flow and numbering.

During the day traffic arrives with a certain intensity at flow f with arrival rate $\lambda_f(t)$. Arrival rates are expressed in car/hours and strongly fluctuate during the day as could be seen in Figure 1.1, individual arrivals typically occur through a Poisson distribution.

Arrivals form a queue x_f which gathers at the stopline. Departures from the intersection occur with the saturation rate μ_f as long as a queue x_f is available. The saturation rate is the maximum rate at which cars can leave a lane, approximately around 1900 cars/hour. Depending on the state of the traffic light and the queue size there are generic expressions which describe the evolution of the queue, $\dot{x}_f$

Table 2.1: Generic expression of the queue-evolution under different traffic light settings

Traffic light	Queue	Queue-evolution	Departure rate
Green	$x > 0$	$\dot{x} = \lambda(t) - \mu$	μ
Green	$x = 0$	$\dot{x} = 0$	$\lambda(t)$
Red		$\dot{x} = \lambda$	0

Amber times are not displayed in Table 2.1 as individual drivers can decide to drive through amber (so it acts as a green light) or stop (red light). For any lane to be able to handle the arriving traffic it needs to hold that $\lambda_f < \mu_f$, expressed in the utilization ρ_f

$$\rho_f = \frac{\lambda_f}{\mu_f} < 1 \tag{2.1}$$

Considering Table 2.1 and (2.1) the higher the utilization the more time is being spent in a green phase to ensure the queue completely dissolve.

As the arrival rate $\lambda(t)$ is a distribution it can temporarily occur that $\lambda_f(t) \geq \mu_f$. As this would mean the lane is overflowing traffic is limited to arriving at the saturation flow. These cars create platoons until the arrival rate drops. Platoons are multiple cars which drive so close to one another they can be seen as a moving queue. This also implies that multiple departures from a queue always create a platoons for downstream intersections. We use this fact when considering networks of intersections.

An intersection always has one or more lanes which cross each other, These are called conflicts, if a lane is in conflict with another they can not receive green light at the same time. Looking at Figure 2.1 we can easily determine which lanes are in conflict and create the conflict graph of Figure 2.2. Each line indicates the lanes are in conflict with one another.

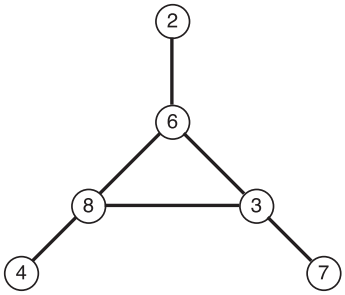

Figure 2.2: Conflict graph of T-junction of Figure 2.1

.

Just like a single lane needs to be able to handle the arriving traffic ($\rho_f < 1$) the same holds for a intersection. With conflicting traffic flows using the same spot (conflict area) the total utilization of this spot can never exceed 1. Considering the conflict graph of Figure 2.2 we can group the conflicts leading to the following constraints:

$$\begin{aligned} \rho_3 + \rho_6 + \rho_8 &< 1 \\ \rho_3 + \rho_7 &< 1 \\ \rho_4 + \rho_8 &< 1 \\ \rho_2 + \rho_6 &< 1 \end{aligned} \tag{2.2}$$

.

Additional constraints which cannot be directly derived from the conflict graph may exist depending on the lay-out of the intersection [4], for the used T-junction this is not the case.

A conflict between traffic flow i and j has a clearance time σ_{ij} associated with it. A clearance time is a period of time in which the traffic lights of both lanes are red. After flow i has been served and the amber time has elapsed it takes σ_{ij} before flow j can get served. This time is needed to allow the last remaining traffic which is still driving on the intersection to leave. Note that situations occur were $\sigma_{ij} \neq \sigma_{ji}$, depending on the layout of the intersection. Clearance times can also be zero or even negative (for example when the traffic lights are not placed directly at the intersection).

Any traffic light controller needs to assign green phases to each direction which in turn should be long enough to ensure the queue can dissolve. On the other hand the green phase should not be too large as due to conflicts as other directions can not be served in the meantime. Clearance times further decrease the effective time that can be spent serving traffic. To cope with these demands several traffic light controllers are in use and these are briefly explained in the upcoming sections.

2.2 Fixed Time Schedule

A Fixed Time Schedule (FTS) is designed to make the average waiting time for all traffic as low as possible. It assumes a constant λ_f and by respecting the conflicts between each lane the green time for each lane can be calculated. The order of switching is fixed as well as the duration of each green period. Green periods are long enough to clear the queue (with rate $\lambda_f - \mu_f$) that has been building up (with rate λ_f) during the red period. Additionally they also depend on the utilization of the total intersection. During a clearance period σ_{ij} the intersection is effectively idle, therefore green periods become larger to decrease the number of switches over time. By having large green periods the relative losses of the clearance time decreases.

A FTS is highly predictable and performs very well during peak hours. During peak hours the arrival rate is very high; there is always a car waiting at the traffic light and during the green period cars are served at saturation rate μ_f. However as seen in Figure 1.1 arrivals fluctuate from hour to hour (and also from minute to minute), the performance of FTS drops when the assumed arrival rate changes. As a FTS does not have any information about the actual traffic situation it does not change its behaviour.

The order and duration of green periods can be set by the user but can also be optimized. For this optimization a tool developed by S.T.G. Fleuren (promovendi TU/e) [5] [6] is used throughout this report. The resulting schedule of an FTS is displayed in a phase-diagram which shows the green duration of each lane. An arbitrary phase diagram of Figure 2.1 and with respect to the conflict graph of Figure 2.2 is shown below.

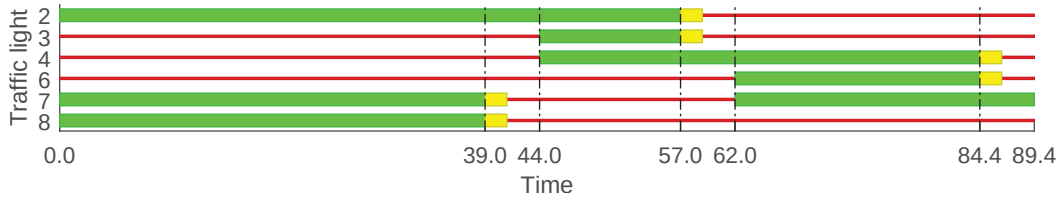

Figure 2.3: Fixed Time Schedule of T-junction of Figure 2.1.

The phase diagram shows the duration of each green phase per lane. At the end of the entire phase diagram it starts at zero again. The diagram clearly shows that lanes 2, 4 and 7 have long green phases. This is not because they have a lot of traffic that needs to be processed but because they have few conflicts (one each). They receive green as long as no conflicting lane is being processed. Using the same reasoning lanes 3, 6 and 8 receive much shorter green phases.

2.3 Vehicle Actuated Control

The Vehicle Actuated Control (VAC) is widely used in the Netherlands [7] [8] [9]. While the order of switching is (semi-)fixed the duration of each green period can fluctuate depending on traffic demand. An extensive explanation can be found in [9].

Contrary to a FTS the VAC does monitor and use the actual traffic situation, $\lambda_f(t)$. Sensors are located in the road that detect traffic driving overhead, at the very least a sensor is located at the stop line to detect if a car is waiting for the traffic light. Additional sensors are placed to detect if the queue has dissolved and/or to detect upcoming traffic at the intersection.

The green phase may end as soon as no more upcoming traffic is detected, or not start at all in case no traffic is available. Other conflicting directions can thus start their green phases earlier decreasing the waiting time of cars in that lane. Suppose the T-junction of Figure 2.1 and lanes 2, 7 and 8 are currently green. Lane 8 has served all available cars, by ending this green phase lane number 4 (see the conflict graph of Figure 2.2) start its green phase earlier. On the other hand if lane 7 or lane 2 have finished serving there is no benefit to end the green phase as all other directions are still in

conflict with lane 8. Additionally, each direction has a maximum green time to ensure green phases do end and other lanes can get served in a timely manner.

As the VAC utilizes the actual traffic situation and is much more flexible than a FTS its performance is a lot better when demand is low. Under high demand the VAC starts to act as a FTS since queues can not get cleared on time.

2.4 Smart Traffic

A complete explanation of the new traffic light controller Smart Traffic is given in Chapter 4, only the main differences between the current available traffic light controllers and Smart Traffic are given here.

Like the VAC, Smart Traffic takes the current traffic situation into account. Unlike the VAC, it does not only look if a queue has formed (to start a green phase) or if a queue has dissolved (to end the green phase) it also takes into account the size of the queue and the time each car has spent waiting. Which lane gets a green phase depends on the total waiting time of the traffic waiting. This way it ensures that every car gets processed (a car with long waiting time receives priority) and the incurred waiting time is more evenly distributed across the lanes.

Additionally, Smart Traffic also takes into account the upcoming traffic at the intersection. Departures from a upstream intersection (as seen by the sensors of that intersection) are sent towards the intersection controlled by Smart Traffic. As the travel time between the intersection can be approximated, Smart Traffic not only knows the current traffic situation of its intersection but also the traffic that is due to arrive in the future. As we know the current and future traffic situation, the effect of each green phase can also be predicted. Meaning that we know how long a green phase will take, how many cars pass the intersection during this phase and the time spent waiting for each of those cars. This also means that for every direction that does not receive a green phase, the increase in waiting time and number of cars is known. Thus, before any light is turned green its effect on the total intersection is already known. Comparing all green phases we pick the phase that leads to the biggest decrease in overall waiting time.

This approach is particularly useful in cities were a lot of sensor-data is available to create an accurate traffic situation. Additionally, traffic often flows between intersections controlled by traffic lights, when an upstream intersection sets a direction on green, the downstream intersection receives a packed amount of cars, so called platoons, after the travel time between intersections. These platoons can be seen as arrivals that occur close to the saturation rate, $\lambda_f(t) \approx \mu_f$. Outside of the platoons there are few arriving cars. Stopping a platoon leads to a quick increase in waiting time as multiple cars are involved. By adjusting the green phase in such a way that the currently waiting queue (if any) dissolves right before the platoons arrives, no car in the platoon has any waiting time. As Smart Traffic tries to minimize the overall waiting time it often chooses this option.

In contrast, the VAC does not make these decisions as it does not see the arriving cars in time to take them in account. Even if it does see platoons (by a sensor placed far from the intersection) it does not adapt its schedule to make the platoon pass without stopping. The Smart Traffic approach does not guarantee a fixed order of green phases and durations are exactly matched with the queue size. In practice this means that the last car passing in a green phase is driving underneath an orange light. For an extensive explanation see Chapter 4.

2.5 Network Control

Controlling a network of traffic lights is currently only done in a green-wave manner. An upstream intersection gives a green phase for a direction, after a delay (travel time between intersections) the downstream intersection switches the same direction to green to let the cars pass without stopping. The green wave is efficient for the direction it is aimed at, but interrupts the processing of other directions, which can lead to an increase in overall waiting time.

Alternatively if intersections are close to one another they are often controlled by one traffic light controller as both intersections directly influence one another. However even intersections that are far from one another directly influence the arrival rate of each downstream intersection. Assume the example of Figure 2.4 with three intersections with just two traffic flows per intersection.

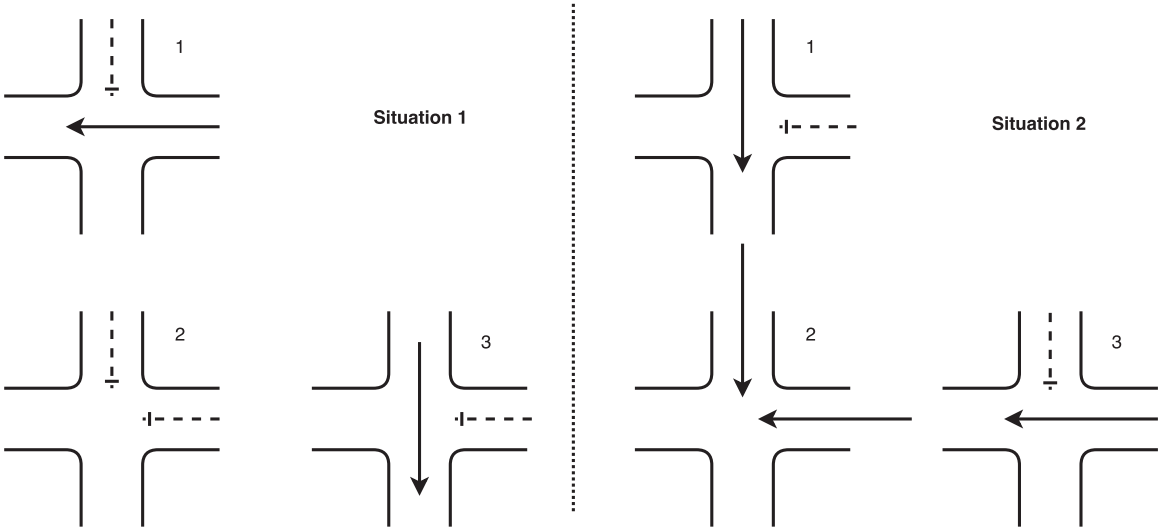

Figure 2.4: A small network of three intersection on which due to trafficlight settings intersection 2 is starving (situation 1) or oversaturated (situation 2)

This example shows that intersection 2 is starved of input or over-saturated based on the settings of intersections 1 and 3. For this example with only unidirectional streets, a global traffic light control could be build to ensure these situations do not occur and waiting time is minimized. However, real network of intersections are much more complex, consisting of multiple intersections were each intersection can have a dozen or more directions to serve. This combined with the stochastic behaviour of arriving traffic flows and individual, selfish, driver behaviour leads to complex dynamics which are hard to predict [3], [10], [11]. A globally controlled network would therefore require a lot of computing power whereas the resulting traffic light control could potentially still result in unpredicted behaviour due to additional arrivals and small perturbations, the 'butterfly effect' [12]. Therefore, we are less interested in a global optimized traffic light controller and more considered to acquire stable, but not necessarily optimal, behaviour. This example shows the challenges when trying to effectively control a network of traffic lights. How this stability is acquired is shown in Chapter 5.

This chapter has introduced basic terminology used in traffic management, as well as a short description of currently employed traffic light controllers and the main difference between these controllers and Smart Traffic. A traffic light controller needs to assign green phases to multiple lanes which are in conflict with one another. The green phases of each lane need to be long enough to allow the waiting traffic to depart but not too long as other lanes cannot get served due to conflicts. Currently the FTS handles this problem by taking average arrival rates in account and assigned green phases of fixed order and length. The VAC changes the length of each green phase according to the actual traffic situation. Smart Traffic chooses the next lane to serve based on the waiting time of actual and upcoming traffic and adapts the duration of the green phase accordingly. To compare these three different traffic light controllers a model is created which simulates traffic behaviour around intersections controlled by traffic lights. By controlling the intersection with each controller the performance is measured and compared. The next chapter focusses on the construction of the model and how traffic behaviour is simulated.

Chapter 3

Model assumptions

The previous chapter has shown basic notation used in traffic management and different traffic light controllers including Smart Traffic. To determine the performance of Smart Traffic the performance of these controllers need to be compared. A traffic model is created which simulates traffic dynamics and queueing behaviour around traffic lights. In this chapter we present the different traffic dynamics occurring around an intersection and the assumptions made to create the model. The assumptions made in this chapter are used throughout the rest of the report unless stated otherwise. These assumptions are made with respect to:

- Car behaviour
- Acceleration
- Saturation flow rate
- Effective green and red time
- Stacking Queue
- Reaction time
- Arrivals

As Smart Traffic aims to minimize the waiting time (or delay) per car, the assumptions should have minimal influence on the waiting time. The waiting time is calculated in the following way: Car i travels a distance s over time period t. Driving at its desired speed V the total travel distance would be $V \cdot t = s_{max}$. Any delay the car experiences leads to a different travelled distance s thus a waiting time T_l of:

$$T_l = \frac{s - s_{max}}{V}. \tag{3.1}$$

Only when the incurred waiting time by the model, with assumptions, is close to the waiting time occurred in a real traffic situation can the model be used as a good representation to test Smart Traffic.

After the assumptions are explained a short description of the most important settings of the model are given. For a complete overview of how the model is constructed see Appendix E. The traffic dynamics of this model are subsequently validated. To achieve this an intersection with a FTS (see Chapter 2) is modelled. The results of the model are compared to results generated by simulation (VISSIM) as well as a theoretic approach. The simulation with VISSIM is assumed to recreate the 'real world'. The theoretic approach is used to calculate the mean waiting time. In this report the approach of van den Broek is used [13] as well as an adaptation to calculate the mean waiting time in continuous time [14]. Comparing the model to the 'real world' and the theory provides an indication of how accurate the model is. The traffic dynamics of the model need to be correct to ensure that Smart Traffic uses a correct input. In the upcoming chapter the model is expanded with the Smart Traffic traffic light controller and tested.

3.1 Car behaviour

On the road a lot of different cars exist with as many different drivers and behaviour. As we are not interested in all the possible behaviour and to improve clarity only one type of car, and driving behaviour is modelled.

Assumption 3.1) Single type of identical traffic. *The arriving traffic is composed of only one type of traffic, e.g., only passenger cars.*

This assumptions does not only assume that only passenger cars arrive in the system it also states that every car has the same behaviour. This means the desired speed (V) for each car is the same. As V is constant no overtaking occurs in the system. Furthermore, the size of each car as well as the desired safety distance in between cars when queuing is constant as well.

Driving consist of constantly evaluating the surroundings and changing the driving speed accordingly. VISSIM is doing this in according to the Weidemann-model [15]. In VISSIM every time step of 0.1 s the behaviour of each car in the system is recalculated. On the other hand due to Assumption 1, the model can work with discrete events; only when the car needs to take action (for example a light has turned red) a calculation takes place. As the behaviour of each car is identical the results are predictable and consistent. This means the model needs to perform less calculations which are also less complex. Thus, less computing power is needed and the model takes less time to produce results.

3.2 Acceleration

As stated before, an important aspect of a continuous simulation is the constant changing of driving speed according to the current situation. However to create a —fast— discrete model a constant driving speed V is assumed. Alternatively, cars can be at a standstill; $V = 0$. Any acceleration behaviour is not taken into account. Instead it is compensated for with a delay. See Figure 3.1.

Assumption 3.2) Traffic follows a GO/NOGO rule. *Traffic is either driving at the desired speed or at a standstill in a queue. There is a fixed delay before a car starts moving*

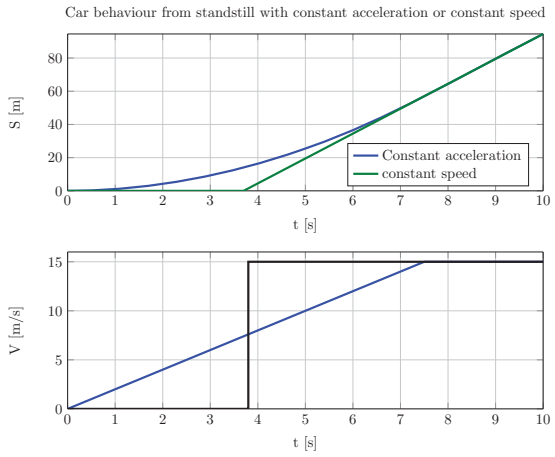

Figure 3.1: At t=0 the first car starts accelerating ($a = 2$), when half the desired speed ($V = 15$) is reached at $t = 3.75$ the second cars starts driving with speed V. Both cars cover the same distance thus their delay is the same

Figure 3.1 shows that a GO/NOGO driving model gives the same results for a model with constant acceleration as long as the GO/NOGO model starts at a later time. This period of time is exactly the time it takes for a car with constant acceleration to reach half of its desired speed. The acceleration delay is exactly: $V/2a$. The acceleration itself, for speeds under 100 km/h, is relatively constant at around 2.8 m/s^2.

Assumption 3.3) The acceleration delay is only dependent on speed V. *The total delay consist of: $V/2a$, this delay occurs every time the car stops regardless of the situation or waiting time.*

In case of deceleration the figure can be interpreted exactly the same but should instead by read from right to left. The results remain the same as well as the overall delay (3.1) suffered by each car.

3.3 Saturation flow rate

The saturation flow rate is the maximum rate at which cars can leave the system. Consider a queue with an infinite amount of cars; The first couple of drivers will cross the light while still accelerating taking up a relative long time before the car has completely crossed the line. At higher speeds the crossing occurs faster. While the safety distance between cars grows (each car becomes 'larger') the overall flow rate will increase until cars cross with the desired speed V. This behaviour is shown in Figure 3.2.

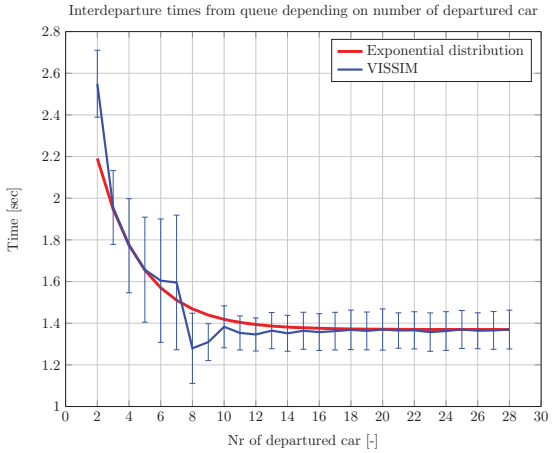

Figure 3.2: Inter departure times between cars when they cross the traffic light. First and second car of the Queue are an exception (1.06 s and 2.55s) others follow an exponential distribution, after 8 cars have left the queue the remaining cars cross with the desired speed (here 50km/h) and with the saturation flow rate μ.

The results of Figure 3.2 are gathered by looking at 360 cycles of a traffic light with an infinity queue. The first car of the queue is an exception; it leaves the simulation around $1s$ with a very small deviation as this delay is almost completely the driver reaction time. The remaining cars follow, roughly, a exponential distribution and quickly approach the maximum departure rate.

As stated by Assumption 3.1), each car acts the same thus there is no variance possible. Furthermore by Assumption 3.2) cars only drive at the maximum speed. This means that there is no distribution of inter-departure times of cars as this would imply a different speed. For the sake of consistency and clarity only the maximum flow rate is used in the model.

Assumption 3.4) The outflow occurs at the maximum flow rate μ. *The outflow occurs at the maximum rate but only starts after the light has turned green, the acceleration delay has passed and the first driver reacted.*

3.4 Reaction time

The reaction time of each driver is around one second, it determines how fast the cars can start accelerating and thus leave the queue. A slow reaction time does not only influence the car's own delay but also the cars behind as they cannot start driving.

The departure rate is already known (Section 3.3). A factor in this departure rate is each driver's reaction time. Looking at VISSIM this reaction time is only a factor for the first couple of cars. In general, drivers look ahead thus they see that the light has turned green even if they are further back in the queue and cannot do anything. When it is their turn to act they do so immediately. So only

the reaction time of the very first driver determines how quickly he can leave. The second car in the queue might still need some reaction time but this is, in most cases, very small (under 0.3 s). Exceptional cases were a driver does not notice the light is green for several seconds or the car stalls are not taken into account.

In the model it is assumed that the reaction time is a fixed delay. This delay is simply added to the acceleration delay as each car in the queue receievs the same penalty. The sum of acceleration delay and reaction time is called the set-up delay and occurs each time a light turns to green while a queue is present. There is no need to model additional reaction time per driver.

Assumption 3.5) Reaction time is zero. *Reaction times are a fixed factor in the acceleration delay and already imbedded in the model*

3.5 Effective green and red time

A traffic light has 3 possible states: Green, Red and Yellow. Mathematically it suffices to only have two states: Effective green and effective red [16]. The effect of this is explained in this section.

First looking at when a light switches from red to green; in Figure 3.1 and Section 3.4 it was shown that cars can follow a GO/NOGO rule as long as a set-up delay is taken into account. The set-up delay lasts for exactly: $t^s = t^{acc} + t^{reaction}$ with $t^{acc} = V/2a$ and $t^{reaction} = 0.5$s. This means that when a light turns green *AND* a queue is present for the first t^s seconds no cars will cross the intersection. In case there is no queue $t^s = 0$.

Secondly when the light is green cars leave according to the saturation flow rate μ. Cars will keep leaving at this rate until the light switches or the queue is empty. When the queue is empty cars leave with the arrival rate $\lambda(t)$.

Finally the light can turn yellow. During the yellow period some cars will chose to stop while others will keep driving. This means the outflow rate μ will continuously drop during the yellow period. It can safely be assumed that no cars drive through the red light. The probability of stopping is directly related to the passed duration of the yellow light. As each driver behaviour is identical it is known when cars will keep driving (effective green) or stop (effective red). In VISSIM driving behaviour during yellow is tested. Departures occur at saturation rate μ with a speed of 50 km/h. It is found that 50% of the cars stop after 2 seconds of yellow light. This rate is not dependent on the total yellow duration since this time is unknown to the drivers. It is dependent on the driver's speed but as we focus on traffic within a city and low speeds this gives only small fluctuations. For this report it is assumed that each green duration is extended by 2 seconds unless stated otherwise. The remaining part t^l of the yellow duration is assumed to be effective red.

The total effective green and red times can now be calculated. Assume a cycle of length C seconds with a red duration or R seconds, green duration of G seconds and yellow duration of Y seconds. According to the definition the effective red time of r seconds becomes: $r = R + t^s + t^l$. Effective green time takes a total of g seconds with $g = C - r$. However, we are interested in minimizing the delay per car (3.1) thus it is expedient to try and have a set-up delay t^s of 0 s. In other words make sure the light turns green just before a queue starts forming. To visualize this aspect t^s is considered part of the effective green period. The definition for effective red becomes; $r = R + t^l$ and effective green; $g = C - r$.

3.6 Stacking Queue

Normally when a queue is formed it will propagate backwards. According to Assumption 3.1) each car has the same size and the same safety distance so the matter of propagation is exactly know. However, Figure 3.2 already shows when each car of the queue crosses the stop-line. Furthermore through Assumption 3.2) the movement speed is known. This means that there is no need to model the backwards propogation of the queue.

Thus, when a car enters the system it will drive all the way to the stop-line upon which it stacks on top of the already waiting cars present. In this case First-In-First-Out (FIFO) is maintained

Assumption 3.6) The time it takes for a car to reach the stop line under queued condition is the same as reaching the position in free flow conditions. *Arriving cars stack on top of the already present queue at the very front of the stop-line.*

Assuming this simplifies the travelling time; each car drives the same distance at the same speed. The number of cars in the queue only influence when the car can leave. Any queue behaviour which would normally occur (moving forward but remaining in the queue) does not need to be modelled.

3.7 Arrivals

An arriving car has no influence on the other arriving or departing cars in the system and occurs at a random point in time. While there are times of high intensity (rush-hour) and low-intensity (night) each individual arrival still independent from each other. As such arrivals are modelled as a poisson process.

Assumption 3.7) Arrivals in a lane occur through a poisson process. *Each car drives independent from one another and as such does not influence the arrivals of other cars.*

However, traffic lights are often part of a network. An upstream traffic light determines when the cars are able to enters a downstream intersection. In these cases the arrivals into the downstream intersection are not Poisson. Only when the distance in between traffic lights is very big and overtaking can take place or a lot of (non-regulated) sideroads also enter the system the arrivals are Poisson. Especially in cities this is seldom the case, instead a moving group of densely packed cars is sent between intersection. These platoons differ in frequency and size due to the different settings of the upstream traffic light. To create this behaviour a dummy traffic light is positioned at the edge of an arriving traffic flow. While not part of the system is takes Poisson generated arrivals and only lets them pass when the light is green. The settings of this dummy light, and thus the way cars and platoon enter the system can be manually set. This adaptation creates a more realistic arrival process.

With the manner of arrivals known the most important assumptions for the traffic dynamics are presented. A short description of the most important settings of the model are presented to show which the different situations the model can recreate.

3.8 Model settings

Under the assumptions presented in the previous sections the traffic dynamics are modelled. In this model several settings are available for a user to create arbitrary intersections. Each intersection consist of several lanes, with specific features for each lane, and a controller that ensures the traffic light is behaving in a proper way. First the lane specific settings are given

- Arrival rate, *Number of cars arriving per lane (given in car/h), these arrivals occur as a Poisson process unless specified otherwise (see below).*
- Platoons, *Cars are put into platoons by using a dummy traffic light that is not part of the rest of the model. The intensity (number of cars in each platoon) and frequency (how often a platoon arrives) can be set. This remains a stochastic function dependent on the arrival rate.*
- Speed, *The maximum allowed speed on this lane. A higher speed leads to a different set-up delay, see Section 3.5.*
- Detector distance, *Distance in m of the detector furthest away from the stop line. As soon as a car crosses the detector it has entered the system. There is another detector at the stop line which ensures cars leave the system.*

Apart from the individual lane settings the intersection has one controller which requires specific settings of its own.

- Conflictmatrix, *Matrix that shows which lanes are in conflict with one another. Additionally it shows clearance times; the time it takes for the remaining traffic to leave the intersection before the light turns green. During this time both traffic lights are red.*
- Minimal green time, *The minimal time a traffic light remains green before it is allowed to switch states.*
- Maximal green time, *The maximal time a traffic light is allowed to stay green, the green period is forced to end after this time regardless of traffic.*

All these settings given above can be set by the users and are enough to create the model. The user can thereafter create any local intersection and by applying different traffic light controller see the resulting throughput. In the following section the validation is done by using a FTS as we are interested to check the difference in traffic dynamics between the model, VISSIM and theory.

3.9 Validation

Consider a simple intersection of 3 lanes as given in Figure 3.3.

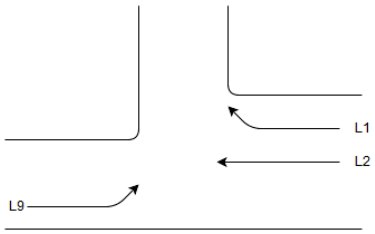

Figure 3.3: Intersection with 3 lanes. The left turning lane (L9) has conflicts with the lane that goes straight (L1) and the right turning lane (L2)

.

The optimal FTS for this system is displayed in Figure 3.4.

Figure 3.4: Snapshot of the Fixed Time Schedule for a 3 lane intersection. Clearance times are taken into consideration.

.

The inflow rate λ_i for each lane are; $\lambda_{L1} = \lambda_{L2} = 100$ car/h and $\lambda_{L9} = 150$ car/h. The allowed speed for each lane is set at 50 km/h. Car arrivals occur through a Poisson process, there are no generated platoons. The exact derivation of the mathematical approach can be found in the respective papers [13], [14].

The mean waiting time for the four approaches are given in Table 3.1.

The resulting waiting time across the different approaches are close to one another. Assuming VISSIM is the real situation all answers show a deviation close to or within 10% of the mean waiting time. Deviations naturally occur as VISSIM is much more advanced than the model or the mathematical approach.

Table 3.1: Different results for the mean waiting time for a 3 lane intersection controlled by a FTS. Results are obtained through simulation (Model, VISSIM) or through a mathematical approximation (vandenBroek, Continous).

	Model	VISSIM	vandenBroek	Continuous
L1	7.36	7.64	7.33	7.60
L2	13.6	12.16	12.95	13.27
L9	12.25	11.28	11.67	11.98
Average	10.39	9.97	10.18	10.47

The most important difference is that VISSIM runs continuously; meaning that every 0.1s every car in the system will determine which action it needs to take. This is similar to an actual driver: Depending on the input driving behaviour will change (Does the light turn red? Did the vehicle in front brake or accelerate, how much traffic is on the road? etc.). The model however is much more discrete: Each car that enters the system knows exactly when it reaches the stopline. When a queue departs each individual car knows when it can leave. This difference in calculation methods is most obvious in the passage of cars from a queue at the stop line. In VISSIM these occur as shown in Figure 3.2 while in the model there will be no passages during set-up delay t^s and afterwards with saturation flow μ. So while the processing of traffic dynamics is completely different between the model and VISSIM the mean waiting time remains within acceptable margins.

In this chapter the assumptions used in the model are explained. These assumptions serve to simplify traffic dynamics around intersections which improve computing speed during simulation. By applying the assumptions the model is created, the different settings that serve to manually create an intersection for the model are shown. Simulations are performed to test the performance of the model. This performance is compared between similar simulations in another traffic simulation program (VISSIM) and by utilizing theoretical approach of the mean waiting time. As the traffic dynamics of the model are now validated the Smart Traffic controller can now be build and implemented into the model. In the next chapter the optimization strategy of Smart Traffic is described after which the model is used to gather results for different cases to determine the performance of Smart Traffic.

Chapter 4

Smart Traffic control

With the underlying traffic model known and validated Smart Traffic is implemented to control the traffic lights and increase throughput of an intersection. This chapter describes how the Smart Traffic optimization works and it is tested on a simple, local, intersection. A local intersection, in contrast to a network, is an intersection in which the arriving and departing flows are never blocked or otherwise regulated. Cars may arrive in a platoon fashion but otherwise occur randomly. The results and advantages of Smart Traffic on a local intersection are also given in this chapter. With Smart Traffic executed and tested on a local intersection, the next step is to control a network of intersections with Smart Traffic. A network of 4 intersections with multiple traffic flows is considered, with each intersection individually controlled by Smart Traffic. Adaptations for Smart Traffic to control a network as well as results are given. In the next chapter a network is tested which is known to cause instability when intersections are locally controlled.

4.1 Smart Traffic Optimization

Assume an intersection with two conflicting traffic flows as shown in Figure 4.1. Clearance time between the two lanes is 5 seconds and the underlying traffic model works according to assumptions made in the previous chapter. Cars are spotted well in advance to provide enough time to perform the optimization.

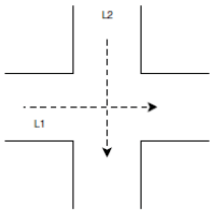

Figure 4.1: Intersection with two conflicting directions. Clearance time between traffic lights is 5 seconds

.

4.1.1 Assumptions

The Smart Traffic optimization used in this rapport aims to minimize the waiting time per car. Other possible optimization strategies might be to minimize the average stops per car or CO_2 output but these cases are not considered in this report. The two most important aspects of Smart Traffic are described below:

- The schedule (order and times in which traffic light turn green) is calculated and fixed for at least 30 seconds in the future.

An upcoming development of driving is the interaction between driver and road. An aspect in this is that traffic lights are able to communicate with the car to tell the driver when his traffic light turns green. This implies that once a green light has been scheduled it cannot be rescheduled to prevent

mixed signals to be sent to the driver which in turn may lead to dangerous situations. Additional the schedule is send to surrounding intersections in the network to serve as input as to when traffic is arriving at these intersections

- A green period may end after the maximum green time has elapsed or when a gap occurs between cars.

A gap is defined as three seconds without departures. For safety reasons it is desirable to not change the traffic light from green to red while cars are leaving at the maximum saturation rate. For this reason the traffic light can only change when a gap occurs between departures. Another possible time in which light can switch to yellow is when the maximum green time is reached. This maximum green time is a setting for Smart Traffic and can be manually set. It is not the maximum green time as defined for FTS and VAC. For Smart Traffic the maximum green time is the time after which another optimization instance is triggered. It is possible that the optimization shows that keeping the current light on green is best; extending the green phase with another maximum green time (or when a gap occurs).

4.1.2 Traffic dynamics

To describe the traffic dynamics we use much of the same notation as given by Lämmer [17]. In Appendix A a step-by-step example of the calculations given below is shown. Assume road section i with length L_i and speed limit V_i. Traffic dynamics are represented by stochastic arrival flow $Q_i^{arr}(t)$ and departure flow $Q_i^{dep}(t)$ with saturation flow Q_i^{max}. As the travel time (L_i/V_i) is known it can be predicted when a car reaches the stopline.

$$Q_i^{exp}(t) = Q_i^{arr}(t - L_i/V_i). \tag{4.1}$$

The total number of vehicles to reach the stop-line $N_i^{arr}(t)$ at time t is given by.

$$N_i^{arr}(t) = \int_{-\infty}^{t} Q_i^{exp}(t')\mathrm{d}t'. \tag{4.2}$$

In the same manner the total number of vehicles that have left $N_i^{dep}(t)$ road section i is given by

$$N_i^{dep}(t) = \int_{-\infty}^{t} Q_i^{dep}(t')\mathrm{d}t'. \tag{4.3}$$

The difference between $N_i^{dep}(t)$ and $N_i^{arr}(t)$ gives the number of cars in the queue $n_i(t)$. As such it also gives the increase of the waiting time $w_i(t)$ over time.

$$\mathrm{d}w_i/\mathrm{d}t = n_i(t) = N_i^{arr}(t) - N_i^{dep}(t). \tag{4.4}$$

The arrival process and corresponding waiting time described by equations (4.1), (4.2),(4.3), (4.4), is visualized in Figure 4.2.

Now assume the factor γ_i which is either 0 (red) or 1 (green). The evolution of the queue length can now be expressed as a nonlinear hybrid dynamical system [18] [19]

$$\frac{\mathrm{d}n_i}{\mathrm{d}t} = \begin{cases} Q_i^{exp}(t) & \text{if} \quad \gamma_i(t) = 0 \\ Q_i^{exp}(t) - Q_i^{max} & \text{if} \quad \gamma_i(t) = 1 \quad \text{and} \quad n_i(t) > 0 \\ 0 & \text{if} \quad \gamma_i(t) = 1 \quad \text{and} \quad n_i(t) = 0 \end{cases} \tag{4.5}$$

Equation 4.5 shows that there are 3 possible evolutions for the queue length; it grows during the red period, it shortens when green and is at minimum zero when the queue has dissolved but the light remains green. In the last state traffic passes the intersection without stopping. These equations

describe how the waiting time evolves over time t for a road section i with a traffic light. To complete the description of the hybrid dynamical system, losses such as acceleration and reaction time also need to be taken into account (see Chapter 3). With a complete description of the system, the needed green time to clear a queue can be anticipated

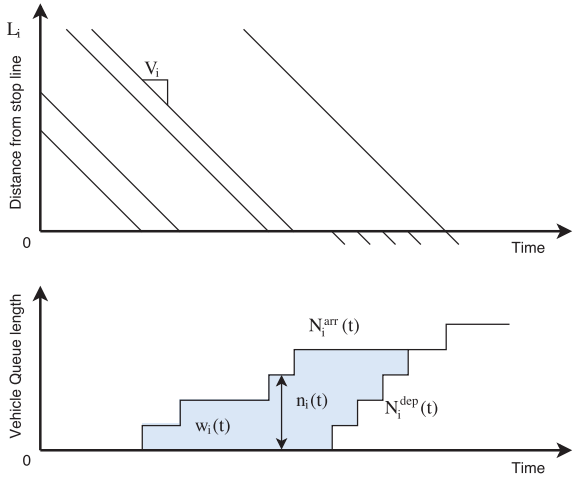

Figure 4.2: Arrival process (top figure) of cars driving towards an arbitrary intersection. The line starts when cars arrive in the lane Q_i^{arr}. As V_i is known so are the expected arrivals at the stopline Q_i^{exp}. Vehicle departures are shown as negative distance distance from stop-line. The supplementary queue length (bottom figure) for these arrivals and departures is also shown. Arrivals leads to a growing queue N_i^{arr} followed by a growing number of departed vehicles N_i^{dep}. The cumulative waiting time incurred as given by the blue area.

4.1.3 Required green time

The traffic light has two possible phases: red and green (Section: 3.5). In turn each of these states consist of smaller sub-phases which are shown in Figure 4.3.

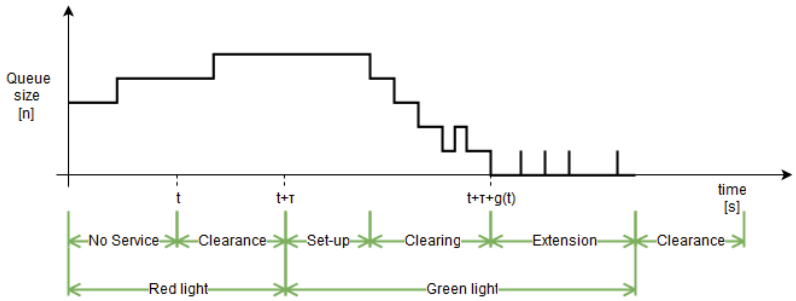

Figure 4.3: Arbitrary queue length evolution with different sub-phases of the traffic light. During a clearance phase traffic is cleared from the intersection. During set-up the acceleration and reaction delay, see assumption 3.3 and 3.5, no departures occur. During the clearing phase departures occur at maximum saturation flow μ while during the extension phase no queue is available and departures occur at arrival rate $\lambda(t)$.

Figure 4.3 shows how the queue length evolves according to (4.4) and (4.5). The switching losses τ consist of the clearance time between road sections. The set-up time during the green period consist of the acceleration delay and reaction time. If there is no queue and a car arrives immediately after the light turns green it occurs no set-up delay. The clearing phase $g(t)$ is dependent on the number of cars waiting at $t + \tau$ as well as any arriving cars $Q_i^{exp}(t)$ during the clearing period. During the clearing period cars are served with saturation flow Q_i^{max}. It is possible to extend the green period in which case cars leave the intersection without stopping (4.5).

Note that $t + \tau + g(t)$ provides the time at which the queue has been cleared. This is not the same as the time in which the green period ends $\hat{g}(t)$, a green period may only end when a gap h occurs.

$$\hat{g}(t) = \begin{cases} g(t) & \text{if} \quad n_i(t+\tau+g(t)+h) = 0 \\ t+\tau+g(t) = t' & \text{otherwise} \end{cases} \tag{4.6}$$

with t' the first arrival after $(t+\tau+g(t))$ given by the smallest solution of the following expression:

$$N_i^{exp}(t') > N_i^{exp}(t+\tau+g(t)) \tag{4.7}$$

Since a new car can reach the stopline before $(t'+h)$ (4.6) is iterative until a gap is found.

The maximum green time (section 4.1.1) does not influence the required green time $\hat{g}(t)$. It merely ensures that when the greentime becomes large Smart Traffic recalculates the required green time with the newest data (additional $Q_i^{arr}(t)$) and if necessary switches the active traffic light. When to switch the traffic lights is described in the next section.

4.1.4 Waiting time optimalization

Smart Traffic aims to minimalisme the waiting time. It is known when a queue (if any) is cleared and a gap occurs; $\hat{g}(t)$. In case this time is far into the future it is limited to the maximum green time t_{maxG}. It is assumed that $t_{maxG} > \tau$ to prevent a new optimization to take place before a light turns green. Taking the two lane intersection of Figure 4.1 into account there are two possible options:

a. Extend the green phase until a gap occurs.

b. End the green phase and switch to the other traffic light and serve this until a gap.

In both cases we are interested in how the queue length (4.5) and thus waiting time (4.4) evolves over both lanes and how long the next instance lasts. Assume Lane 1 is green at time t_0: For option (a.) it is known that at the end of this phase the queue is empty and the remaining waiting time for this lane is 0. This happens after $\hat{g}_1$. Meanwhile for lane 2 the total waiting time grows. This growth is linear dependent on number of waiting cars $n_2(t)$. Find the first car that is waiting in the current queue, i.e. the highest value of t for which

$$N_i^{exp}(t) - N_i^{dep}(t) = 1 \tag{4.8}$$

This time t_a^* is the time at which the queue has formed and the total waiting time for a lane starts growing. The remaining waiting time at the end of option (a) for lane 2 is

$$WT_a = \int_{t_a^*}^{t_0+\hat{g}_1} n_2(t)\mathrm{d}t \tag{4.9}$$

With option (b) a clearance and set-up phase is first initialized (see Figure 4.3) for lane 2 before the queue is being served. The duration for lane 2 is given by $\tau + \hat{g}_2$, at which point the remaining waiting time for lane 2 is zero. However before lane 1 can get served (again) another period τ has elapsed. The waiting time for lane 1 is again determined by first finding the arrival time of the first car in the queue t_b^* (4.8). The remaining waiting time for option (b) is:

$$WT_a = \int_{t_b^*}^{t_0+\hat{g}_2+2\tau} n_1(t)\mathrm{d}t \tag{4.10}$$

The two formulas presented above (4.9) (4.10) give the total waiting time at the end of the green phase for option (a) and (b). We are interested in the rate at which this waiting time is reached.

$$WT_a^2/\hat{g}_1 \leq WT_b^2/(\hat{g}_2+\tau) \tag{4.11}$$

The above criteria can be generalized to any traffic flow i. As we are only interested in a two lane intersection the generalization is not performed in this chapter, see Lämmer [17] for this approach.

The main difference between the criteria as described above and conventional traffic light control is that queue length $n_i(t)$ takes future arrivals into account. Meaning that green phase duration $\hat{g}_i$ takes the current *and* future queue length into account as well as arrivals that occur immediately after a queue has been cleared. Similarly not only the losses of switching are taken into account but also the additional losses of switching back, both represented by τ.

As the aim is to minimize waiting times platoons are served automatically, since when a platoon arrives in the system the value of n_i jumps to a higher value and with it the value $\hat{g}_i$. If a platoon is actually served in time (without stopping) depends on other traffic dynamics. A gap or the maximum green time needs to be reached before a switch can actual occur.

4.2 Local Intersection and results

Assume the intersection of figure 4.1 with identical lanes with the following settings:

Table 4.1: Lane Settings of simple intersection

λ	600 cars/h
L	1000 m
V	50 km/h
τ	5 s

With L and V known the travel time for each car is $L/V = 72s$. Three different traffic light controllers are used for this intersection: FTS, VAC and ST. Arrivals occur at random but cars do sometimes arrive in a platoon fashion. This can be visualized as one traffic light downstream which creates platoons and multiple side-lanes from which cars join the main lane at random. A sample of the queue evolution for a single lane for each of these controllers is shown in Figure 4.4

As stated in the previous section Smart Traffic aims to minimize the waiting time, this is achieved in two ways: The queue length is smaller when the light turns green and the average waiting time of cars is lower. Practically this is achieved by trying to clear the queue (in a platoon fashion) so that upcoming cars can join this platoon without occurring any set-up delay (see Figure 4.3). If the queue cannot be cleared in time, because there is no gap on the other lane, cars often join the queue while the light is green, having received only a minor delay.

The results of this intersection for each traffic light control is shown in Table 4.2

Table 4.2: Mean waiting time and percentage of cars which do not incur stops. Averaged over 10 runs of 10 hours of simulation

	Waiting time [s]			Non-stopping [%]	
	μ	σ	%	μ	σ
FTS	9.63	0.059	100	22.47	0.571
VAC	9.32	0.084	96.8	11.17	0.529
ST	5.38	0.023	55.87	22.04	0.700

Results for the same intersection but with different arrival rates and platoon sizes are found in Appendix B. The general results are: How bigger the platoons are that arrive in the system how better ST performs, this can lead up to a 50% decrease in waiting time in comparison with a FTS. Over half of the cars cross the intersection in a green wave manner and thus do not have to stop. If arrivals are purely random but the intersection is not yet saturated ST the mean waiting time is still less than FTS but more cars need to stop. Due to the nature of ST it naturally creates platoons; these are the most efficient way to clear an intersection. When the intersection is saturated and arrivals occur at random FTS performs slightly better (-1% waiting time) than ST.

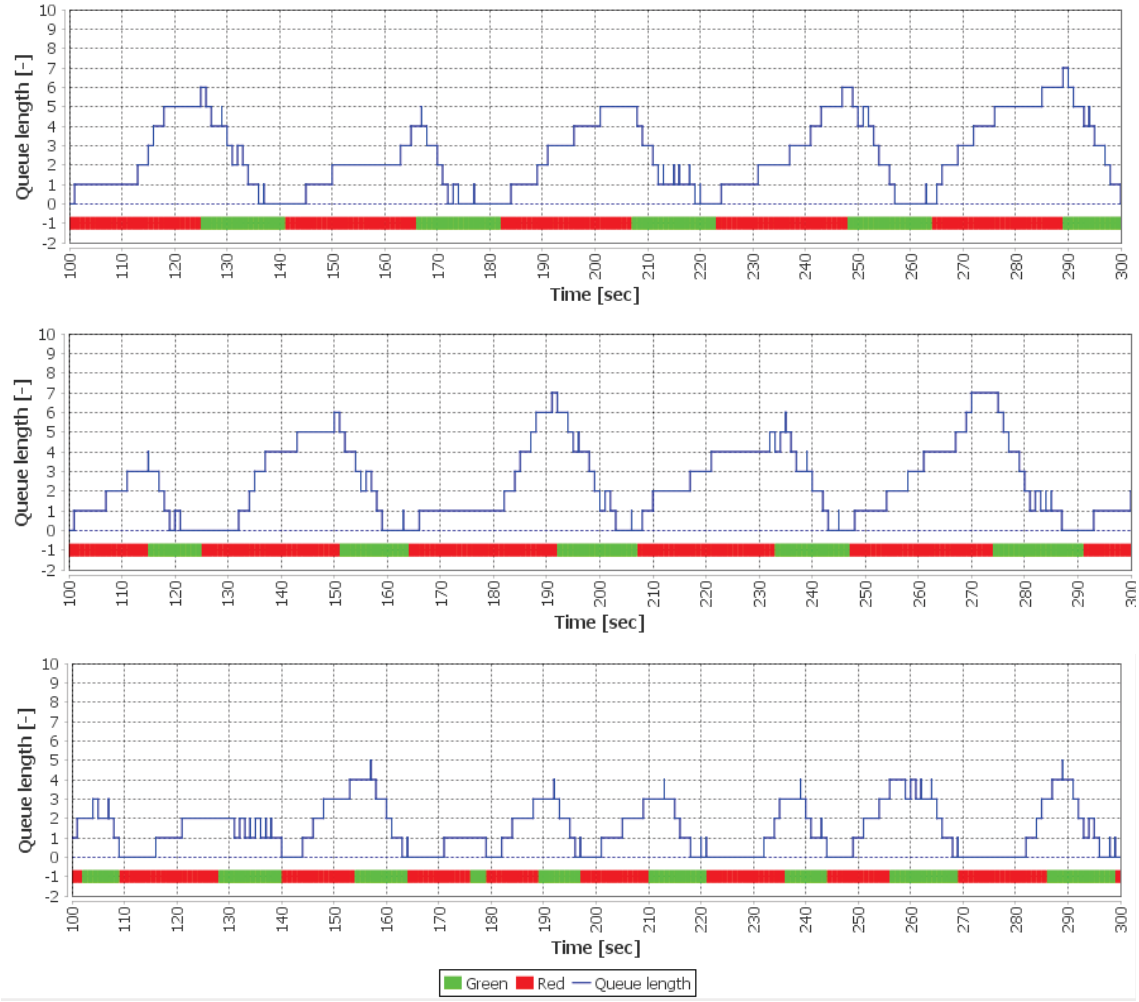

Figure 4.4: The queue length evolution over a single lane for a two-lane intersection with identical lanes and different traffic light controllers. (a) shows a FTS, (b) shows a VAC and (c) shows ST.

4.3 Evaluation and network control

Smart Traffic is implemented on a local, two-lane intersection. It shows an overall decrease in average waiting time. An important aspect in this is the predictability of arrivals as well as the required green time. By calculating the needed green phase to clear the queue (and other arrivals if so desired) the waiting time over all cars in the intersection can be predicted. As such the green phase which leads to the least amount of waiting time of the system can be chosen. The true strength of Smart Traffic lies in arrivals which occur in platoons, by being able to predict when a platoon is forced to stop Smart Traffic can take preliminary action to ensure the platoon receive a green light and pass without incurring any waiting time.

On departure from a traffic light cars travel in a platoons fashion. A downstream traffic light that is controlled by Smart Traffic sees these platoons coming and by clearing the current queue just in time the platoon joins up with the back end of the queue without stopping. This action creates bigger platoons which can be handled by Smart Traffic and further decrease waiting time.

The next section focusses on small networks of intersections controlled by local Smart Traffic instances. Each instance (intersection) controls similar to the used example of Figure 4.1 and does not take into account which directions are being served by other intersections. Forecasted departures are shared with downstream intersections to ensure the downstream intersection has as much information as possible about upcoming arriving traffic. This is especially useful with intersections that are closely

placed to one another as the forecast horizon (30 s) may be bigger than the travel time between the intersections. The section focusses on a network were instability (growing queues over time) may occur, for the results of a basic network see Appendix C. The local control is tested on a network in which the traffic flows influence on another, so called dynamic feedback. The adaptation needed to execute Smart Traffic on the network and results are presented in the next section.

4.4 Berenkuil Roundabout

The Berenkuil is a roundabout located in Eindhoven. It is part of the main road around Eindhoven and the main road (N270) between Eindhoven and Helmond also enters the Berenkuil from the eastern direction, as such a lot of traffic is process by the roundabout. To better control the traffic flows on the Berenkuil traffic lights are placed on the entry ways towards the Berenkuil as well as on the Berenkuil itself. The traffic lights effectively split the roundabout into 4 parts, each part can be seen as an individual intersection. The traffic flows between these parts highly depends on one another which leads to additional constraints for stability that are further explained in Appendix F. The schematic lay-out of the roundabout with lane numbering is shown in Figure 4.5.

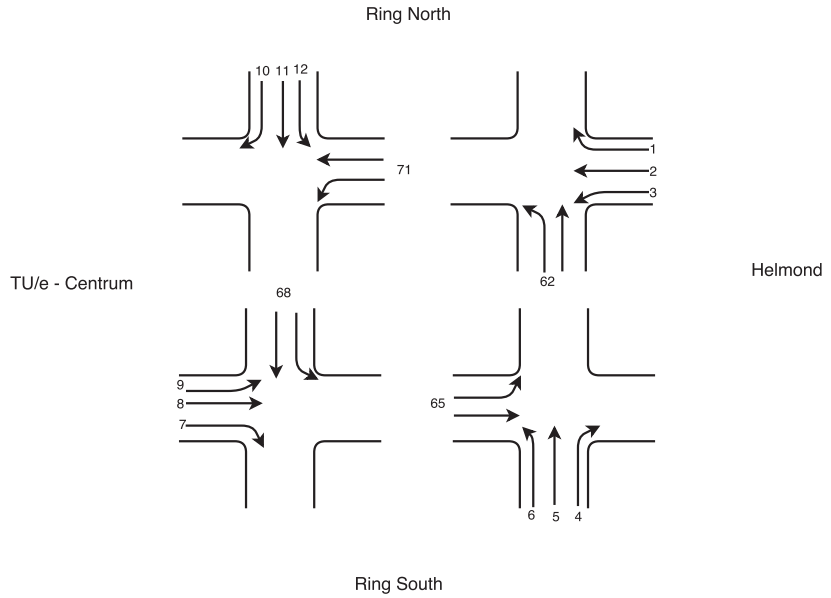

Figure 4.5: The four intersections of the Berenkuil with all traffic flows and lane numbering, lanes 62,65,68,71 combine all feeding traffic flows. Traffic flows 1,2 and 3 consist of traffic driving on the N270 and arrives from Helmond. Flows 7,8,9 consist of traffic departing from the centre of Eindhoven

.

Not all directions of the Berenkuil have the same number of lanes assigned to them. Moreover the arriving traffic flows for traffic turn left or going straight position themselves on the same lane thus they share the available capacity. On the other hand, traffic flow 1 does not enter the roundabout at all instead turning away just before reaching it, as such it has no traffic lights and is not taken into further consideration. The available capacity for each lane as well as arrival rates for morning and evening rush hours are presented in Appendix G. Similar to the intersection of Figure 4.1 Smart Traffic controls a single intersection. The berenkuil consist of four intersections so each intersection is independently controlled by Smart Traffic. We are interested in the queue sizes over time for each intersection, as seen in Appendix G intersections have a high utilization. Smart Traffic needs to correctly schedule each green phase to ensure departures occur at or around the maximum saturation flow. If the scheduling is not done correctly this leads to a decrease in throughput and a growing number of cars waiting. This problem is further complicated by having only limited information about upcoming arrivals for each intersection.

Each Smart Traffic instance only controls its own branch of the roundabout. For traffic that is approaching the roundabout the arrivals are known from upstream sensors. The position of traffic driving on the roundabout is known by the Smart Traffic instances. An upstream instances schedules the departures of each car, the time of departure from this schedule is send through the downstream instance. This way downstream instances know about upcoming arrivals and start scheduling them. Additionally, there is no individual traffic light control for arriving directions (for example 4,5,6). As they use the same lanes all lights either turn simultaneously green or red. The same holds for the lights that are positioned on the intersection, serving lanes 62, 65, 68 or 71. As such each Smart Traffic instance has two possible directions it can serve; traffic positioned on the roundabout or traffic entering the roundabout.

To simulate this network the following assumptions are made:

- The distance between each of the four branches is 20 m.
- Historic arrival rates and lane capacity are used.
- Growing queues on the roundabout do not lead to queues blocking directions, an arbitrary number of cars can wait on the roundabout
- Buffer capacity is shared between lanes that receive simultaneously green, cars are evenly distributed across all lanes regardless of direction they are headed.
- Smart Traffic knows the destination of each car. For example; if six cars are waiting for a, shared, lane it knows that two cars turn left. These two are scheduled and presented to the downstream instance. This way downstream instances receive accurate information about upcoming traffic.
- Traffic never makes an U-turn over the roundabout.

Given the intensities of the morning rush-hour as presented in Appendix G lane 65 receives the most traffic that is making a left turn (arriving from lane 12). In addition, the lane needs to handle simultaneously arriving traffic from lane 8 and 9. The resulting queue-evolution with two different traffic flows (lane 12 and the combination of lane 8 and 9) entering this lane is presented in Figure 4.6.

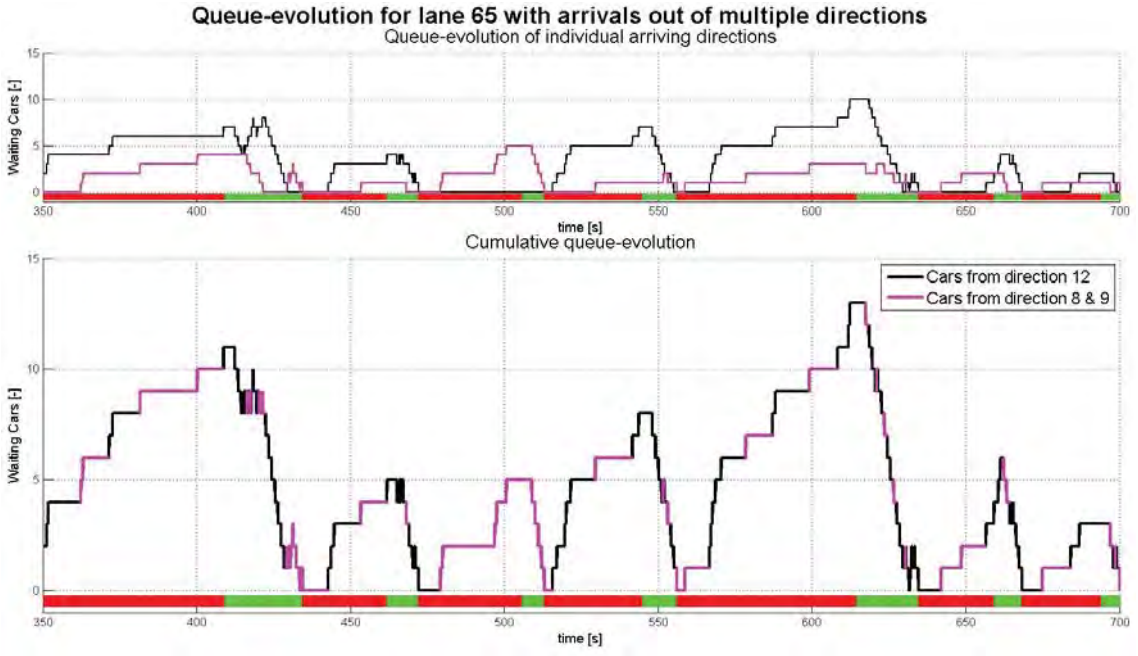

Figure 4.6: Queue size over time for lane 65 with morning rush-hour traffic intensities. Traffic flow nr. 12 enters this lane after being placed in the queue of lane 68. Traffic flow 8 and 9 enter simultaneously with each other. Traffic is processed in the order of arrival (FIFO). Both individual traffic flows as the combined flow of lane 65 is given.

As can be seen in Figure 4.6 each car is processed in order of arrival, multiple flows arrive in lane 68 but these are all processed as one single traffic flow. During a green phase additional arrivals may occur which arrival rate is higher than the process rate of the lane, this leads to a growing queue during a green period. Due to the close proximity of the individual intersections Smart Traffic may experience arrivals that occur during a green phase and can be processed in that same green phase. Smart Traffic correctly extends the green phase to ensure these extra arrivals are also processed. The overall resulting behaviour is stable for each of the four Smart Traffic instances. As seen in the Figure the queue length does become large, larger than the available room for cars to be on. So while the overall performance is stable and all cars are eventually processed by the network the queue length in between intersections become too large. This can be prevented by increasing the cost for cars that are waiting on the intersection with an arbitrary penalty. By introducing a penalty Smart Traffic automatically prioritize these queues but this does lead to a decrease in overall performance. This example shows that Smart Traffic can correctly process the traffic with intersections in close proximity and with multiple traffic flows.

We have seen the increased performance of Smart Traffic in comparison to other traffic light controllers as well as the performance of Smart Traffic on a network of intersections. The increased adaptability of Smart Traffic leads to shorter waiting time for involved traffic. Furthermore, a network of intersections is controlled with each intersection of the network controlled by a Smart Traffic instance. By scheduling the departures from a network and sending this information to downstream intersection a stable network is established, queue lengths and waiting time remain bounded over time. However, networks exist were the local adaptability of Smart Traffic does not increase performance instead it causes undesirable behaviour which lead to increase of waiting time and, in worse case, to unstable behaviour as queues can grow arbitrarily. A network were this is the case is shown in the next chapter. The cause of this undesired behaviour is also explained in the next chapter.

Chapter 5

Network instability

In the previous chapter we have seen the Smart Traffic controlling an isolated intersection as well as network of intersections controlled by local Smart Traffic instances. It is shown that overall waiting time decreases for an isolated intersection under most circumstances. Controlling a network of intersections generally shows in stable results, by sharing arrivals and departures time between intersection enough information is available for Smart Traffic to establish stable control. However, this does not guarantee stable behaviour in all possible network settings. In this chapter we show an example of network that becomes unstable (queues growing arbitrary large) when executed by local Smart Traffic instances, even though a stable FTS has been shown to exist. To explain what causes the instability we define the amount of work in a system and introduce an extra constraint for stability; the amount of work should decrease over time. We show that local controlled Smart Traffic instances violate this constraint which causes the instability. In the next chapter we propose a supervisor which aims to stabilize the network by utilizing the theory explained in this chapter.

5.1 Kumar-Seidman Network

Consider the Kumar-Seidman network [3] as depicted in Figure 5.1, this small network consist of only two intersections and two arriving traffic flows. The two traffic flows are in conflict at both intersections.

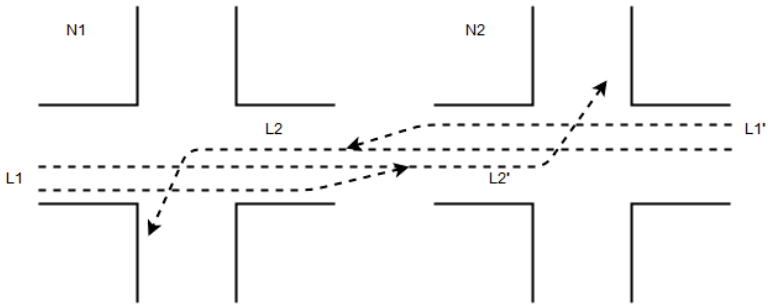

Figure 5.1: The Kumar-Seidman network. Traffic enters through $L1$ or $L1'$ crosses the first intersection and merges into one lane in the distance between the intersection. All cars turn left on the second intersection they encounter ($L2$ or $L2'$.) This causes a conflict with the other lane of the intersection.

For this example a stable FTS exist [3] allowing the network to be controlled in such a way that the queues remain bounded. Unlike a normal FTS this is not done by clearing all queues during each green phase, instead green phases switch even when clearing a queue to increase overall performance [20]. The FTS is thus designed for both intersections and not for each individual intersection. The linking of intersections is needed to ensure stability in this case. Without linking we test this network by applying local Smart Traffic for each intersection. The decentralized approach of Smart Traffic tries to control each individual intersection as best as possible. This approach is opposed to a centralized Fixed-Time Schedule which takes both intersections, and green phases, into account. The arrival rate for each lane is $\lambda_i = 1100$ car/h, platoon arrivals are omitted from this example and the clearance time is $\sigma = 3s$. Saturation flow is set on $\mu_i = 0.5$ car/s, in case two lanes can serve a direction the

maximum saturation flow doubles. The distance between intersections is 100 m at 50 km/h this gives a travel time of 7.2 seconds. The maximum allowed green time is set on 30 seconds. The resulting queue evolution of intersection N1 is shown in Figure 5.2.

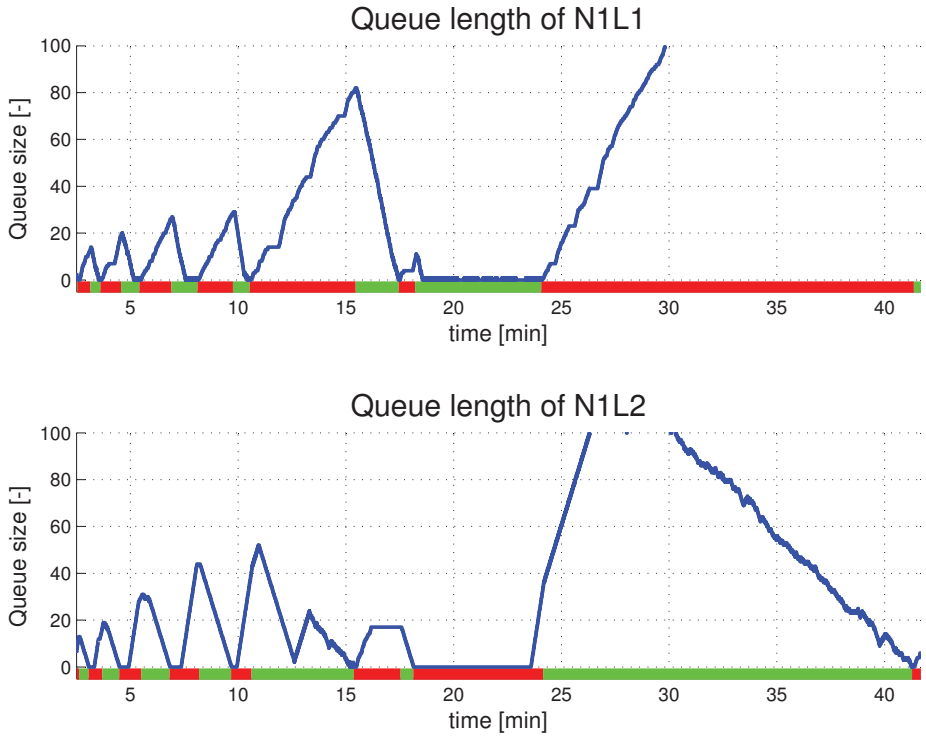

Figure 5.2: Queue evolution of the left intersection (N1) in the Kumar-Seidman Network when controlled by local Smart Traffic instances. Both queues keep growing larger over time. The green and red duration are also presented.

The figure clearly shows a growing queue for both lanes, the results of the other intersection (not showed) shows a similar result. Queues are eventually emptied but the queue sizes and waiting time keep growing over time. Looking at the figure from minute 17 till 24 only lane $N1L1$ is served as no cars are arriving at $N1L2$. Smart Traffic, seeing only this situation decided to keep serving lane $L1$. However, taking the lay-out of the network of Figure 5.1 into account these arrivals do occur but are held up by intersection $N2$. When these cars finally are processed by $N2$ they form a platoon of over 300 (!) cars arriving in succession. This clearly shows that purely local controlled intersection may results in unexpected behaviour leading to queues growing arbitrarily large. We aim to explain this instability by introducing the amount of work in a system.

5.2 Work evolution in the Kumar-Seidman Network

First we reintroduce the Kumar-Seidman network of Figure 5.1 as a manufacturing system, this makes it easier to describe the dynamics in the system which are explained in the next section.

Figure 5.3 is the same network as described in the previous chapter. Input arrives with flow λ into buffer 1 and buffer 3 and flows towards buffer 2 and 4 respectively. The machines A and B can only serve one job at a time which correspond to a traffic light in which the lanes are in conflict with one another. The maximum process rate (saturation flow) is given as μ_i. The actual process rate is presented by u_i with $u_i \leq \mu_i$ for $i = 1, 2, 3, 4$. This rate changes depending on the availability of jobs in the buffer. For ease of calculation it is assumed that there are no set-up or clearance times ($\sigma_{ij} = 0$). Additionally there are is no travel time between the two machines, as soon as a job is processed by A it is immediately available at machine B and vice versa.

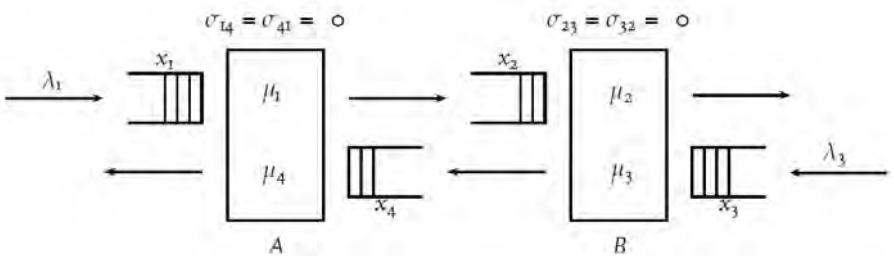

Figure 5.3: The Kumar-Seidman network as introduced in [21]

.

An easy first check for stability is considering if each machine (or intersection) has enough capacity to handle the arriving jobs (or traffic). Similar to (2.1) the machine has sufficient capacity if its utilization is smaller of equal to 1, for each machine this gives

$$\begin{aligned} \lambda_1 \frac{1}{\mu_1} + \lambda_3 \frac{1}{\mu_4} < 1 \\ \lambda_1 \frac{1}{\mu_2} + \lambda_3 \frac{1}{\mu_3} < 1. \end{aligned} \tag{5.1}$$

Using this expression for the network used in the previous sectopm we state that:

$$\begin{aligned} \lambda_1 = \lambda_3 = 1100/3600 \\ \mu_1 = \mu_3 = 1 \\ \mu_2 = \mu_4 = 0,5 \end{aligned} \tag{5.2}$$

Thus both machines have an utilization of $\frac{11}{12}$. Still in the previous section it was shown that the resulting behaviour is unstable.

5.3 System Dynamics

Using continuous dynamics we define how the buffer sizes are evolving over time.

$$\begin{aligned} \dot{x}_1 &= \lambda_1 - u_1(t) \\ \dot{x}_2 &= u_1(t) - u_2(t) \\ \dot{x}_3 &= \lambda_3 - u_3(t) \\ \dot{x}_4 &= u_3(t) - u_4(t). \end{aligned} \tag{5.3}$$

Note that λ_1 is the arrival rate into the system for jobs that enter via buffer x_1. Arrivals at x_3 arrive with rate λ_3. Depending on the buffer sizes the machine can perform on its maximum process rate or, when the buffer is empty, with the arrival rate. Using this there are 16 possible combinations of buffer sizes and active processes. These 16 modi describe all possible behaviour of the entire system in contrast to per individual machine. All 16 modi are shown in Table 5.1 as well as the conditions that apply for each mode. It only shows the buffers that are actively served, the remaining buffers are served with $u_i = 0$ and not displayed.

Note that Table 5.1 shows modi $[u_1 = \mu_1, u_2 = u_1]$ and $[u_4 = u_3, u_3 = \mu_3]$ which have multiple solutions. In case $\mu_1 > \mu_2$ machine B produces at most with rate μ_2 meaning the buffer size always grows ($\dot{x_2} > 0$) thus the state itself can never be reached. In case $\mu_1 \leq \mu_2$ machine B produces at rate μ_1 and the buffer remains empty. The same reasoning holds for the state: $[u_4 = u_3, u_3 = \mu_3]$. As per Figure 5.1 $\mu_1 > \mu_2$ and $\mu_3 > \mu_4$ thus these two modi cannot be reached and are not used throughout the remaining part of this chapter.

Table 5.1: Process flows over different machine and buffer states

machine A	machine B			
$u_1 = \mu_1$	$u_2 = \mu_2$	for	$x_1 > 0$	$x_2 > 0$
$u_1 = \mu_1$	$u_2 = u_1$	for	$x_1 > 0$	$x_2 = 0$
$u_1 = \lambda_1$	$u_2 = \mu_2$	for	$x_1 = 0$	$x_2 > 0$
$u_1 = \lambda_1$	$u_2 = \lambda_1$	for	$x_1 = 0$	$x_2 = 0$
$u_1 = \mu_1$	$u_3 = \mu_3$	for	$x_1 > 0$	$x_3 > 0$
$u_1 = \mu_1$	$u_3 = \lambda_3$	for	$x_1 > 0$	$x_3 = 0$
$u_1 = \lambda_1$	$u_3 = \mu_3$	for	$x_1 = 0$	$x_3 > 0$
$u_1 = \lambda_1$	$u_3 = \lambda_3$	for	$x_1 = 0$	$x_3 = 0$
$u_4 = \mu_4$	$u_3 = \mu_3$	for	$x_4 > 0$	$x_3 > 0$
$u_4 = \mu_4$	$u_3 = \lambda_3$	for	$x_4 > 0$	$x_3 = 0$
$u_4 = u_3$	$u_3 = \mu_3$	for	$x_4 = 0$	$x_3 > 0$
$u_4 = \lambda_3$	$u_3 = \lambda_3$	for	$x_4 = 0$	$x_3 = 0$
$u_4 = \mu_4$	$u_2 = \mu_2$	for	$x_4 > 0$	$x_2 > 0$
$u_4 = \mu_4$	$u_2 = 0$	for	$x_4 > 0$	$x_2 = 0$
$u_4 = 0$	$u_2 = \mu_2$	for	$x_4 = 0$	$x_2 > 0$
$u_4 = 0$	$u_2 = 0$	for	$x_4 = 0$	$x_2 = 0$

5.4 Work in process

The Kumar-Seidman network is now described as a manufacturing network and all (reachable) modes are defined. By defining the amount of work in a system we determine which mode processes the highest amount of work. The more work is done in a mode the better the complete system is utilized. For a single job the amount of work is its total remaining process time. When a job enters the system at machine A its total work is $\frac{1}{\mu_1} + \frac{1}{\mu_2}$, after the first process step (μ_1) the remaining work is $\frac{1}{\mu_2}$. For jobs that enter the system at machine B the same logic applies. With the buffer sizes known the total amount of work in the system is given by:

$$w(t) = (\frac{1}{\mu_1} + \frac{1}{\mu_2}) \cdot x_1(t) + \frac{1}{\mu_2} \cdot x_2(t) + (\frac{1}{\mu_3} + \frac{1}{\mu_4}) \cdot x_3(t) + \frac{1}{\mu_4} \cdot x_4(t). \tag{5.4}$$

We are interested in the evolution of the amount of work:

$$\dot{w}(t) = (\frac{1}{\mu_1} + \frac{1}{\mu_2}) \cdot \dot{x}_1(t) + \frac{1}{\mu_2} \cdot \dot{x}_2(t) + (\frac{1}{\mu_3} + \frac{1}{\mu_4}) \cdot \dot{x}_3(t) + \frac{1}{\mu_4} \cdot \dot{x}_4(t). \tag{5.5}$$

Combining this equation with the system dynamics of (F.1) we determine how the work increases, or decreases, depending on the mode of the system. The expression for any mode is:

$$\dot{w}(t) = \lambda_1(\frac{1}{\mu_1} + \frac{1}{\mu_2}) + \lambda_3(\frac{1}{\mu_3} + \frac{1}{\mu_4}) + u_1\frac{1}{\mu_1} + u_2\frac{1}{\mu_2} + u_3\frac{1}{\mu_3} + u_4\frac{1}{\mu_4}. \tag{5.6}$$

Arrivals occur with rate λ_i with each arrival containing an amount of work leading to a constant increase of amount of work. Meanwhile, the system decreases this amount of work depending on the combined process rates of $u_i\frac{1}{\mu_i}$. To ensure the amount of work in the system does not increase the amount of work done should be equal to or bigger than the increase in work. We therefor split up equation (5.6) into work entering and leaving the system:

$$\lambda_1(\frac{1}{\mu_1} + \frac{1}{\mu_2}) + \lambda_3(\frac{1}{\mu_3} + \frac{1}{\mu_4}) \leq u_1\frac{1}{\mu_1} + u_2\frac{1}{\mu_2} + u_3\frac{1}{\mu_3} + u_4\frac{1}{\mu_4}. \tag{5.7}$$

Note that the left hand side is identical to the sum of stability criteria for the individual machines (5.1) and this value can never exceed 2. We can now define the decrease of work in the system

for every mode. As long as the mode satisfies (5.7) the total amount of work in the entire system decreases. In other words the system should always be in one of the states that guarantees a decrease in work to ensure stability.

Executing the right hand side of equation (5.7) for the 14 remaining modes of Table 5.1 gives a new table:

Table 5.2: Process flows over different machine and buffer states and work **decrease** per unit of time

machine A	machine B	$\dot{w}$
$u_1 = \mu_1$	$u_2 = \mu_2$	2
$u_1 = \lambda_1$	$u_2 = \mu_2$	$\frac{\lambda_1}{\mu_1} + 1$
$u_1 = \lambda_1$	$u_2 = \lambda_1$	$\lambda_1(\frac{1}{\mu_1} + \frac{1}{\mu_2})$
$u_1 = \mu_1$	$u_3 = \mu_3$	2
$u_1 = \mu_1$	$u_3 = \lambda_3$	$\frac{\lambda_3}{\mu_3} + 1$
$u_1 = \lambda_1$	$u_3 = \mu_3$	$\frac{\lambda_1}{\mu_1} + 1$
$u_1 = \lambda_1$	$u_3 = \lambda_3$	$\lambda_1 \frac{1}{\mu_1} + \lambda_3 \frac{1}{\mu_3}$
$u_4 = \mu_4$	$u_3 = \mu_3$	2
$u_4 = \mu_4$	$u_3 = \lambda_3$	$\frac{\lambda_3}{\mu_3} + 1$
$u_4 = \lambda_3$	$u_3 = \lambda_3$	$\lambda_3(\frac{1}{\mu_3} + \frac{1}{\mu_4})$
$u_4 = \mu_4$	$u_2 = \mu_2$	2
$u_4 = \mu_4$	$u_2 = 0$	1
$u_4 = 0$	$u_2 = \mu_2$	1
$u_4 = 0$	$u_2 = 0$	0

The table shows that if both machines (intersections) operate at maximum rate μ_i the work decreases with 2 regardless of the job type processed. As stated by equation (5.1) the maximum allowed amount of work that enters the system is also 2. As long as the system stays in a state were the work decreases by at least 2 it always remains stable. If the amount of work entering the system is lower other modes can also guarantee stability as long as equation (5.6) holds for that mode.

5.4.1 Mode graph of the Kumar-Seidman network

Table 5.2 showed the amount of work processed in each mode. This can be visualized in a mode graph. The modes with the highest decrease of amount of work are placed on top and in decreasing order the rest of the modi are placed. Additionally, the increase in work is also know (see equation 5.7) and displayed as the dotted red line.

Figure 5.4 is created for the values of the Kumar-Seidman network given in (5.2). Any changes in arrival rate lead to a changes in amount of work entering the system. The position of the red line changes accordingly. When platoons arrive ($\lambda_i = \mu_i$) the amount of work entering reaches the maximum value of 2. Additionally, the Kumar-Seidman network consist of two identical intersections. Changing the departure rates of one intersection (5.2) leads to a different lay-out of the figure. This can also be seen by the expressions of Table 5.2.

Figure 5.4 shows that, for these conditions, only 4 modes are available to ensure the total amount of work in the network decreases and thus the system is stable. These 4 modes are called efficient modes, all other modes are inefficient. It shows that this system is only efficient when **both** machines operate at the process rate. Taking this conclusion into account we reconsider the Kumar-Seidman network as introduced in the start of this chapter. We are interested in the different modes the system enters when controlled by local Smart Traffic instances. If the system reaches an inefficient mode the amount of work in the system starts increasing which leads to instability.

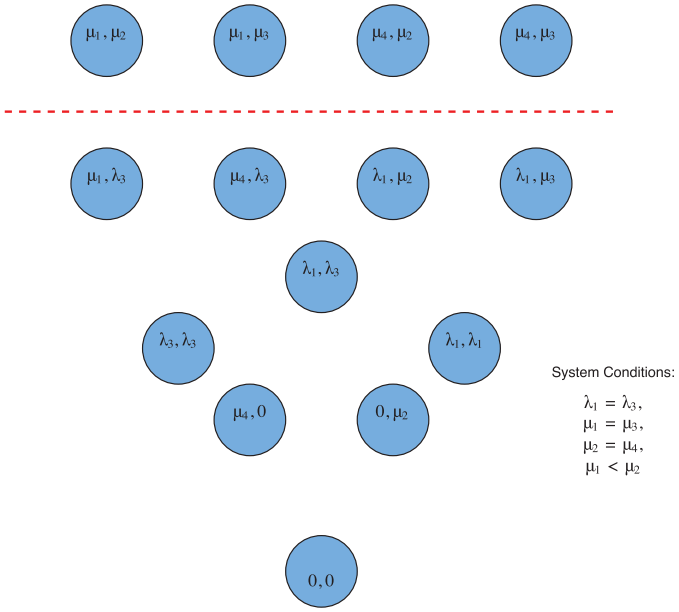

Figure 5.4: Work decrease per unit of time per mode of Kumar-Seidman. The higher the position of the mode the higher the decrease. The red-line indicates the increase in work per unit of time.

5.5 Mode-evolution of local controlled Kumar-Seidman Network

Again we consider the Kumar-Seidman network of Figure 5.1 controlled by local Smart-Traffic instances. Assume that for the initial conditions all buffers are empty save for x_1 which has an arbitrary number of jobs available. Seeing the jobs at x_1 the left intersection starts serving this queue with rate μ_1. These jobs arrive at the right-hand intersection which immediately starts serving them with rate μ_2. The first mode the total network is in becomes $[\mu_1, \mu_2]$ (see Table 5.2). The remaining evolution of modi is described in Appendix D as well as further extensive explanation. The resulting queue behaviour for the Kumar-Seidman network with these initial conditions are given in Figure 5.5.

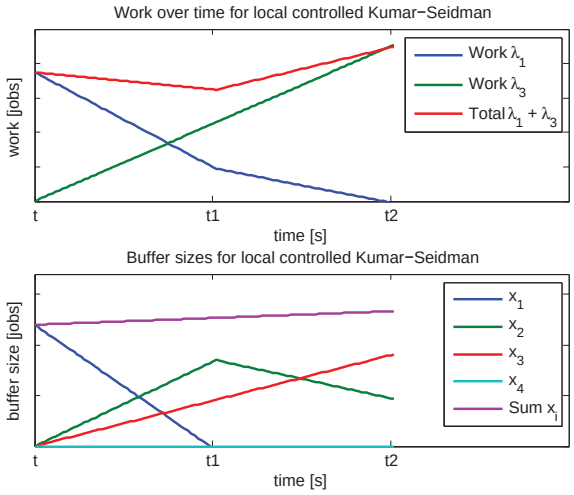

Figure 5.5: Work and queue evolution for a local controlled Kumar-Seidman network. Travel-time and the clearance time are omitted. Work is denoted per incoming flow λ_i and the total amount of work, queue sizes are displayed per queue and the total queue size

.

The situation presented by Figure 5.5 shows the system in two modi while the traffic lights do not change. The first mode $[\mu_1, \mu_2]$ is an efficient mode (Figure 5.4), from t_1 onwards the system is in inefficient mode $[\lambda_1, \mu_2]$, this leads to an increase in work in the system.

Considering the left intersection it sees the following situation between t_1 and t_2: Buffer x_1 and x_4 are empty and the only arrivals arrive in x_1 with rate λ_1, these arrivals are immediately being processed. There is no way that the left intersection can process jobs with saturation flow μ_i as all queues are empty, it therefor defaults to a inefficient mode.

In this chapter we have discussed the Kumar-Seidman network. For this network the total utilization of each intersection is smaller than 1 and a stable FTS exist for this network. However, when this network is controlled by local Smart Traffic instances instability occurs with queues growing arbitrary large. This implies that the instability is caused by the Smart Traffic instances. To explain the instability we define the amount of work in a system, an expression for the remaining amount of processing time over all cars in the system. Due to arrivals with rate λ_i the amount of work in the system continuously increases, this increase is balanced by the intersections processing the cars and leading them out of the system. As long as the processing happens at the saturation flow μ_i the amount of work in the system decreases and stable behaviour is achieved. We redefine the different rates and traffic flows the system can serve into modes. A mode is efficient when it decreases the amount of work in the system. However a mode may be inefficient leading to an increase in amount of work, causing queues to grow. As long as the total system (so not local intersections) remain in an efficient mode the system remains stable. However, due to the lay-out of the Kumar-Seidman network situations occur where one of the two intersection cannot process at the saturation flow due to a lack of arriving cars. This causes the total system to reach an inefficient mode. In turn this leads to a growing amount of work in the network, causing queues to grow and unstable behaviour. We want to expand the Smart Traffic control strategy so it prevent the system from reaching an inefficient mode. In the next chapter we propose a supervisor which, by detecting upcoming inefficient system modes, sets the traffic light settings on a network level to ensure the system stays in efficient modes and can thus exhibit stable behaviour.

Chapter 6

Network Supervisor

In the previous chapter the definition of work is explained and how the amount of work is used as an indication to determine the efficiency of an entire network. It is shown that applying Smart Traffic on individual intersections can lead to an increase in work of the total network. Due to this increase in work the entire network becomes unstable leading to the behaviour as seen in Chapter 5. In this chapter a supervisor is introduced which acts on a network-wide level. By ensuring the network is always in an efficient mode (total amount of work in the network is decreasing) the complete network executes stable behaviour.

6.1 Supervisor in Modes

In Figure 5.4 it was shown that 4 modes of the Kumar-Seidman Network are processing jobs at such a rate that the overall amount of work in the system decreases. All other modes cause the amount of work in the system to increase and are deemed inefficient. Additionally it was shown that a Smart Traffic instance that only controls one intersection can reach these inefficient modes and thus cause unstable behaviour. To prevent this behaviour a supervisor has been designed which identifies the upcoming modes of Smart Traffic and, when the mode is inefficient, it can change the trafficlight settings of the entire network to ensure the network remains in an efficient mode. Which changes can occur depend on the (inefficient) mode of the system and the buffer sizes. All possible supervisor actions are displayed in Figure 6.1 as arrows.

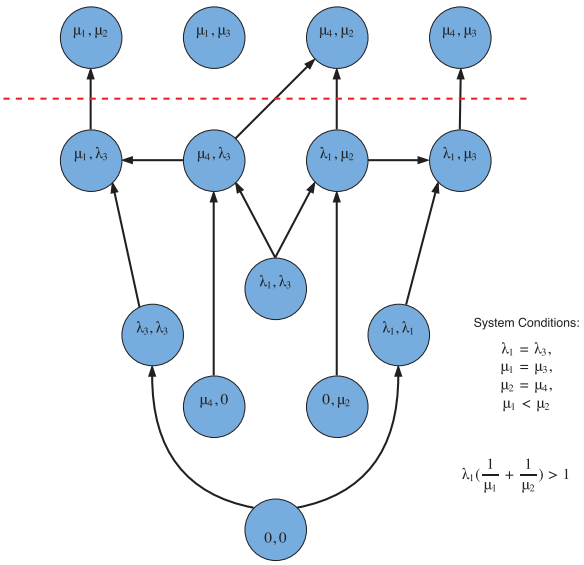

Figure 6.1: Mode graph with the modes assigned by decreasing order of efficiency. The red line indicates the amount of work that enters the system. A mode is efficient if it is positioned above the line and inefficient otherwise. A decrease in arrivals into the system leads to more modes becoming efficient. Arrows indicate which actions the supervisor can execute to ensure the system remains in an efficient mode. Table 5.1 shows what conditions (queue sizes) exist in each mode.

Most of the supervisor actions only influence one trafficlight and are automatically performed by Smart Traffic. For example in case the system is in mode $[\lambda_3, \lambda_3]$ it is serving empty queues x_4 and x_3. Meanwhile a queue starts forming at x_1, the Smart Traffic instance working at the left intersection shall automatically switch to serve this queue. This leads the system to reach mode $[\mu_1, \lambda_3]$ which is more efficient. All arrows that point upwards are executed in such a manner and do not require additional supervisor actions. However there are two arrows that point sideways and these actions are never performed by Smart Traffic. First lets consider these mode: $[\lambda_1, \mu_2]$ and the mirrored state $[\mu_4, \lambda_3]$. If any jobs are available in buffer x_4 or x_3 respectively the local Smart Traffic instance switches to mode $[\mu_4, \mu_2]$ which is an efficient mode. However if these buffers are empty Smart Traffic does not switch to a more efficient mode leading to the inefficient situation shown in Figure 5.5. The required action that the supervisor performs is to stop serving the job that is being processed at maximum saturation flow, this action causes jobs to arrive at x_2 or x_4 which, in turn, leads to an increased efficiency of the total network. The results of this action is further explained in the next section.

6.1.1 Mode-evolution with Supervisor in Kumar-Seidman network

Consider the same system and initial conditions (All buffers empty except for x_1) as used in Chapter 5 which showed a growing amount of work in the system with only local control. We now use the supervisor to switch the traffic lights whenever the system reaches an inefficient mode. This leads to the work and queue evolution as shown in Figure 6.2.

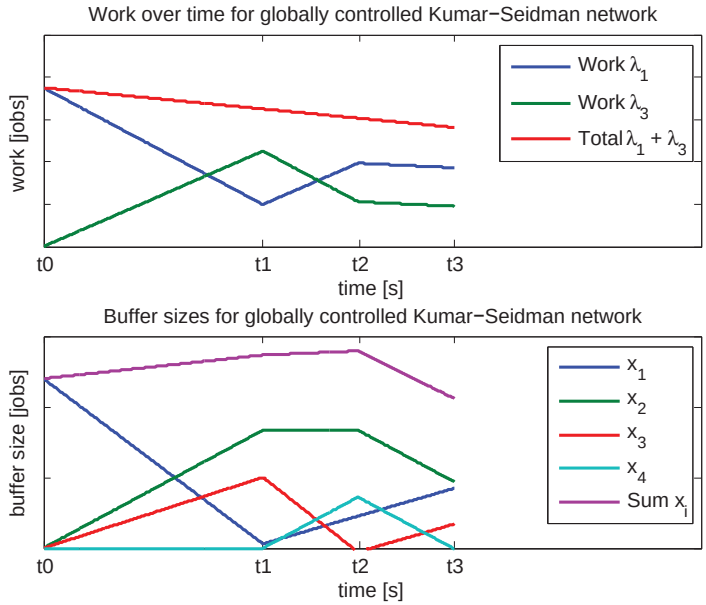

Figure 6.2: Work and queue evolution for the Kumar-Seidman network with a supervisor for stability. Travel and clearance times are omitted.

Figure 6.2 shows that the amount of work continuously decreases over time. In certain modes the total queue size still grows over time but after one cycle (t_0, t_3) the overall queue size has decreased. The modes the system is in are:

$$\begin{array}{rll} (\mu_1, \mu_2) & \text{for} & [t_0, t_1] \\ (\lambda_1, \mu_2) & \text{for} & [t_1, t_1] \\ (\lambda_1, \mu_3) & \text{for} & [t_1, t_1] \\ (\mu_4, \mu_3) & \text{for} & [t_1, t_2] \\ (\mu_4, \mu_2) & \text{for} & [t_2, t_3] \end{array}$$

Note that this t_2 is not the same as in the local controlled system. Also after t_3 the system is again in mode (μ_1, μ_2). There are 2 actions the supervisor performs, both on t_1, this action causes both machines to switch and ensures they both perform at the optimal rate. After the supervisor has acted the control is handed back to the local clearing policy. By applying the supervisor the system shows stable behaviour.

6.2 Simulation results

The described supervisor of the previous section is added to the model. It checks the efficiency of the total network every time Smart Traffic has scheduled a new green phase. In case this green phase causes the network to be in an inefficient mode the supervisor acts and by changing the green phases it ensures the system remains in a stable mode. We execute the supervisor for the same model as used in Chapter 5, leading to the following queue-evolutions:

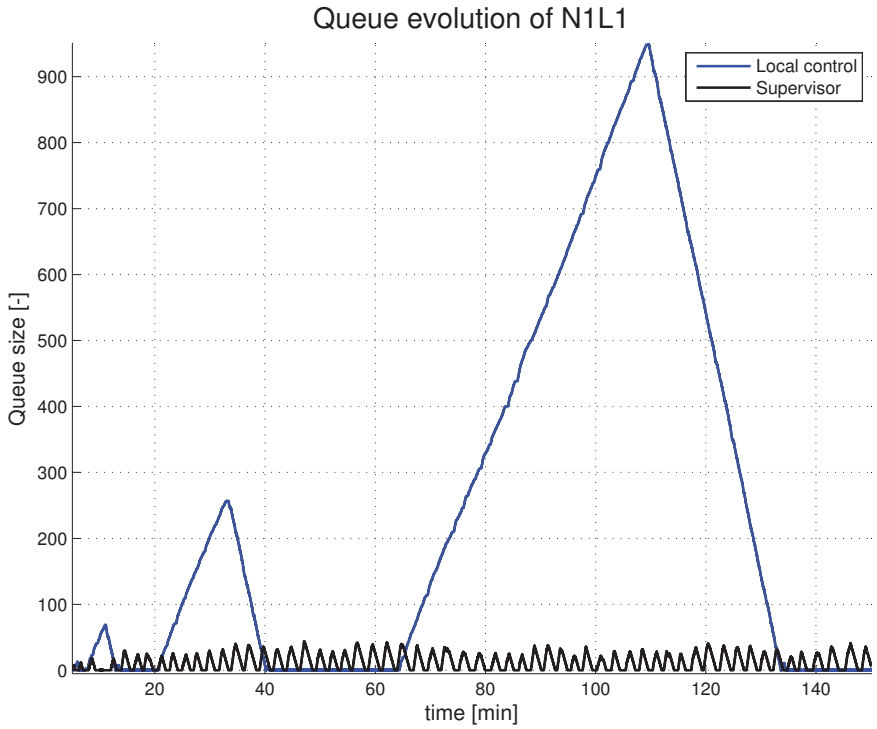

Figure 6.3: Queue evolution for lane N1L1 of the Kumar-Seidman network under local Smart Traffic instances (blue line) and with the applied supervisor (black line). Supervisor causes the queues to remain bounded over time. The queue initially grow due to the very high utilization of the network but become stable over time.

The queues in this simulation remain bounded unlike the results of the same network without supervisor of Figure 5.2. Due to the very high utilization of the total network the queue length remains very long. Queue lengths grow this large as green phases last relatively long. This is done to decrease the amount of switching between lanes as during switching no traffic can cross the intersection. As queues remain bounded over time stability is reached proving the working of the supervisor in a network that is locally controlled by Smart Traffic instances. This example is small with only two incoming traffic flows and two intersections but the approach of defining the work and identifying modes can be extrapolated to bigger networks.

6.3 Evaluation

The supervisor as described in this chapter is implemented in the model with a resulting stable behaviour of the Kumar-Seidman model. Modes are assigned by measuring the increase or decrease of amount of work for each upcoming green phase. When a mode is inefficient (amount of work increases) the supervisor has predefined paths to take to make the system reach an efficient mode. This causes the system to always be in an efficient mode, the amount of work remains as low as possible and stability is achieved.

As the arrival rate, and thus the amount of work entering the Kumar-Seidman network is high there are only few efficient modes available. The high utilization of the network is also displayed by the large queues (up to 50 cars) and large green times. As arrival rates fluctuate over time the red line shown in Figure 6.1 also changes position over time, more modes become efficient when the arrival rates are lower. So under lower arrival rates the supervisor becomes less strict while still guaranteeing stability. Additionally, when platoons arrive ($\lambda_i = \mu_i$) the amount of work entering the system reaches the maximum value. When these platoons are detected by Smart Traffic traffic lights adjust to serve this platoons, and the system automatically reaches an efficient mode. The supervisor as such only monitors the behaviour of traffic across the entire network by calculating the efficiency of every intersection. It only has to act when the combined settings reaches an inefficient mode. By predefining the path needed to reach an efficient mode to actions of the supervisor are limited and can quickly be taken. Under most circumstances local controlled Smart Traffic instances are sufficient to cause stable behaviour across a network while optimizing the amount of waiting time incurred by the traffic. The supervisor is a slim method to monitor stability and quickly respond when an inefficient mode is about to be reached.

Chapter 7

Conclusion and Recommendations

The aim of this research project was to test the viability of Smart Traffic by comparing it to current traffic light controllers (FTS and VAC). Additionally, Smart Traffic was tested on a network of intersections to determine its performance. A supervisor is created which, with only minimal disturbances, guarantees stability on a traffic network and is a useful addition to local Smart Traffic instances to improve the overall performance of the newly designed traffic light controller that is Smart Traffic. This chapter recapitulates the conclusions drawn throughout this rapport. This is followed by recommendations of the research that is yet to be executed, or is currently underway, to make Smart Traffic into a complete product that can be used on the streets.

7.1 Conclusion

To test the performance of Smart Traffic, a simulation model is created which approximates traffic dynamics. By only focussing on traffic dynamics that occur around a traffic light, the model remains relatively slim and provides a good overview of possible reactions of individual drivers. As it only focusses on a small part of all possible traffic behaviour its calculation speed is very high leading to fast results. This is further increased by using discrete event simulation instead of continuous simulation. As traffic dynamics are generalized, it is unnecessary to review each car its possible actions over each time step. It is shown that while the resulting waiting time for the traffic in a FTS-setting slightly differs from other traffic simulation models (VISSIM) and theoretical approaches, these difference are within acceptable bounds (10%). As traffic dynamics do not change when the traffic light is controlled by a different policy, the model is taken as a solid basis on which Smart Traffic can be tested and compared to other traffic light controllers.

Smart Traffic, by predicting the resulting delay of each car waiting for and arriving at an intersection, knows what each upcoming green phase can do to keep the waiting time as short as possible. As it knows the consequences of its own actions, it can pick the green phase which provides the best result. Aiming to keep the incurring waiting time for cars as low as possible and by detecting traffic in advance it automatically aims to prevent upcoming platoons from stopping. Stopping a platoon would inquire a steep increase of total waiting time for the intersection. This manner of anticipation and pro-active control is unseen in the other tested traffic light controllers, the FTS and VAC. Overall the performance of Smart Traffic is better than both other controllers. Especially when upcoming traffic arrives almost exclusively as platoons (caused by upstream traffic lights) the decrease in waiting time can be up to 60% compared to a FTS and 40% for a VAC. As platoons are scheduled in advance they can cross the intersection without stopping, 80% less cars have to stop for the intersection in contrast to FTS. When arrivals occur randomly these gains decrease; when all arrivals occur randomly the decrease in waiting time for Smart Traffic is around 20% lower in comparison to a FTS and only 5% better than VAC. An intersection with saturated traffic flows and purely random arrivals does not profit by applying Smart Traffic and in these cases an FTS is a better solution. However, as traffic demands drops and rises throughout the day, intersections are not continuously (over-)saturated. The Smart Traffic approach is much more adaptable and provides better overall results for a local intersection.

In the second part of this research, networks of intersections are considered. By applying the Kumar-Seidman network on Smart Traffic instances that only control one intersection it was quickly shown

that this leads to unstable (queues becoming arbitrarily large) behaviour. This behaviour is caused by the traffic being sent in between the intersections. As the Smart Traffic chooses to serve the direction which causes the highest decrease in waiting time it does not consider that this action might block traffic being send towards a intersection that is starved of input. These actions lead to the entire network to become instable. To solve this problem the first step was to detect these situations. This is done by defining the amount of work in a system. The speed at which a direction can get served is an indication of how fast it executes its work (cars crossing the intersection). As all cars entering the network are also detected it is also known how fast the work in a network increases. For a network to be stable the amount of work done by each intersection should be equal to or bigger than the amount of work entering the system. Secondly, the network can be further divided into modes, with each mode describing the actions of individual intersections and the total amount of work executed by it. For a system to be stable it should always remain in a mode which decreases the amount of work in a system. Thus, a distinction can be made between stable modes and unstable modes. Thirdly and lastly, a supervisor was created for the Kumar-Seidman case. This supervisor detects the modes the network is going to be in. If the network is about to reach an unstable mode it intervenes in the local Smart Traffic instances. By blocking this possible realisation it forces Smart Traffic to execute a different, local sub-optimal, realisation. This approach was executed and tested on the Kumar-Seidman network and resulted in stable network behaviour. This approach is particularly useful as it prevents unstable behaviour before it occurs. It does limit the local Smart Traffic instances in its available realisations but losing local optimization is a small price compared to global stability.

7.2 Recommendations

The recommendations for this project are split up in two parts. First recommendations which relate to the traffic dynamics of the model are given. Secondly there are several recommendations that focus on Smart Traffic and which aspects need additional research before it can be unrolled on the street.

In the model, arrivals occur according to a Poisson distribution and/or platoons but the average arrival rate remains constant. This rate is manually set to test the performance of Smart Traffic with different intensities but one single simulation does not handle changing arrival rates over time. As seen in the introduction, arrival rates fluctuate throughout the day. A good additional test for Smart Traffic is to simulate the arrivals over 24 hours with changing rates to determine its performance throughout the day. Additionally, the considered intersections are limited in size. Increasing the number of arriving traffic flows and conflicts in the lane increases the complexity of the calculations performed by Smart Traffic. Additional road-users can also be implemented in these traffic flows. By increasing the complexity, additional insights can be formed about Smart Traffic. For example, in situations were a lane with low intensity has many conflicts with other lanes. Due to the conflicts the lane is only seldom scheduled and due to the low intensity it takes a long time before the total waiting time of the lane is high enough for it to become a priority in being scheduled. While this is exactly how Smart Traffic is supposed to perform, it leads to long waiting times. By increasing the complexity of the simulation these situations can be found and adaptations in Smart Traffic can be made.

An important aspect of the current Smart Traffic model is that it registers all cars in advance and knows their destinations (does a car turn left or crosses the next intersection). In reality only the historical destinations are known which does not define the destination and route of individual cars. Additionally cars may enter the network through sideroads were no sensors are presented. These changes in the traffic situation cause the forecasting step of Smart Traffic to be inconsistent with the actual traffic situation, meaning the scheduled green phases might not be optimal anymore. The inconsistency can be partly tackled in a last second optimization; green phases are scheduled in a regular way but can be extended, or cut off, when a queue has dissolved faster or slower than predicted. This adaptation means that the green phases scheduled afterwards are also shifted. The influence of this shift on the overall performance is not yet known. Additionally, Smart Traffic still needs to update the detected traffic situation to ensure correct behaviour. As the traffic situation

is different than what Smart Traffic is predicting, the resulting performance most likely drops. The exact influence on performance and how to adapt Smart Traffic to be more robust is a topic of future research.

As seen in this report, by determining the amount of work in a network and by using efficient modes, stable behaviour can be obtained. Stability in the Kumar-Seidman network was obtained by never allowing the network to be in an inefficient mode. However, it is possible that a network can be in an inefficient mode for a short amount of time, as long as the growing amount of work during this mode is evened out by efficient modes. Therefore, by applying a less strict supervisor which allows inefficient modes, the network behaviour can remain stable. The constraints necessary for such a supervisor are not known and require more research.

Bibliography

[1] CROW Verkeer en Vervoer. Verkeer in steden dreigt vast te lopen. December 2016.

[2] CBS Centraal Bureau voor de Statistiek. Verkeerintensiteit; rijkswegen, provinciale wegen, landsdelen. December 2012.

[3] P.R. Kumar T.I. Seidman. Dynamic instabilities and stabilization methods in distributed real-time scheduling of manufacturing systems. *IEEE Transactions on Automatic Control*, 35(3):289–298, 1990.

[4] D. van Zwieten. *Fluid Flow Switching Servers*, volume 1. Universiteitsdrukkerij Technische Universiteit Eindhoven, 2014.

[5] S.T.G. Fleuren E. Lefeber. Optimizing fixed-time control at isolated intersections. part i: A single green interval per traffic light. Intern Report DC 2016.067, Eindhoven University of Technology, Dynamics and Control Group, Department of Mechanical Engineering, Eindhoven, The Netherlands, 2016.

[6] S.T.G. Fleuren E. Lefeber. Optimizing fixed-time control at isolated intersections. part ii: Optimizing the number of green intervals. Intern Report DC 2016.068, Eindhoven University of Technology, Dynamics and Control Group, Department of Mechanical Engineering, Eindhoven, The Netherlands, 2016.

[7] R. Akcelik. *Traffic signals: capacity and timing analysis*. Australian Road Research Board, 1981.

[8] R. Cowan. An improved model for signalised intersections with vehicle-actuated control. *Journal of Applied Probability*, pages 384–396, 1978.

[9] A. Wilson. *Handboek verkeerlichtenregelingen 2014*. Number 343. CROW, 2014.

[10] L.A. Safonov E. Tomer V.V. Strygin Y. Ashkenazy S. Havlin. Multifractal chaotic attractors in a system of delay-differential equations modeling road traffic. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 12(4):1006–1014, 2002.

[11] S. Wiggins. *Introduction to applied nonlinear dynamical systems and chaos*, volume 2. Springer Science & Business Media, 2003.

[12] H.G. Schuster W. Just. *Deterministic chaos: an introduction*. John Wiley & Sons, 2006.

[13] M.S. van den Broek J.S.H van Leeuwaarden I.J.B.F. Adan O.J. Boxma. Bounds and approximations for the fixed-cycle traffic-light queue. *Journal of Transportation Science*, 40(4):484–496, 2016.

[14] L. Moormann. Expected delay at fixed cycle traffic light queues. Bachelor Final Project DC2016.050, Eindhoven University of Technology, Dynamics and Control Group, Department of Mechanical Engineering, Eindhoven, The Netherlands, April 2016.

[15] M. Fellendorf. Vissim: A microscopic simulation tool to evaluate actuated signal control including bus priority. In *64th Institute of Transportation Engineers Annual Meeting*, pages 1–9. Springer, 1994.

[16] R.E. Allsop. Estimating the traffic capacity of a signalized road junction. *Transportation Research*, 6(3):245–255, 1972.

[17] Dirk S. Lämmer D. Helbing. Self-control of traffic lights and vehicle flows in urban road networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(04):P04019, 2008.

[18] R.J. Evans A.V. Savkin. *Hybrid dynamical systems: controller and sensor switching problems*. Springer Science & Business Media, 2002.

[19] Dirk S. Lämmer R. Donner D. Helbing. Anticipative control of switched queueing systems. *The European Physical Journal B*, 63(3):341–347, 2008.

[20] Dirk S. Lämmer D. Helbing. Self-stabilizing decentralized signal control of realistic, saturated network traffic. Citeseer, 2010.

[21] E. Lefeber J.E. Rooda. Control of reentrant manufacturing system with setup times: the kumar-seidman case. *SE-Report Nr. 2006-04*, 35(3), 2007.

[22] M.J. Lighthill G.B. Whitham. On kinematic waves. ii. a theory of traffic flow on long crowded roads. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 229, pages 317–345. The Royal Society, 1955.

[23] M. Papageorgiou. Some remarks on macroscopic traffic flow modelling. *Transportation Research Part A: Policy and Practice*, 32(5):323–329, 1998.

[24] D. Helbing M. Treiber, A. Kesting. Delays, inaccuracies and anticipation in microscopic traffic models. *Physica A: Statistical Mechanics and its Applications*, 360(1):71–88, 2006.

[25] D.Q Mayne J.B Rawlings C.V. Rao P.O.M. Scokaert. Constrained model predictive control stability and optimality. *Automatica*, 36(6):789–814, 2000.

[26] C.B. Alba E.F. Camacho. *Model predictive control*. Springer Science & Business Media, 2013.

Appendix A

Smart Traffic road-map

A step-by-step approach of Smart Traffic for a local intersection is given below. It consist of forecasting the arrivals and determining how these are best handled by the system. The general parameters of the model are given in Chapter 3, the general working of Smart Traffic is expressed in Chapter 4.

Consider the intersection as given in Figure 4.1 with the following settings:

- Fixed clearance time, τ
- All arrivals at stopline for all lanes are known, $Q_i^{arr}(t)$
- No maximum allowed green time, $t_{maxG} = \infty$
- Green phase can end when a hiaat occurs (3 seconds without arrival), h
- The queue length evolution $n_i(t)$ of the lanes at t is:

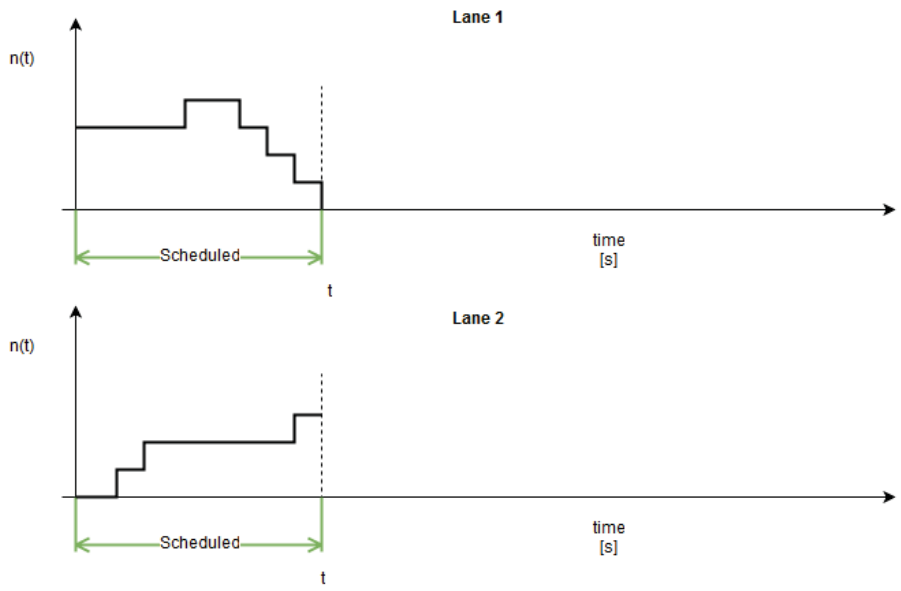

Figure A.1: Queue evolution over Lane 1 (green) and Lane 2 (red) until time t. At t Smart Traffic executes a new forecast to determine the optimal schedule

.

At time t the green period is scheduled to end. There are two possible actions for Smart Traffic to chose from:

- Switch the lights so Lane 2 can get served
- Extend the green phase for Lane 1, no trafficlight changes

Each action has its required green time g and waiting time per lane $w_i(t)$. This gives the total costs for each action. How the queue n evolves under both actions is shown in the upcoming sections.

Switch green phase

In case the trafficlight switch $n_i(t)$ evolves as shown in figure A.2.

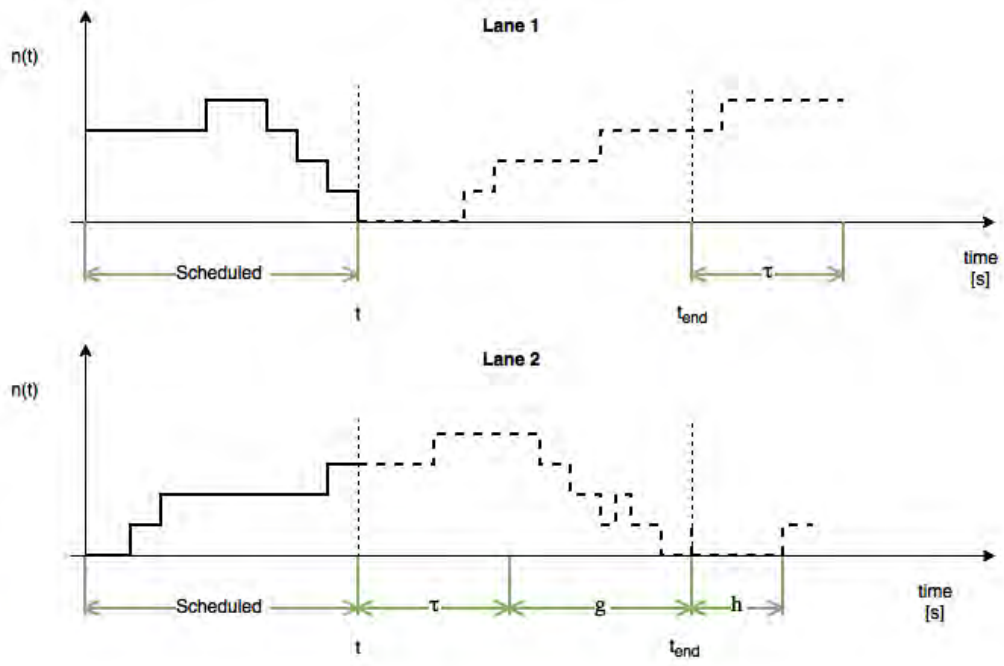

Figure A.2: Queue evolution $n(t)$ over two lanes with internal conflict when light switches at t. Including clearance time τ, green period g and hiaat time h. The time before lane 1 can become green again is an additional τ to take switching back into account

.

The total waiting time at the end of the green period are zero for lane 2. For lane 1 the waiting time is determined up until $t_{end} + \tau$. The total incurred waiting time for each car becomes larger (due to set-up and saturation flow losses). These extra costs are not taken into account as they lie too far into the future.

Extend green phase

In case the green phase extends, until at least one extra departure has occurred, the queue evolution over the system is shown in figure A.3.

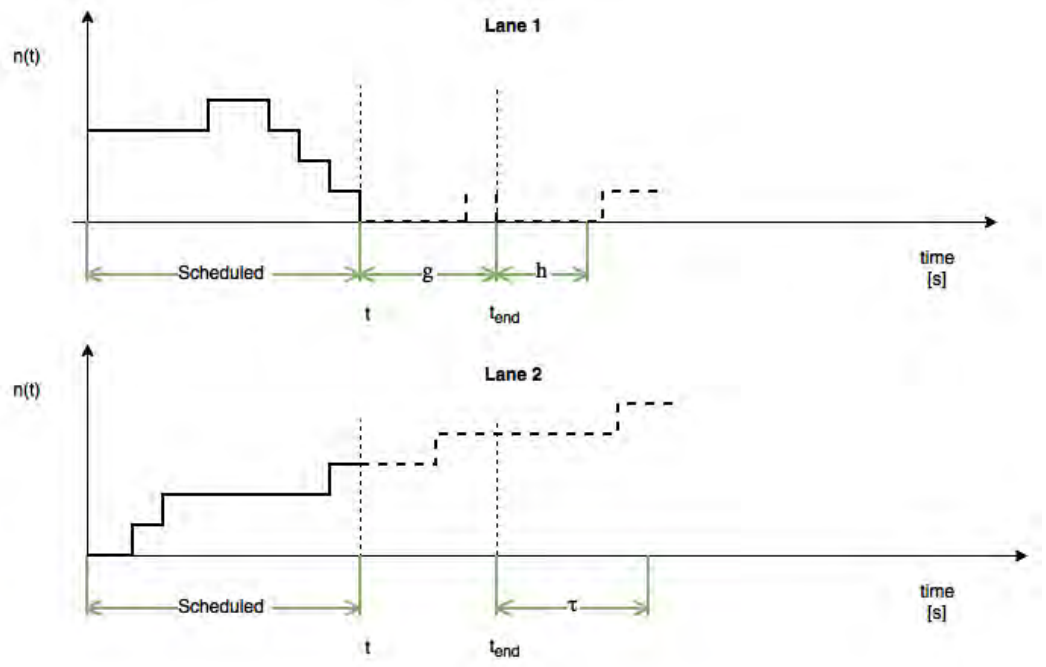

Figure A.3: Queue evolution $n(t)$ over two lanes with internal conflict with extended green phase at t. Extension takes up green period g until an hiaat h. The time before lane 2 can become green is an additional τ to take switching time into account

.

When extending the green phase of lane 1 it is possible that multiple cars leave the intersection before an hiaat is found. Especially when platoons are arriving extending is profitable as platoons pass without stopping. The total waiting time of lane 2 is the area under $n(t)$ from *the first arrival in the queue* till $t_{end} + \tau$.

Optimization step

The last step Smart Traffic executes is comparing Figure A.2 with Figure A.3 and determine which action will incur the least waiting time using (4.9) and (4.10) (presented in Chapter 4). From this example it is clear that switching the trafficlight is the best option.

Another possible situation would be that 5 cars arrive in platoon fashion at Lane 1 just after t. With only one, recently arrived, car waiting at Lane 2. Smart Traffic, unlike any other trafficlight control strategy, guarantees that the platoon passes without stopping. Additionally for situations were the difference between switching or extending green phases is much less obvious Smart Traffic optimization strategy ensures that the waiting time is as low as possible.

Appendix B

Smart Traffic Test Results

Below follow the results of Smart Traffic on a local intersection in comparison to a FTS and a VAC. The intersection is shown in figure 4.1 and fixed values are taken from Table4.1. The test variable are arrival intensity $\lambda_i(t)$ and platoons. We are interested in the mean waiting time and how many cars cross the intersection without stopping. Simulation duration is 10 hours and every tests is executed 10 times.

- $\lambda_1 = 500$ cars/h, no platoons
- $\lambda_2 = 500$ cars/h no platoons

		Mean Waiting time [s]	Non-Stopping [%]	Relative Mean Waiting time [%]	Relative Non-Stopping [%]
FTS	L1	9.4	25.90	100	100
	L2	9.4	25.83		
	Mean	9.4	25.88		
VAC	L1	7.94	11.76	87.2	45.2
	L2	7.96	11.68		
	Mean	7.95	11.72		
ST	L1	7.38	9.50	81.1	36.4
	L2	7.41	9.42		
	Mean	7.40	9.48		

- $\lambda_1 = 500$ cars/h, with 80% platoon arrivals
- $\lambda_2 = 500$ cars/h no platoons

		Mean Waiting time [s]	Non-Stopping [%]	Relative Mean Waiting time [%]	Relative Non-Stopping [%]
FTS	L1	10.31	29.07	100	100
	L2	8.73	27.73		
	Mean	9.52	28.4		
VAC	L1	7.83	24.41	82.9	63.7
	L2	7.96	11.76		
	Mean	7.89	18.09		
ST	L1	3.73	47.59	51.6	131.5
	L2	6.08	27.09		
	Mean	4.91	37.34		

- $\lambda_1 = 500$ cars/h, with 80% platoon arrivals
- $\lambda_2 = 500$ cars/h, with 80% platoon arrivals

		Mean Waiting time [s]	Non-Stopping [%]	Relative Mean Waiting time [%]	Relative Non-Stopping [%]
FTS	L1	10.27	29.14	100	100
	L2	9.14	33.89		
	Mean	9.71	31.5		
VAC	L1	7.28	17.57	82.4	43.9
	L2	8.71	10.09		
	Mean	8.0	13.83		
ST	L1	4.60	55.92	41.9	181
	L2	3.54	58.05		
	Mean	4.07	57.0		

- $\lambda_1 = 800$ cars/h (saturated), no platoons
- $\lambda_2 = 800$ cars/h (saturated), no platoons

		Mean Waiting time [s]	Non-Stopping [%]	Relative Mean Waiting time [%]	Relative Non-Stopping [%]
FTS	L1	10.58	14.64	100	100
	L2	10.53	14.38		
	Mean	10.56	14.52		
VAC	L1	11.87	8.94	110	61.6
	L2	11.84	8.97		
	Mean	11.86	8.95		
ST	L1	10.79	7.66	101.9	52.1
	L2	10.72	7.46		
	Mean	10.76	7.56		

- $\lambda_1 = 800$ cars/h (saturated), with 80% platoon arrivals
- $\lambda_2 = 800$ cars/h (saturated), with 80% platoon arrivals

		Mean Waiting time [s]	Non-Stopping [%]	Relative Mean Waiting time [%]	Relative Non-Stopping [%]
FTS	L1	10.78	22.77	100	100
	L2	10.41	28.27		
	Mean	10.60	25.52		
VAC	L1	13.89	3.64	131	12.4
	L2	15.49	3.15		
	Mean	14.69	3.39		
ST	L1	7.09	34.86	61.9	129.7
	L2	6.62	31.33		
	Mean	6.88	33.11		

Appendix C

Feeding Network

Consider the small network of two intersections given in Figure C.1.

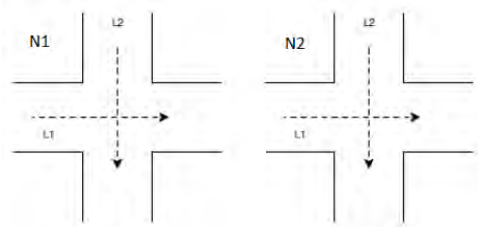

Figure C.1: Two intersection which are linked through Lane 1. Each intersection is controlled by its own Smart Traffic instance. Distance between intersections may vary

.

The arrivals, at the edge of the network, are known in the same manner as a local intersection. The traffic that arrives at N2 from L1 has already been processed by N1. This leads to several changes in comparison to a purely local intersection, these changes are briefly explained below.

- The travel distance between networks is from stopline to stopline. So the travel distance is width of the intersection and the downstream road section.
- Forecasted departures from N1 (upstream) are presented as arrivals to N2 (downstream). This means N2 knows its arrivals before cars have even left intersection N1.
- A forecasted departure is matched with the actual departure as soon as it occurs. This is done to prevent mismatching in case the predicted (forecasted) departures occur at a different time or not at all.
- The travel time between intersections can be larger than the forecast horizon (30 s). Upstream forecasted departures are not taken into account as they fall beyond the horizon. Only actual departures (upstream) are shown in the downstream intersection.
- The optimization strategy of Smart Traffic is equal to the strategy as presented in Chapter 4. The network uses a decentralized strategy.

This network control strategy uses a decentralized strategy, this as opposed to a centralized strategy. A centralized strategy would mean one Smart Traffic instance for the complete network. As a typical network consist of many more intersections and lanes a centralized optimization strategy becomes more complex and that much harder to find the optimum. Performing this task on-time (every 0.1s) requires more computing power than currently available.

Simulating Figure C.1 with arrival rate $\lambda_i = 600$ car/h, average speed of 50 km/h and distance between intersections of 500 m (36 s travel time) the queue length evolution $n_i(t)$ is shown in Figure C.2.

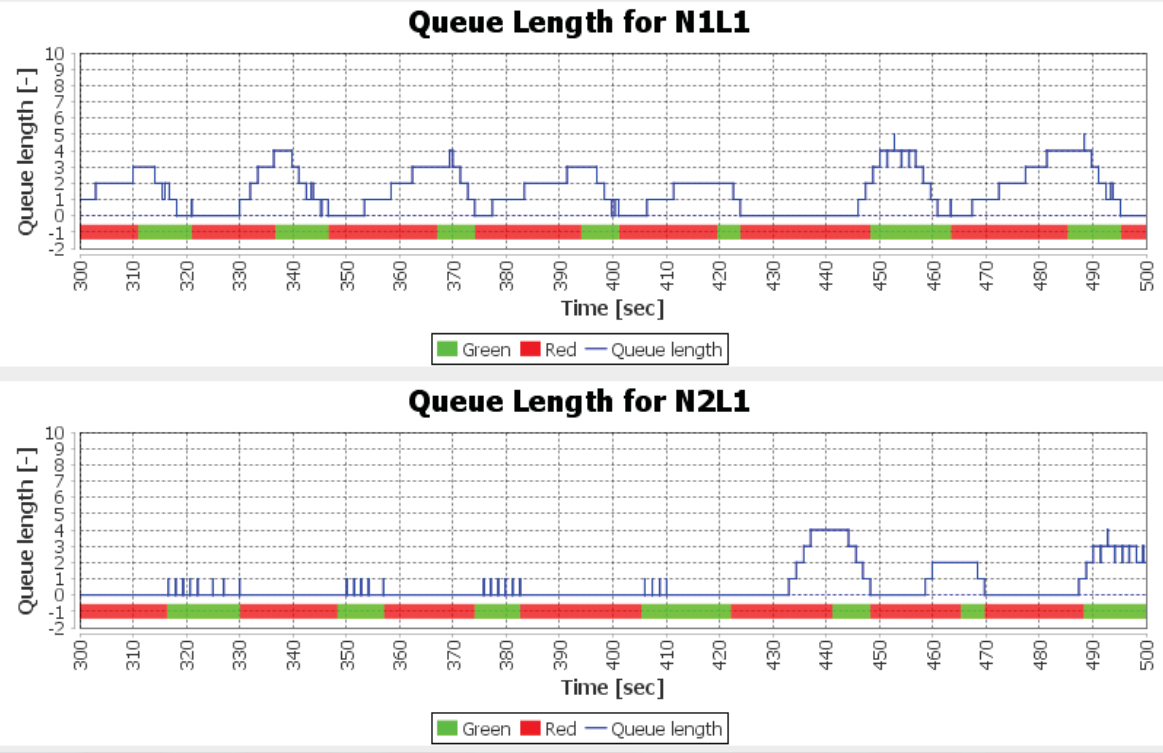

Figure C.2: Queue evolution of the first intersection with random arrivals and the second intersection which receives cars from the first intersection

The arrivals into the system occur at random without any platoons. At the first intersection around 85% of cars are forced to stop before they leave. When they leave from a queue they continu driving as a platoon and arrive as such at the next intersection. Stopping a platoon is much more costly than individual cars so platoons can often keep driving in a green-wave. This happens for over half of the cars. If platoons are halted it is often caused because there is no hiaat for the light to start the switching procedure. Having one lane of an intersection serving mostly platoons also decreases waiting time for the other lane. Lane L1 only needs a short green time, giving more green time to Lane L2. Overall results are shown in table.

Table C.1: Mean waiting time and percentage of cars which do not incur stops in a simple network. The individual lanes of Intersection 1 have the same results as they are identical

	Waiting time [s]	Non-stopping [%]
N1L1/2	7.65	15
N2L1	5.36	43
N2L2	6.98	20

These results are to be expected as serving platoons is being done at the saturation flow and often without incurring any set-up delay. This example also shows that Smart Traffic is best used in a network of trafficlights (typically in a city) as this will lead to most platoons which then are served by local green-waves.

Appendix D

Kumar-Seidman Supervisor example

In this chapter we compare the proposed supervisor against a local clearing policy. First the local clearing policy is used on the Kumar-Seidman network which causes the system to be unstable. Afterwards the same clearing policy with the supervisor shows stable results.

Local Clearing policy

Assume the system given in Figure 5.3 with the following settings:

- $\lambda_1 = \lambda_3 = 1$
- $\mu_1 = \mu_3 = 1/0.3$
- $\mu_2 = \mu_4 = 1/0.6$
- $\rho_i = \lambda_i/\mu_i$

With these settings and with equation 5.1 it shows that the system has enough capacity to exhibit stable behaviour. The system follows a local clearing policy; Jobs are served from a queue until it is empty. Once the queue is empty the machine switches to serve a non-empty queue or serve at λ_i if no such queue exist.

Suppose the initial system at t_0 is empty save for x_1 which is filled with $n_1(t_0)$ jobs. Clearing this buffer will take $t_1 = n_1(t_0)/(\mu_1 - \lambda_1)$ after which $t_1 \cdot \mu_1$ jobs have arrived in x_2. As $\mu_2 < \mu_1$ machine B starts serving buffer x_2 with rate μ_2 from t_0 until it is empty at t_2. The expression for $t_2 = n_1(t_0)/(\mu_2 - \lambda)$, this can be rewritten into:

$$n_1(t_0) \cdot \frac{1/\mu_2}{1 - \rho_2} \tag{D.1}$$

Between t_1 and t_2 machine A is serving products with λ_1 with buffers x_1 and x_4 empty. At t_2 Machine B has finished clearing buffer x_2 and a queue has been building at x_3, this queue size is:

$$n_3(t_2) = t_2 \cdot \lambda \tag{D.2}$$

Combining D.1 and D.3 gives the following expression for the queue size:

$$n_3(t_2) = n_1(t_0) \cdot \frac{\rho_2}{1 - \rho_2} \tag{D.3}$$

At t_2 the system is symmetrical to its initial position; All buffers are empty except for x_3. In case $\frac{\rho_2}{1-\rho_2} > 1$ the total queue size in the system has grown and, with equation 5.4, it is shown that the amount of work has also grown. This means the system is unstable for any $rho_2 > 0.5$. Per the

assumed settings ($\rho_2 = 0.6$) this implies that the amount of work in the system grows, thus the system is unstable. This behaviour is shown in figure D.1.

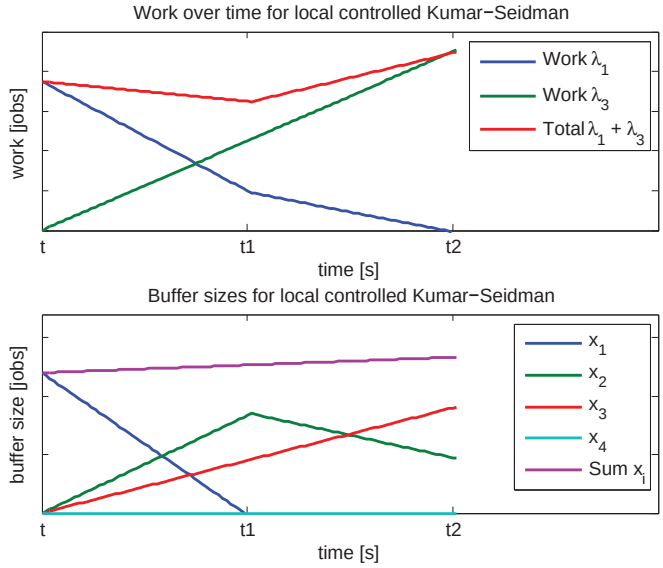

Figure D.1: Work and queue evolution for a local controlled Kumar-Seidman network. The work is denoted per flow λ_i and the total amount of work while all queue sizes are displayed per queue and the total queue size

.

The system switches between the following modes:

$$\begin{array}{lll} (\mu_1, \mu_2) & \text{for} & [t_0, t_1] \\ (\lambda_1, \mu_2) & \text{for} & [t_1, t_2] \end{array}$$

The figure shows that at the initial position there are an arbitrary number of jobs at x_1. At t_2 the amount of work as well as the amount of jobs have increased with the system being in a mirrored position. As both flows are identical this proves instability in case a local clearing policy is used.

Supervisor control

To prevent the system to become unstable we want to prevent the system to reach any mode were the work does not decrease. We use the same system and initial position as in the previous section but with the proposed supervisor. For this system and by using Table 5.2 four modes are identified which ensure that the work decreases. By using the supervisor of Figure 6.1 the system should always be in one of these four modes.

Applying the supervisor leads to following work and queue evolution:

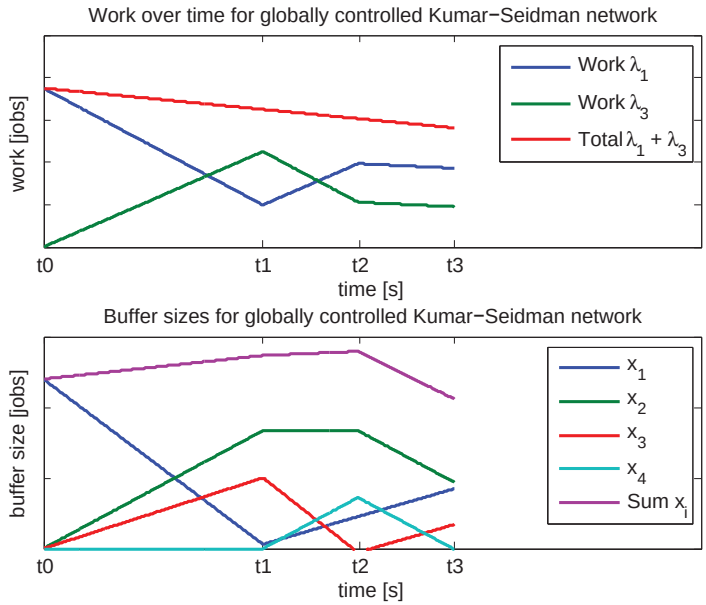

Figure D.2: Work and queue evolution for the Kumar-Seidman network with a global stabilization policy.

Figure D.2 shows that the amount of work continuously decreases over time. In certain modes the total queue size still grows over time but after one cycle (t_0, t_3) the overall queue size has decreased. The modes the system is in are:

$$\begin{array}{rll} (\mu_1, \mu_2) & \text{for} & [t_0, t_1] \\ (\lambda_1, \mu_2) & \text{for} & [t_1, t_1] \\ (\lambda_1, \mu_3) & \text{for} & [t_1, t_1] \\ (\mu_4, \mu_3) & \text{for} & [t_1, t_2] \\ (\mu_4, \mu_2) & \text{for} & [t_2, t_3] \end{array}$$

Note that this t_2 is not the same as in the local controlled system. Also after t_3 the system is again in mode (μ_1, μ_2). There are 2 actions the supervisor performs, both on t_1, this action causes both machines to switch and ensures they both perform at the optimal rate. After the supervisor has acted the control is handed back to the local clearing policy. With the supervisor the system shows stable behaviour. This proves the working of the supervisor.

Appendix E

Model build

The working of the algorithm is explained in Chapter **referienties!!!**, the exact lay-out of the model used throughout this report is explained in this Appendix. Specifically the methods used and how these are linked to one another. We first discuss the general lay-out of the model, secondly individual lanes with the appropriate driver behaviour is explained as well as how they respond to different traffic lights. The build-up of each lane is identical and multiple, conflicting, lanes make up an intersection. We show how the model is expanded under different traffic light controllers; FTS, VAC and Smart Traffic. Lastly we apply the network supervisor of chapter **referienties!!!** across multiple intersections.

General build

Smart Traffic makes use of model predictive control (MPC) [25] [26] to determine which direction to serve next. The MPC approach is used throughout many different field of expertise as a control and optimization strategy but for traffic management this approach is new. A schematic representation of MPC applied for traffic management is given in Figure E.1.

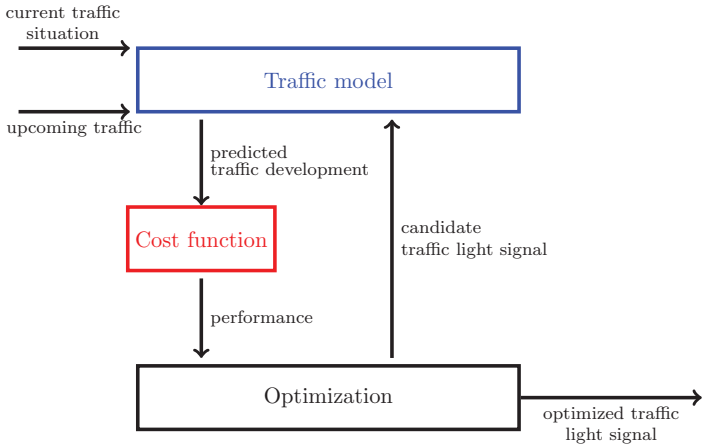

Figure E.1: Schematic overview of MPC applied for traffic management

Looking at Figure E.1 both outside inputs are known from sensors in the road or possible other data sources.A traffic model uses these inputs to predict how the traffic situation develops in the future under different traffic light settings. The performance of each prediction is measured by a cost function. Performance is often defined as the incurred waiting time for the traffic but may also be defined as CO2-output, number of stops or other. Once the resulting performance of all possible candidates are know the candidate with the best performance (in these cases lowest overall waiting time) is executed by the traffic lights. The accuracy of this approach is dependent on the availability of incoming data as well as a traffic model that closely represents the traffic behaviour. As cars enter and leave a road network the available data is constantly updated as such the traffic model

has a limited horizon for its predictions. It can be predicted how long a car need to drive from one intersection to the next but it becomes much harder to predict the position of a single vehicle when it crosses multiple intersections and might take a turn at any of them. As such the optimized traffic light signal is only one or two green phases long. After this time the current traffic situation has changed and the steps of MPC are retaken to create a new optimized signal.

First the model needs several input variables to run, these input variables can be loaded into the model through Excel or using manual input. Before any simulation starts, frames open that let the user set all variables or use the default values. This way many different intersections can be created and quickly tested. The user can chose to test, and compare, the created intersection under multiple traffic light controllers. At this point these screens only work for local controlled intersections. Networks of intersections can not be manually set and require extra coding. The required input variables are given in Section 3.8

Lane build

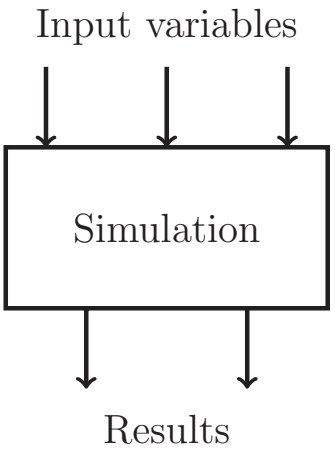

Figure E.2: Black-box simulation

Looking at the black-box of Figure E.2 we first consider a single lane with a traffic light, we expand this overview to create the complete model. In this example the traffic light is controlled by a FTS, this means only information (light settings) is sent towards the lane. No information from sensors are send towards the traffic light controller.

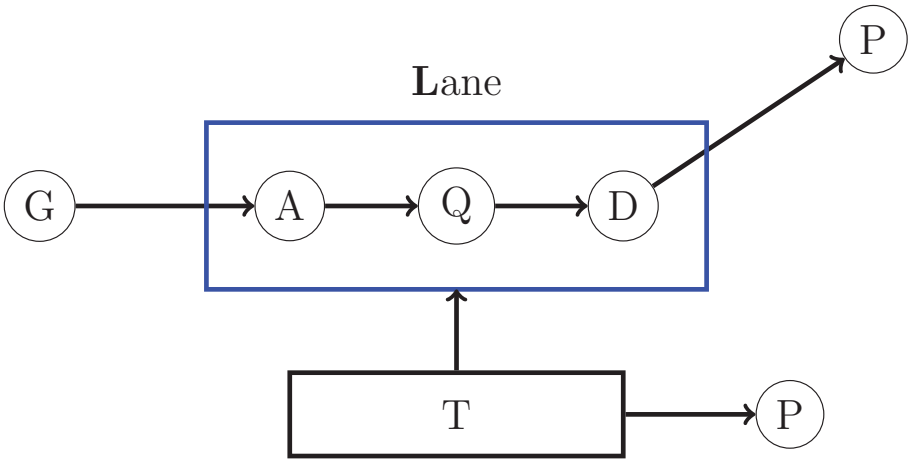

Figure E.3: Single lane actions

In Figure E.3 each car is generated by process G, it arrives in the lane at process A. At this point the car drives over a sensor and is seen by the system, it receives an unique ID. In this example this information is not used by the traffic light controller and the car keeps driving until it joins the queue Q. As soon as it reaches the front end of the queue and the traffic light T is green it can depart D. At this point it leaves the lane, and in the case of single intersection, it leaves the system. As it leaves its information (time of entry, time in queue, time of departure) is logged P and used by post-processing after the simulation has completed. The same holds for light settings generated by the traffic light process T, each time the light changes color these times are also logged in process P.

Note that the entire model uses discrete events meaning processes only become active when an event occurs, this also means only one process can be active at the same time. This improves simulation speed as not every process needs to perform a calculation each time-step. It is known from every car that is processed by the arrival process when it reaches the stop line (and joins the queue), no process become active or car-position is tracked in between these processes. So while each car goes through the same three processes in a lane, these process do not necessarily occur in sequence as additional arrivals or departures may occur.

```
1  //All possible events CARS can do
2  double actionLane(int ID) {
3      if (event.isEmpty()){return -1;}            //throw nullpointer, no events in lane
4
5      double x = event.remove();                  // see time of first event
6
7      // Car arrives in system
8      if (arrivals.contains(x)){
9          //car starts traveling down system at time x
10         //add travel time to lists of all cars traveling in lane
11         traveling_cars.add(x+travel_time);
12         event.add(x+travel_time);
13         data.collectData("Arrival",x);          //log event (arrival in lane)
14         arrivals.remove(0);                     //remove from arrivals
15
16     // Car leaves intersection, no stopping
17     } else if(traveling_cars.contains(x) && queue == 0){
18         data.collectData("Join Queue",x);       //log event (start waiting time)
19         data.collectData("Leave system",x);
20         traveling_cars.contains.remove(x);      //car is done travelling
21
22     // Car joins in queue
23     } else if (travelling_cars.contains(x)) {
24         queue++; //queue grows
25         traveling_cars.contains.remove(x); //car is done travelling
26         data.collectData("Join Queue",x);   // log event (start waiting time)
27
28     // Car leaves intersection from queue
29     } else if (leave_lane.contains(x)){
30         if (queue>0){
31             queue--; //shorten queue
32             data.collectData("Leave system",x); // log event (end waiting time)
33         }
34     }
35     //time of next event in queue
36     // at this time lane becomes active again unless light settings changes
37    return event.peek();
38 }
```

Departures can only occur when the light turns green. As soon as the lane receives a signal that the light turns green a list of possible departures time is generated. As each car behaviour is identical the exact departure times from a queue are also known. Any car that is in the queue departs according to this list until the queue has dissolved. As soon as it has and the light remains green cars do not have to stop anymore and leave the lane without stopping.

```
1  double actionLane(int ID, Trafficlight.Light state, double t0){
2      //All possible events TRAFFICLIGHT can cause
3         // Green light
4         if(state.equals(GREEN)){
5             leave_lane = queue.leaving_instances(t0,mu);// time instances cars can leave
6             event.addAll(leave_lane); // add instances to list of events
7             data.collectData(state, t0); //log event (start green time)
8
```

```
9           // Red light
10          }else if(state.equals(RED)){
11              event.removeAll(leave_lane); // remove time instances since no cars can leave
12              data.collectData(state, t0); //log event (end green time)
13          }
14          return event.peek(); //time of next event in queue
15      }
```

Intersection build

Save for the input variables each lane consist of the same processes, as such an arbitrary number of lanes can be generated. The traffic light controller assigns green phases across all lanes

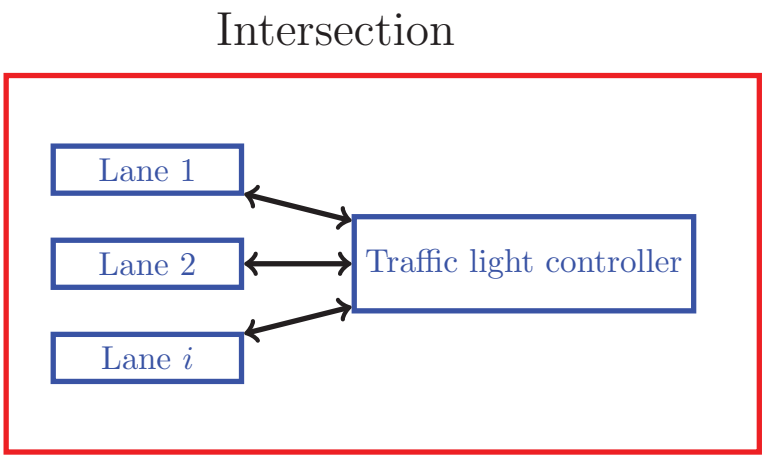

Figure E.4: Multiple lane instances are created to model each direction of an intersection

For a FTS the traffic light control only sends information towards the traffic lights positioned on each lane. For a VAC sensor information is used about upcoming arrivals (Arrival process of each lane) and the availability of a queue (Queue process of each lane). This sensor information enables the VAC to change the green phases accordingly. The appearance or dissolvement of a queue is pushed towards the controller, meaning a signal is send from the lane to the controller if changes in the queue occur.

```
1
2       void queue_availabilityk(int id, boolean CarsinLane, double t){
3           if (TS_list[id] == Light.RED){
4               if (CarsinLane == true){
5                   // cars are waiting the lane wants to turn green
6                   TS_list[id] = Light.WAITRED;
7                   String s = getKeyFromValueInt(M, id);
8                   L.put(s, t);
9                   t_action = Collections.min(L.values());
10                  //update next event (t_action) to try and put this lane on green
11              }
12          } else if (TS_list[id] ==Light.GREEN){
13              if (CarsinLane == false){
14                  // there is no queue left, the lane may turn red
15                  TS_list[id] = Light.WAITGREEN;
16                  String s = getKeyFromValueInt(M, id);
17                  L.put(s, t);
18                  t_action = Collections.min(L.values());
19                  //update next event (t_action) to check if putting this lane on red is:
20                  // a) needed as ending this green phase cause conflicts to dissolve and
21                  // b) frees up other lanes (with a queue) can get served
22              }
23          }
24      }
```

Smart Traffic build

Unlike the VAC Smart Traffic receives information about every process that occur in each lane. Each time a car enters a lane by riding over a sensor this information is send towards Smart Traffic. Similarly for each car that leaves the intersection. By knowing the arrivals and departures it knows how many cars are currently on the lane and how long they have been there. Smart Traffic can forecast the effect of each green phase on the traffic simulation and chose the option that leads to the minimal amount of waiting time.

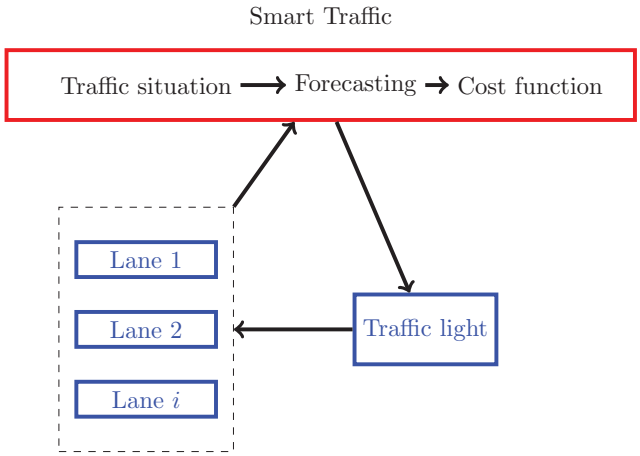

Figure E.5: Connection between Smart Traffic and traffic that is driving over the lane, providing information through sensor to Smart Traffic. Smart Traffic changes the traffic light to minimize the costs (waiting time)

Each forecast is done over all possible stages. These stages are all combinations of directions that can be served simultaneously. In this report we only consider small intersections of two different traffic flows (with 1 conflict) and there are only two stages to chose from. An intersection with multiple traffic flows has more stages to chose from but the overall calculation approach is the same.

When a schedule (of green phases) is about to be complete a new schedule is going to be needed. The forecasting process start and each stage requires the following information:

- When can the stage receive green, this is know out of the current schedule with respect to clearance times.
- What are the current and upcoming arrivals of the stage.

```
1   private void setarrivalList(int id, double time){
2       // update list of arriving car
3       List<Double> A_update = A.get(id); //current arriving time instance of cars
4       A_update.add(time+travel_time); //add extra arrival at stop-line !
5       A.set(id,A_update);
6   }
7
8   private void setDepartureList(int id, double time){
9       // update list of cars, first car has left the lane
10      List<Double> D_update = A.get(id); //current arriving time instance of cars
11      D_update.remove(); //remove first arrival at stop-line !
12      A.set(id,D_update);
13
14      List<Double> F_update = F.get(id); //list of forecasted departures scheduled to ↩
            take place
15      double F_departure = F_update.peek();
16      if(F_departure != time){
17          log[id].write("Unscheduled departure" + F_departure);
18      }
19      F_update.remove();
20      F.set(id,F_update);
21  }
```

By knowing its arrivals it knows how long a green phase needs to be scheduled to ensure all arriving cars can cross the intersection in the green phase. This green phase is shortened to stop when a gap occurs. All arrivals that occur after this point are excluded as they are not yet scheduled for departure. For this stage we know the total waiting time occurred by each car as the arriving time and departure time are known. While serving this stage all other directions do not receive a red light. Cars arriving, or already waiting at, these directions experience an increase in waiting time for the entire duration of this stage.

```
1     private void green_Stage(int id, double time){
2         double C = 0; //costs for this stage
3         //find first time that lane can turn green
4         //either ending other lanes or it already is green
5         double t_green = getGreentime(id);
6         List<Double> P = getArrivals(id); //Arrivals in queue over time
7         List<Double> D = getEndtime(id,P,t_green); // Schedulde departures of queue
8         D = getGap(D); // Departures that occur after a gap are excluded
9
10        if (D.size()>0){ //if any car has left the system
11            t_end = D.get(D.size()-1); //time at which this stage can end
12            Dt_count[id] =D.size(); //number of cars departing this stage
13            P = cleanArrivals(P,t_end); //arrivals after last leave are excluded
14            C = getGreenCost(D, P); //cost if you switch (sum of waiting time per car)
15        } else { //nothing leaves
16            t_end += clear_t;
17            C = Double.POSITIVE_INFINITY; //complete idle time
18        }
19
20        C += red_Stage(id, time, t_end); //cost of cars of other lanes that need to wait
21        //add cost and time to table;
22        Cost_table.get(Cost_table.size()-1).add(C);
23        Cost_table.get(Cost_table.size()-1).add(G);
24    }
```

We can now determine which stage has the least amount of total cost involved with it as we know the total incurred waiting time over the entire intersection and the duration of the upcoming stage. The stage with the least cost assigned to it made part of the schedule and received by the traffic light which executes it.

```
1  Trafficlight.Light[] controlLights(double t0, List<ArrayList<Double>> T_table,
2          List<ArrayList<Integer>> A_table){
3      //get next schedule from forecasting
4
5          t = t_action;
6          O = getKeyFromValue(L,t_action);
7          int ix = M.get(O);
8          //index of lane that is going to undertake an action
9
10         if (TS_list[ix] == Light.GREEN){ //Light is currently Green
11             int K = findForecast(t, ix, T_table); //Find next action
12             if (A_table.get(K).get(ix) == 1){ // If light remain green
13                 L.put(O, T_table.get(K).get(ix)); // Put new end green time
14             } else { //if lights switch
15                 TS_list[ix] = Light.RED; //put current lane on red
16                 L.put(O, infinity); //Current lane no actions needed
17
18                 Light [] Schedule = getNextAction(O, ix, K) // find what other lights ↩
                       need to do
19                 for(int i= 0; i<Schedule.size(); i++)
20                 {
21                     if (Schedule[i] = Light.CLEARRED){ //start clearance time
22                         TS_list[i] = Light.CLEARRED;
23                         O = getKeyFromValueInt(M, il);
24                         L.put(O, t+tau); //wait remaining yellowtime + clearance time ↩
                               before green
25                     }
26                 }
27             }
28         } else if (TS_list[ix] == Light.CLEARRED){ // Clearing time end
29             TS_list[ix] = Light.GREEN; // start green
30             int K = findGreentime(t, ix, T_table); //find green duration
31             L.put(O, T_table.get(K).get(ix));   //put green duration in place
32         }
33
34         //find new time at which traffic light becomes active
```

```
35          t_action = Collections.min(L.values());
36          return T_list;
37      }
```

Supervisor build

The upcoming green phase scheduled by the forecasting instance of Smart Traffic has a certain efficiency assigned to it. This is calculated by the supervisor. It is specificied for the Kumar-Seidman network and therefor very strict. Even small decreases in efficiency are not permitted. The point during the green phase in which the efficiency drops below the threshold (0.95) is found. Efficiency drops as soon as cars are processed at a rate below the saturation flow. At this point the supervisor needs to make sure that additional arrivals have occurred at the intersection so it can switch to a lane and serve it at saturation flow. Which switching is needed can also be found in Chapter 6.

```
1   import java.util.ArrayList;
2   import java.util.List;
3
4
5   //executed at the end of every Smart Traffic instance
6       public Object [] efficiency(List<Double> Active_ID, List<ArrayList<Double>> T_table,
7               List<List<Double>> Dt, List<List<Double>> A, String Intersection_Name){
8           Cost = C; //
9           light_state = T_table.get(T_table.size()-1);
10          Dt_table = Dt;
11          Dt_count = Dtc;
12          Death_count = Death_c;
13          Arrivals = A;
14          Lane_sizes = L;
15          force_switch = false;
16          double eff = 1;
17          int index;
18
19          if (light_state.get(0)){ //Next action would be to keep the light green
20              //extra departures during extension
21              if (!Dt_table.get(0).isEmpty()){
22                  // time of the scheduled departures
23                  List<Double> D = new ArrayList<Double>(Dt_table.get(0));
24                  double cars = (double) (D.size()-1);
25                  double timespend = D.get(D.size()-1)-D.get(0);
26                  double mu = Lane_sizes[id]/2.0;
27                  eff = (cars/timespend)/mu; //keeping light green gives this eff
28                  index = 0;
29              } //no departures so extening (always +3s) greenphase inefficient!!!!
30          } else {//Next action is to switch lights
31              // time of the scheduled departures
32              if (!Dt_table.get(1).isEmpty()){
33                  List<Double> D = new ArrayList<Double>(Dt_table.get(1));
34                  double cars = (double) (D.size()-1);
35                  double timespend = 0;
36                  double mu = (double) Lane_sizes[r]/2.0;
37                  eff = (cars/timespend)/mu; //switching lights gives this eff
38                  index = 1;
39              }
40          }
41
42          if(eff<0.95){ //in case eff is below 1
43              //find time at which efficiency drops
44              end_time_lane = efficiencybreak(index, Dt_table.get(index));
45              // provides time at which the index needs arrivals from other intersection↩
                    !!!!
46              Supervisor.determineMode(Intersection_Name, lane_change_id, end_time_lane);
47          }
48          return new Object []{force_switch, Dt_table,Dt_count,Cost};
49      }
50
51      //find possible break time
52      private void efficiencybreak(int id, List<Double> D)
53      {
54          int d_pos = -1; //which death is the last one before efficency drops
55          double t_pos = 0; //time of last efficient death
56          int id = -1; //id for light that is on low efficiency
57
58          for(int i =1; i<D.size(); i++){
59              //efficiency for every step
```

```
60              double eff = ((double) i/(D.get(i)-D.get(0)))/(Lane_sizes[id]/2.0);
61              if (eff<0.95){ //at this departure efficiency drops. Breakpoint
62                  d_pos = i-index;
63                  t_pos = D.get(d_pos-1+index);
64                  break;
65              }
66          }
67          return t_pos;
68      }
```

This give a point in time for the supervisor to ensure additional arrivals. This point in time is given to the supervisor at which it acts to switch the traffic lights of the other intersection. This action includes the travel time between the intersections.

```
1  import java.util.ArrayList;
2  import java.util.List;
3
4
5
6  // perform switch action (always from L2 to L1 as per mode graph)
7  // the lane is being efficient by serving L2,
8  // still to ensure the other intersection is efficient this lane switches.
9
10     //supervisorsender
11 static void determineMode(String Lane_name, int id, double t_end_time){
12     //find id of intersection that is going to switch
13         Intersection_ID= getIntersectionID(Lane_name);
14         Lane_ID= getLaneID(Intersection_ID, id);
15         // at this starting time
16         force_start_time  = t_start_green(t_end_time, id);
17
18         //send supervisoraction
19         I[Intersection_ID](Lane_ID,force_start_time)
20     }
21
22     // Smart Traffic receiver
23 public void linkforecast(int lane, double time){ //switch lanes due to supervisor actions
24     List<ArrayList<Double>> T_table_new = F.forceLaneAction(lane, time,indicate.getV());
25     Object K[] = ST.forcelaneAction(lane,time,T_table_new);
26 //  light_action = (double) K[1];
27 //  indicate.addtimestamp((String) K[0], light_action);
28 }
```

Appendix F

Roundabout settings

Large roundabouts with high traffic intensities are sometimes outfitted with traffic lights to better direct traffic. These lights are place right in front but also on the roundabout. Assuming a roundabout with 4 branches each branch can be seen as an intersections. Traffic flows either across the branch onto the intersection or is driving on the intersection and crosses the intersection to leave the roundabout. Each branch has its own set of constraints and conflict graphs as explained in Chapter 2. Below the constraints for a single branch as well as the roundabout are shown.

Roundabout, branch control

Assume a roundabout that has traffic lights directing the flow onto and on the roundabout. Each branch of the roundabout has a set of traffic lights which can be seen as an individual intersection. The traffic flows and conflict graph are shown in F.1

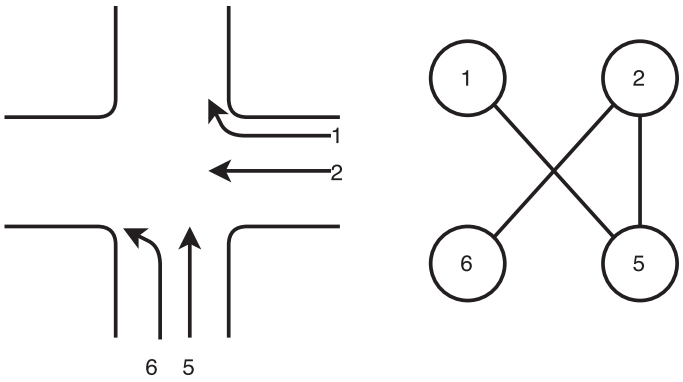

Figure F.1: Single branch of a roundabout with traffic flows and corresponding conflict graph. Direction 2 combines traffic that goes straight and turns left.

Looking at the conflict graph we define 3 sets (or modes) of lanes that can be green without conflicts. The fraction of time spend in these lanes should be equal to or smaller than one.

$$\alpha_{1,2} + \alpha_{5,6} + \alpha_{1,6} \leq 1 \tag{F.1}$$

Using the conflict graph we can identify 3 constraints

$$\begin{aligned} \rho_1 + \rho_5 &< 1 \\ \rho_2 + \rho_5 &< 1 \\ \rho_2 + \rho_6 &< 1 \end{aligned} \tag{F.2}$$

As long as these constraints hold the branch can show stable performance. The above constraints can be constructed for every single branch and combined into 1 roundabout giving a total of 12 constraints

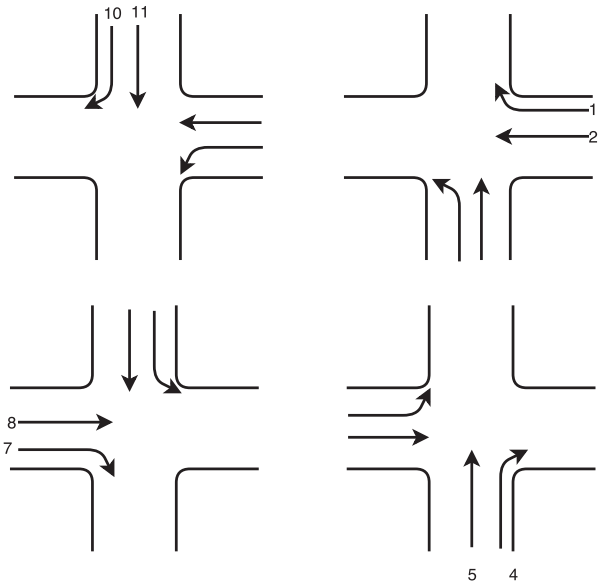

Figure F.2: Roundabout with all traffic flows

The internal traffic flows are a combination of external traffic flows. For the top right branch the traffic staying on the roundabout is equal to $\alpha_5 \cdot \rho_5$ with α_5 being the fraction of traffic that wants to turn left and enters through lane 5. In a similar fashion the traffic leaving the intersection is $(1 - \alpha_5) \cdot \rho_5 + \alpha_8 \cdot \rho_8$. With $\alpha_8 \cdot \rho_8$ the traffic that wants to turn left that entered from direction 8. Combining these expressions with equation F.2 lead to the following set of constraints

$$\begin{aligned} \rho_1 + (1 - \alpha_5) \cdot \rho_5 + \alpha_8 \cdot \rho_8 &< 1 \\ \rho_2 + (1 - \alpha_5) \cdot \rho_5 + \alpha_8 \cdot \rho_8 &< 1 \\ \rho_2 + \alpha_5 \cdot \rho_5 &< 1 \end{aligned} \tag{F.3}$$

These constraints are identical for each branch. No additional constraints for the system are needed.

Roundabout, System control

In addition to 4 branches the roundabout can also be expressed as 1 intersection. In this case traffic never stops on the roundabout but completely clears it before a conflicting lane receives green. This is shown in the figure below and leads to different constraints.

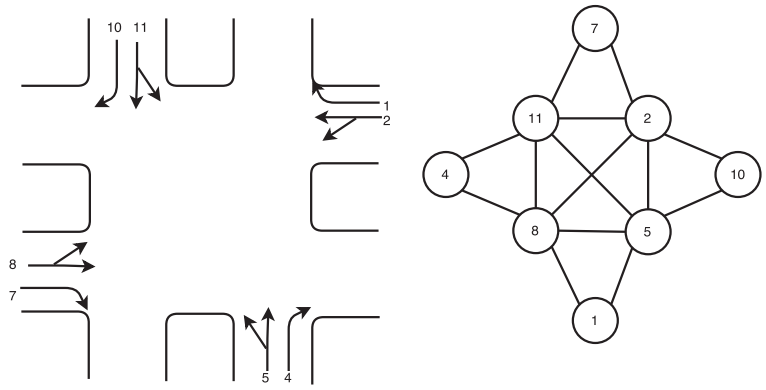

Figure F.3: Roundabout as a single intersection with traffic flows and corresponding conflict graph. Traffic going straight or turning left have the same conflicts.

Out of the conflict graph several constraints can immediately be found;

$$\begin{aligned} \rho_1 + \rho_5 + \rho_8 &< 1 \\ \rho_4 + \rho_8 + \rho_{11} &< 1 \\ \rho_2 + \rho_7 + \rho_{11} &< 1 \\ \rho_2 + \rho_5 + \rho_{10} &< 1 \\ \rho_2 + \rho_5 + \rho_8 + \rho_{11} &< 1 \end{aligned} \tag{F.4}$$

These constraints are more strict than the constraints given in F.3 as no traffic is allowed to stand still on the roundabout limiting the flexibility of the system. Furthermore there are only 5 modes for the system to be in, these can also be found out of the conflict graph, these modes are:

$$\alpha_{4,5,7} + \alpha_{1,2,4} + \alpha_{7,8,10} + \alpha_{1,10,11} + \alpha_{1,4,7,10} \leq 1 \tag{F.5}$$

Optimizing over these modes show that the constraints of F.4 are all necessary constraints. This shows that depending on how a network of trafficlights is directed different stability constraints are found. Good trafficlight control is necessary to allow the system to handle as much traffic as possible while remaining stable.

Appendix G

Berenkuil data

For the Berenkuil in Eindhoven the arrival intensities λ_i and saturation flow μ_i are known. These are known from case studies performed in 2006 by Sweco commissioned by the city council of Eindhoven. We assume the current traffic intensities of this roundabout remain similar to this day. Using the constraints determined in Appendix F we determine if the traffic flows of the Berenkuil can be handled by a trafficlight controller that controls each branch individually or the system as a whole. The arrival intensities are known for morning and evening rush-hour and given below.

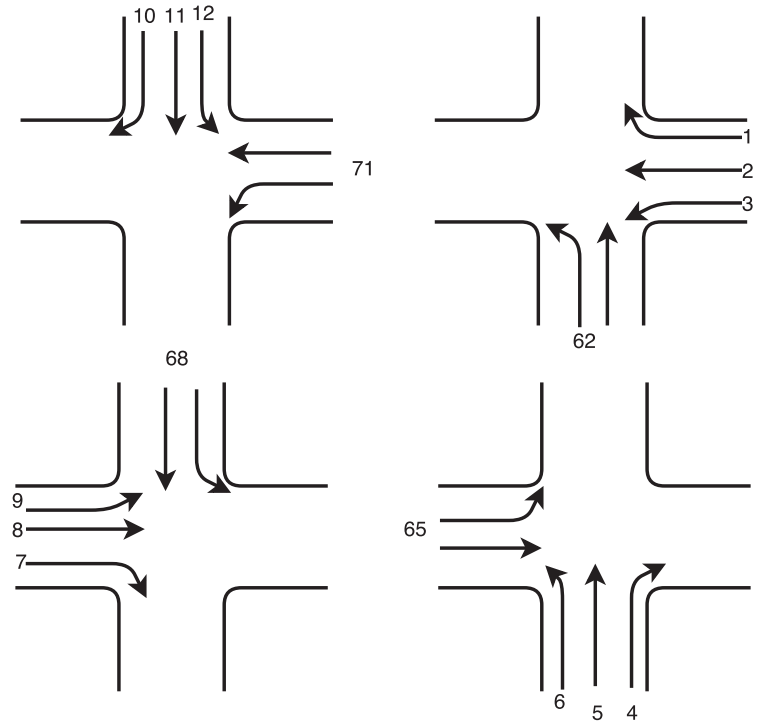

Figure G.1: The berenkuil with lane numbering, lanes 62,65,68,71 combine all feeding traffic flows.

Table G.1: Evening Rush-hour Berenkuil, traffic flows

Lane Nr	Intensity [Car/h]	Lane Nr	Intensity [Car/h]
1	805	7	289
2	212	8	846
3	179	9	60
4	530	10	14
5	1356	11	1436
6	180	12	784
62	1596	68	2399
65	1690	71	571

Table G.2: Morning Rush-hour Berenkuil, traffic flows

Lane Nr	Intensity [Car/h]	Lane Nr	Intensity [Car/h]
1	200	7	136
2	970	8	202
3	347	9	28
4	340	10	25
5	1508	11	1147
6	229	12	444
62	1765	68	1983
65	674	71	1547

Table G.3: Berenkuil Lane Capacity

Lane Nr	Intensity [Car/h]	Lane Nr	Intensity [Car/h]
1	2000	7	1700
2	4000	8	4000
3	1900	9	2000
4	3400	10	1700
5	4000	11	4000
6	1700	12	2000
62	5700	68	7400
65	3900	71	5600

Lane 1 does not have a traffic light and is not taken into account when calculating the constraints with the given traffic flows,

Branch control

We are interested in stability for a controller that controls per branch, see figure G.1 and constraints F.3. The constraints are calculated by arrivals to the berenkuil not by trafficflows on the berenkuil (62 to 71).

Constraint	Morning rush-hour	Evening rush-hour
$\rho_2 + \rho_3 + \rho_6 < 1$	0.56	0.25
$\rho_2 + \rho_6 + \rho_{10} < 1$	0.39	0.17
$\rho_3 + \rho_{11} + \rho_{12} < 1$	0.69	0.85
$\rho_3 + \rho_7 + \rho_{11} < 1$	0.55	0.62
$\rho_8 + \rho_9 + \rho_{12} < 1$	0.29	0.63
$\rho_4 + \rho_8 + \rho_{12} < 1$	0.37	0.75
$\rho_5 + \rho_6 + \rho_{12} < 1$	0.73	0.84
$\rho_2 + \rho_5 + \rho_9 < 1$	0.63	0.42
$\rho_2 + \rho_6 < 1$	0.38	0.16
$\rho_3 + \rho_5 + \rho_9 < 1$	0.57	0.46
$\rho_3 + \rho_6 < 1$	0.32	0.2
$\rho_2 + \rho_6 + \rho_{12} < 1$	0.60	0.55
$\rho_3 + \rho_{12} < 1$	0.40	0.49
$\rho_2 + \rho_6 + \rho_{11} < 1$	0.66	0.52
$\rho_3 + \rho_{11} < 1$	0.47	0.45
$\rho_3 + \rho_8 + \rho_{11} < 1$	0.52	0.66
$\rho_8 + \rho_{12} < 1$	0.27	0.60
$\rho_3 + \rho_9 + \rho_{11} < 1$	0.48	0.48
$\rho_9 + \rho_{12} < 1$	0.24	0.42
$\rho_5 + \rho_8 + \rho_{12} < 1$	0.65	**0.94**
$\rho_5 + \rho_9 < 1$	0.39	0.37
$\rho_6 + \rho_8 + \rho_{12} < 1$	0.40	0.71
$\rho_6 + \rho_9 < 1$	0.15	0.14

System control

Similar to the previous section the constraints of the berenkuil are used to calculate stability but this time for a system that only allows one direction to be served at a time, see equation F.4. As there are now separate lanes for turning left (for example lane nr 3) the constraints do change. For traffic flows that enter via the same branch (2 and 3) only the traffic flow with the highest utilization is needed. The time it takes to clear the busiest direction is enough to also clear the less busy direction. Again lane 1 does not have a traffic light.

Constraint	Morning rush-hour	Evening rush-hour
$\rho_{5,6} + \rho_{8,9} < 1$	0.42	0.55
$\rho_{2,3} + \rho_{5,6} + \rho_{10} < 1$	0.63	0.44
$\rho_{2,3} + \rho_7 + \rho_{11,12} < 1$	0.6	0.65
$\rho_4 + \rho_{8,9} + \rho_{11,12} < 1$	0.44	0.75
$\rho_{2,3} + \rho_{5,6} + \rho_{8,9} + \rho_{11,12} < 1$	0.96	**1.04**