MASTER

Improving Smart Traffic

Some insights from queueing theory and an event-driven model for predicting traffic at an intersection

Vromans, Imke

Award date:
2021

Eindhoven University of Technology

Final project (2MMR30)

Improving Smart Traffic: Some insights from queueing theory and an event-driven model for predicting traffic at an intersection

---

Author
Imke Vromans
1024722

TU/e Supervisors
R.W. Timmerman, MSc
dr. ir. M.A.A. Boon

Company
Sweco Nederland

Company supervisor
Sandra Kamphuis

August 19, 2021

Abstract

Smart Traffic is a traffic control software application designed by Sweco. It uses microsimulation model SUMO and data from various sources, among which detector loop data and floating car data, to simulate the actual and forecasted traffic image at an intersection. Based on the forecasted image it calculates the delay and number of stops per vehicle to determine the most efficient duration of green times. The two main challenges of the current implementation of *Smart Traffic* are its use of micro microsimulation model SUMO, which is too computationally expensive to predict traffic at large intersections over a longer period of time, and the lack or inconsistency of data. In the first part of this thesis we replaced the SUMO model used in the forecasting module of *Smart Traffic* by a discrete event model, which models traffic at a signalised intersection as a series of discrete events. We show that this discrete event model is able to accurately predict the delay of the vehicles and is more scalable and faster than the current SUMO model. In the second part of the thesis we study the inconsistency of the data by modelling traffic at a signalised intersection as a polling model with switching customers. In the polling model customers may leave a queue to join another queue, which mimics the situation in *Smart Traffic*, where vehicle may switch signal groups due to updated data. We describe the polling model and analyse its stability condition and joint queue length distributions. From an example of the polling model with switching customers we conclude that the lack or inconsistency of data may lead to *Smart Traffic* scheduling non-optimal green times.

Contents

1 Introduction

This thesis was conducted at the Mobility Solutions division at Sweco Nederland. Sweco, originally Swedish Consultants, is the largest engineering and consultancy in Europe with over 17.000 employees in 13 countries [1]. In 2015 the Dutch company Grontmij was acquired by Sweco and renamed Sweco Nederland [2]. Sweco advises in efficient infrastructure, environmental technology and sustainable buildings. Sweco designs and develops cities and societies of the future.

One of the challenges for cities of the future is the ever increasing amount of road traffic, which leads to traffic congestion. Traffic congestion is not only time consuming, but also leads to unsafe and unclear traffic situations. Futhermore, it increases air pollutant emissions due to an excess of idling vehicles [3]. The development of intelligently controlled traffic lights is thus essential to keep cities habitable by reducing traffic congestion [4].

The Mobility Solutions department, part of the Transport and Mobility division of Sweco, focuses on the development of smart solutions for mobility. One of the products in development at the department is *Smart Traffic*. *Smart Traffic* is a software application intelligently controlling traffic lights, which makes use of a predictive real-time traffic model. The model combines information from multiple data sources, ranging from detection loop data to floating car data broadcasted by connected vehicles. *Smart Traffic* optimises traffic flow by predicting the individual waiting time of vehicles. Based on this information the traffic light controller has complete freedom to determine the most efficient and fair duration of red en green times. *Smart Traffic* currently operates only on isolated junctions, however the objective is to extend the use of *Smart Traffic* to a network of junctions. A detailed explanation of *Smart Traffic* is given in Section 2.1.4. *Smart Traffic* has already proven its success in the city of Helmond, where its implementation reduced CO_2 emission by 20% and vehicle waiting times by 22% [5].

The current implementation of *Smart Traffic* gives rise to two main challenges. The first is the use of microsimulation model SUMO (Simulation of Urban MObility). *Smart Traffic* makes use of SUMO to monitor and predict the traffic image. The traffic image details the overall traffic situation and position of the vehicles. SUMO predicts the new position of vehicles for each time step, making use of detailed information such as vehicle-to-vehicle effects. Though this level of detailing makes the model rather accurate it is not suitable for the objective of Sweco: to predict traffic over a longer period of time for a network of junctions, as the detailing makes the model too computationally expensive. Thus the next step in improving *Smart Traffic* is replacing the current microsimulation model by a discrete event model, which is more scalable and less complex.

Unlike microsimulation models, discrete event models do not update the entire system every time step. Discrete event models model the system as a sequence of discrete events. Every event has a predicted time at which it occurs. The event most recent in time is executed and the relevant entities are updated. The simulation then jumps to the next event. Between events it is assumed that the system does not change or changes according to some deterministic pattern. A more detailed explanation on discrete event models is given in Section 2.2. In a previous internship at Sweco a basic discrete event model was created to replace SUMO in the *Smart Traffic* application of Sweco. In the first part of the thesis we improve upon that model.

The second challenge is the lack or inconsistency of information. The main data sources used by *Smart Traffic*, floating car data and detection loop data, are often incomplete. That is, not all vehicles broadcast floating car data and not all roads contain detection loops. Furthermore, the received data may not be accurate. Detection loops may be broken or overly sensitive and indicate the presence of a vehicle when there are none. Or a vehicle may be expected at a certain place due to the floating car data, which may be contradicted by the detection loop data which indicates that no vehicles are present. To overcome this *Smart Traffic* makes use of a match sensor which attempts to match the real-life vehicle with the simulated vehicle. When the match sensor receives conflicting or updated information simulated vehicles may be added, removed or switch position. In the second part of the thesis we focus on the effect of simulated vehicles switching position, due to a correction made by the match sensor, by modelling traffic at a signalised intersection as a polling model with switching customers.

The structure of the thesis is as follows. In Section 2 we review the literature on current traffic light control methods and explain Smart Traffic in more detail. We also describe discrete event models in general and compare this type of model to the microsimulation model SUMO. In Section 3 we implement a discrete event model to predict traffic at an intersection. We give the main assumptions and a detailed model description. We also describe the main model parameters and how they may be determined. The results of the discrete event model are given in Section 4, where we compare the delay calculated by our discrete event model with the delay given by microsimulation model Vissim, which will function as a simulated version of reality. We then focus on the polling model. In Section 5 we first review the current literature on polling models and then model a signalised intersection as a polling model with switching customers. For this model we determine its stability condition and joint queue length distributions. In Section 6 we discuss the results of the polling model with switching customers by considering two examples. In the final section of this thesis we give the conclusion and discuss the results.

2 Literature review

In this section we give an overview of the current methods used in traffic light control and explain Smart Traffic in more detail. We also explain how a discrete event model works and the main differences between microsimulation models and discrete event models.

2.1 Traffic signal control

Traffic signal control aims to enable the safe and efficient movement of vehicles at an intersection. The efficiency of an intersection may be measured in the total vehicle delay, the number of vehicles in the queue, the number of stops per vehicle or the throughput of the intersection. Before we describe the most common strategies for traffic light control, we first introduce some basic notions and terminology.

A group of traffic lights which give the same signal (e.g. green) simultaneously is called a *signal group*. A possible configuration of (non-conflicting) signal groups is called a *stage*. A series of repeating stages is called a *cycle*. The duration of one full cycle is called the *cycle time*. Between two consecutive stages a brief period in which all traffic signals are red is necessary to ensure safe crossings. The time in which all traffic signals are either red or amber is called the *intergreen time*. An example of a cycle, signal groups and stages can be seen in Figure 1.

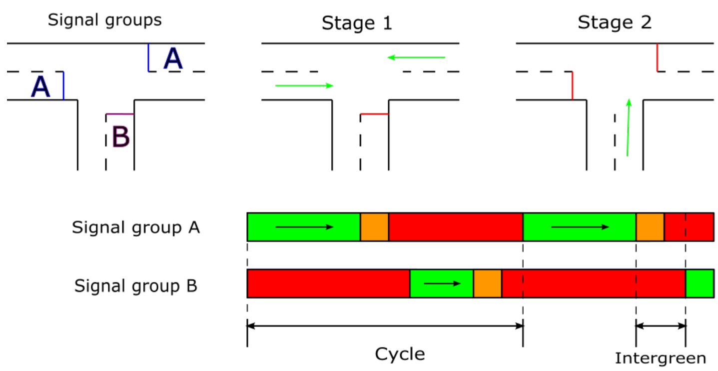


Figure 1: Example of a cycle, signal groups and stages.

The traffic light control methods can be separated into two categories: isolated traffic signal control, which concerns the optimisation of a single junction, and coordinated traffic signal control, which concerns the optimisation of a network of junctions [6].

Papageorigiou et al. [7] describe the following four possibilities for influencing traffic conditions at isolated and coordinated intersections through traffic light control

- *Stage specification*: An intersection generally has a larger number of possible stages. The choice of how many, which and in what order stages are in a cycle may greatly influence the performance of the intersection.

- *Split*: The relative duration of the green time of a stage as a fraction of the cycle time is called the split. Certain directions or signal groups may see a larger number of vehicles than others. It may thus be more efficient, in terms of e.g. total vehicle delay, to grant a certain stage a longer green time duration than other stages.
- *Cycle time*: A longer cycle time reduces the fraction of intergreen time and thus increases the capacity of the junction. However, longer cycle time may also lead to longer waiting times as the stages switch after a longer period of time.
- *Offset*: In a network of junctions, green times of successive junctions may coordinated and optimised to allow for a "green wave". The offset is defined as the time relationship between two successive junctions in seconds or in percentage of the cycle length. Offset optimisation only concerns coordinated intersections.

In the following subsections we will describe three methods to control traffic lights: fixed time signal control, vehicle actuated control and self-optimised real-time control. In Subsection 2.1.4 we describe *Smart Traffic* in detail.

2.1.1 Fixed cycle control

In fixed cycle control, the green times and cycle times are fixed. The signal timing plan, which indicates the starting time of signals for every signal group, is predetermined offline and does not adapt to the current traffic situation. Multiple time plans may be developed to account for varying circumstances, e.g. the time plan at night may be different from the time plan at rush hour.

To calculate the delay in a fixed cycle various methods were developed. Clayton considered a model where arrivals and departures happen at strictly regular intervals [8]. Webster obtained a formula for calculating the overall delay by considering Poisson arrivals [8].

An often used and well known software tool aiding the optimisation of fixed cycle time plans is TRANSYT (Traffic network study tool) [9]. TRANSYT consists of a macroscopic traffic model and a signal optimiser. Using manually entered traffic flows and a platoon dispersion model TRANSYT calculates the Perfomance Index, an economic cost based on the total delay and number of stops. TRANSYT then optimises by running the traffic model multiple times for adjusted time plans and adopts the time plan which reduces the Perfomance Index the most [7]. TRANSYT may be used for a network of junctions, where it is able to create green waves [9].

As fixed cycle control does not adapt to the current traffic situation, it does not lead to the optimal time plan. Particularly in light unpredictable traffic, fixed cycle control performs much worse than other traffic control methods. Furthermore, the fixed signal time plans need to be regularly updated, as studies have shown that fixed time plans degrade over time as traffic demands change [10].

2.1.2 Vehicle actuated control

Vehicle actuated control makes use of detection loops. Detection loops are induction loops embedded in the road. Short detection loops are usually present at a few meters and a few hundred meters from the stop line. These loops register the presence of vehicles and are able to count the number of vehicles which pass by. Between two short loops a long loop may be present which is only able to detect the presence of traffic.

Vehicle actuated control works by extending the amount of green time for a given stage based on the detection of vehicles. Every stage has a minimal green time and maximum green assigned to it. The traffic signal controller will remain in the current stage for at least the minimal green time. When a vehicle is detected during the minimal green time an extension interval is created. As long as the time between two arriving vehicles is shorter than this extension interval the traffic controller will remain in the current stage, until the maximum green time is reached [6].

Basic vehicle actuated control does not take traffic in front of the red light into consideration. Thus vehicle actuated control does not lead to the optimal time plan.

2.1.3 Self-optimised real-time control

More advanced methods were developed which use real-time input from detectors and do take all signal groups into consideration. We name the following

- *MOVA* (microprocessor optimised vehicle actuation) is developed by the British Transport Research Laboratory (TRL) for isolated intersections. MOVA uses Miller's algorithm, which decides to extend green times by weighing the gain to (extra) vehicles passing the junction with the loss to other vehicles waiting in the queue. MOVA uses a microscopic traffic model to predict the position of each vehicle and calculates the total delay every half-second [6].

- *SCOOT* (Split Cycle Offset Optimisation Technique) is often considered the on-line version of TRANSYT. Similar to TRANSYT, SCOOT uses a dispersion model to estimate the arrival patterns. It optimises the split, cycle and offset times, each with independent optimisation procedures. Every procedure only considers a small change in signal settings to estimate the delay and number of stops [11].

- *SCATS* (Sydney Coordinated Adaptive Traffic) was developed in Australia. It does not optimise the traffic signal real-time, but uses historic data and detector information from the previous cycle to determine the optimal cycle. Since it does not use data real-time, SCATS does not have a traffic model to estimate the number of vehicles based on data from upstream detectors, but estimates the number of vehicles directly from information from stop line detectors. Similar to SCOOT, SCATS optimises the split, cycle and offset times in independent procedures [11].

- *UTOPIA* (Urban Traffic Optimisation by Integrated Automation) has a hierarchical structure of three levels, the local level, the area level and the town supervisor level. The local level uses detectors at the start of each link and a microscopic model to estimate the state of the junction and the delays. The area level estimates the state of the entire network using a less detailed traffic model and validates the local level results using a historic traffic database [11]. The town supervisor level uses a macroscopic model and integrates UTOPIA with data from SPOT (System for Priority and Optimisation of Traffic). UTOPIA was specifically designed to give priority to public transport which SPOT provides [9].

2.1.4 Smart Traffic

Smart Traffic is the software package created by *Sweco* to intelligently control traffic lights at an intersection. As the more advanced methods discussed previously, *Smart Traffic* makes use of real-time information from detection loops. In addition to this *Smart Traffic* combines and makes use of several other data sources of which floating car data (FCD) is the main data source. FCD is broadcasted by individual vehicles and contains information on the position and speed of the vehicle [12]. This real-time data, from detection loops and FCD, is used to capture the current traffic image. Based on this image and the stage *Smart Traffic* predicts the future positions of vehicles using microsimulation model SUMO and calculates the delay and number of stops per vehicle.

Due to the large number of stages at a given junction and the time-consuming computational complexity of SUMO, *Smart Traffic* currently simulates only four stages, where every signal group gives a green signal in exactly one stage and red in all others. We will refer to these simulated stages as schedules. The schedules contain information on which traffic lights are scheduled to turn green at which specific times.

We will now describe the way *Smart Traffic* operates. *Smart Traffic* consists of four modules: *Monitoring*, *Forecasting*, *Control* and *Communications*. The *Control* module is the main loop which communicates with all other modules. The loop starts with *Control* asking a snapshot from the *Monitoring* module. *Monitoring* is responsible for capturing the current traffic image. It does this based on historic and actual data, such as FCD and data from detection loops. The snapshot created by *Monitoring* is then sent to *Forecasting* together with a schedule. *Forecasting* predicts the position of the vehicles under the conditions of the schedule and calculates the delay and number of stops for each vehicle. These results get sent back to *Control*. *Control* sends the snapshot created by *Monitoring* and a schedule to *Forecasting* four times each with a different schedule. Finally, based on the results of the four simulations, *Control* calculates the delay and number of stops for all stages and optimises over these results. The optimal stage is then sent to *Communications* which is responsible for controlling the actual traffic lights.

Both the *Monitoring* module and *Forecasting* module make use of microsimulation model SUMO.

2.2 Discrete Event Simulation

Currently, *Smart Traffic* uses the microsimulation model SUMO to predict the delay of vehicle at an intersection. Due the detailing in the microsimulation model of SUMO, the model is not suitable for the objective of Sweco, which is to control a network of junctions and predict traffic over a longer period of time. In this thesis we consider a discrete event simulation as a replacement for SUMO. We describe how a general discrete event simulation (DES) works and the main differences between microsimulation models and discrete event models.

A discrete event simulation models the system as a sequence of discrete events in time. Every event has a specific (predicted) time at which it takes place and a type. The type of event determines the changes the system undergoes. Additionally, the event may contain information on the specific entities involved, for instance if the event only concerns a single customer, this customer may be added to the event.

All predicted events are placed in chronologically ordered list called the Future Event Set (FES). During each step of the simulation the first event, i.e. the event most recent in time, is removed from the FES. The time of the simulation is updated to the time of this event and the event is executed. Based on the type of event the state of the system changes for the relevant entities involved. During the event, other events may be scheduled. These new events will also be placed in the FES. After executing the event, the simulation will jump to the next event. This is again the event most recent in time. The simulation continues until a certain preset time horizon is reached.

One of the main differences between DES and SUMO is the time advancement method. SUMO uses synchronous time advancement. This means that the system gets updated at regular time points. Events which take place between these two time points will be executed at either of the two time points. SUMO updates the entire system every second. This is computationally expensive, since for every second the entire state of the system has to be calculated. DES, on the other hand, uses asynchronous time advancement. Rather than updating the entire system at regular time points, parts of the system get updated at a time point at which an event takes place. This is an efficient way of modelling discrete event systems, as we only need to calculate the state of the system, when a significant chance takes place.

Another difference between DES and SUMO is that in DES we only take into consideration the relevant properties. The extensive detailing used by SUMO and other microsimulation models, e.g. the vehicle to vehicle interactions, are often unnecessary. For instance, if we wish to determine the length of a queue, in terms of the number of vehicles, we do not need to know how far the vehicles are from the queue, but only when and if they are in the queue. By simulating two events for each vehicle, one indicating an arrival in the queue and one indicating a departure from the queue, we are thus able to obtain all information needed, without simulating the trajectory of each vehicle second by second. This makes the simulation much less computationally expensive.

A more detailed description on DES can be found in *Introduction to Discrete Event Systems* by Cassandras and Lafortune [13]. Discrete event simulations are used for a wide variety of applications, from health care services to manufacturing [14][15]. In this thesis we will consider the arrival and departure of vehicles at a signalised intersection as a discrete event system.

3 Implementation discrete event simulation

In this section we implement a discrete event simulation as a replacement of the *Forecasting* module of *Smart Traffic*. The *Forecasting* module predicts the delay and number of stops per vehicle based on a snapshot created by the *Monitoring* module and the schedule created by *Control*. Both the *Monitoring* module and the *Forecasting* module make use of SUMO. We do not consider the *Monitoring* module, as it runs in real-time and thus is less suitable to be replaced by a discrete event model.

To validate the model we require access to real-life data to compare the calculated delay and number of stops to the actual delay and number of stops. As we do not have access to this data we will make use of Vissim. PTV Vissim, is a microsimulation traffic flow model designed by PTV Planung Transport Verkehr AG. Similarly to SUMO, Vissim models each entity individually and simulates complex vehicle-to-vehicle interactions [16]. We will validate and calibrate our discrete event model using Vissim, which will function as a simulated version of reality. The decision to use Vissim for this propose was based on its realism and ease of use. Furthermore, Sweco currently already makes use of Vissim to calibrate their SUMO model.

3.1 Assumptions

Before we describe the model we first give the main model assumptions:

- *Vehicles are either passenger cars or heavy goods vehicles (HGV).* We do not consider busses, trams, cyclists or pedestrians.
- *All road users obey traffic rules,* e.g. road users will not run a red light.
- *All signal groups are non-conflicting.* In real-life conflicting signal groups may be given green simultaneously, e.g. vehicles turning right may be given green at the same time as cyclists going straight ahead. The vehicles must then yield to the cyclists. We assume that no such situation occurs in our model and that the departure of a vehicle is not hindered by the presence of vehicles at another signal group.
- *Road users may only be assigned to a single signal group and may not change between signal groups.*
- *No additional road users enter the system during simulation.* We only consider the delay and number of stops of vehicles present in the snapshot created by *Monitoring*.
- *All input information is correct.* We assume all information received by *Forecasting* from *Control* is correct.

In the subsequent we use the terms vehicle and road user interchangeably.

3.2 Model description

We model a single signalised intersection as a discrete event system. For a detailed explanation of a discrete event simulation we refer back to Section 2.2. A discrete event system is completely determined by its events. We define the following type of events:

- *arrival_at_queue:* This event signals an arrival of a vehicle at the queue or the stop line.
- *departure_from_queue:* This event signals that a vehicle wishes to depart from the queue. If the traffic light is green or amber, the vehicle succeeds and departs the queue. If the traffic light is red, then the vehicle remains in the queue.
- *trafficlight_to_red:* This event changes the traffic light colour to red.
- *trafficlight_to_amber:* This event changes the traffic light colour to amber.
- *trafficlight_to_green:* This event changes the traffic light colour to green.
- *register_results:* This event calculates the delay and number of stops per road user and writes these results into an output file.

Every event in the simulation has a predicted time at which it takes place, a type and a signal group. The *arrival_at_queue* and *departure_from_queue* events also have a road user associated with the event. Additionally, for our simulation we require a *stop_simulation* event, which prevents the simulation from continuing beyond the desired time horizon and prevents the program from ending due to an empty FES.

The SUMO model currently used by Sweco outputs the the delay and the number of stop per vehicle for all signal groups at timestamps $t = 0, ..., 25$. As Sweco wishes to keep the *Control* module the exact same, the output produced by our model has to be similar to the output of the SUMO model. Thus the *register_result* events are executed at the start time of the simulation plus t, with $t \in \{0, ..., 25\}$.

The simulation starts by initialising the signal groups, the maximum simulation time and the FES. The model receives input from *Control* consisting of the snapshot created by *Monitoring* and the schedule. Based on this input all road users will be imported into the model and all traffic light events will be added to the FES. Additionally, for every road user in the snapshot, we either schedule an *arrival_at_queue* event for the road user or we place the road user in the queue. If the traffic light is green and there is a queue we also schedule a *departure_from_queue* event for the first road user in the queue.

After initialisation the main simulation loop starts. During the simulation loop the first event in time will be removed from the FES. The simulation time will be updated to the time of this event and all relevant entities will be updated according to the type of event. Depending on the type of event new events may be scheduled and added to the FES. After executing the event the simulation jumps in time to the next event until the simulation time has reached the maximum simulation time.

In the following subsections we describe the initialisation and main simulation loop of the model in more detail. In Subsection 3.3 we introduce and determine the following four model parameters: *service time*, *the number in line*, *the speed boundaries* and *gap*.

3.2.1 Initialisation

The simulation starts with the initialisation. First the maximum time of the simulation is initialised and the structure of the junction is read from a file. The file has for every arrival place the signal groups available from this arrival place. Each signal group in the file has an identity number, a length, a maximum speed limit and a number of lanes. The number of lanes determines the number of queues at a signal group. If a signal group has multiple lanes we "split" the signal group into multiple signal subgroups. Each of these subgroups will be a considered as separate signal group within the program for ease of computation. All signal subgroups have a single queue and the identity number of the "original" signal group.

Then a new future event set is created, the road users are imported and the traffic light events are initialised. The *Forecasting* module takes the snapshot from *Monitoring* as input. This snapshot file consists of a list of road users, with the following attributes:

- An identity number
- The time at which the road user arrives in the system. This time may be detected by a detection loop.
- The length of the road user's vehicle.
- The type of road user or vehicle, e.g. passenger car.
- The arrival place. This may be a detection loop id or a lane id.
- The position of the road user in meters from the stop line.
- An array of cumulative probabilities. This array is used to determine to which signal group the road user gets assigned. The length of the array is the amount of signal groups available to this road user.
- The current speed of the vehicle.
- The desired speed of the road user. The desired speed is an attribute in Vissim and is required to make an accurate comparison between the Vissim delay and the delay calculated by our discrete event model.
- A number denoting the position of the vehicle in the queue, i.e. what number in line the vehicle is.

For every road user in the file we determine the signal group to which it belongs. To do so we use the known arrival place of the road user and the array of cumulative probabilities. There are a number of signal groups available to the road user from its arrival place. The array of cumulative probabilities determines how likely it is a road user gets assigned to a certain available signal group. Assigning a road user to a signal group is done by drawing a random number between 0 and 1 and comparing this to the array of probabilities. If the signal group has multiple lanes we assign the road user to the signal subgroup with the least amount of vehicles.

After assigning the road user to a signal (sub)group we check if the signal group has a queue. We assume the signal group has a queue if at least one of the road users assigned to the signal group has a speed less than 10 km/h.

We will briefly explain this assumption. In the simulation we make use of two speed boundaries. A road user approaching the back of the queue will be considered in the queue if they are travelling at a speed of less than 5 km/h and a road user at the front of the queue will be considered leaving the queue if they are travelling at a speed of more than 10 km/h. For an explanation on the values of these speed boundaries we refer to Section 3.3. We will assume that the first road user travelling at a speed less than 10 km/h is in the process of leaving the queue. It is possible that this road user is not in the process of leaving the queue, but rather decelerating to become the first road user in the queue. However, putting this road user in the queue early will influence the simulation less than falsely assuming a road user is not in the process of leaving the queue.

For every signal group sg we thus have one of the two following cases:

- *The signal (sub)group has a queue.* The first road user assigned to the signal (sub)group, ru, gets added to the queue at signal (sub)group sg and a stop is added to the number of stops of road user ru. The position of the front of the queue is updated to the position of ru and the position of the back of the queue is updated to the position of ru plus the length of ru. We also check if ru is the first road user travelling at a speed less than 10 km/h. If the signal group is green then a *departure_from_queue* event is scheduled at the current simulation time t for road user ru.

  For all other road users assigned to sg we check if they are in the queue. The queue is defined by the speed boundaries (as explained in Section 3.3). The first vehicle travelling at a speed less than 10 km/h, if this is not ru, will be added to the queue. All vehicles ahead of this vehicle will also be added to the queue, as these road users were in the queue prior to the snapshot. All vehicles behind the first vehicle travelling at a speed less than 10 km/h will be added to the queue if they are travelling at a speed of less than 5 km/h. Every time a road user is added to the queue we update the position of the back of the queue by adding the length of the vehicle of the road user and the variable *gap*. Similarly, if during simulation a road user is removed from the queue we update the position of the front of the queue by adding the length of the vehicle of the road user and the variable *gap*. The first road user in the signal group, ru, will determine the number in line for all following road users. For all road users in the queue, we update the number in line to correspond with the number in line of ru, i.e. if the road user is one place behind the ru the number is line of this road user is the number in line of ru plus one. We also add a stop to the number of stops for all road users in the queue.

  For all road users not in the queue we schedule an *arrival_at_queue* event. The time of the arrival is calculated as the distance which has to be travelled by the road user divided by the maximum speed. The distance which has to be travelled by the road user is the position of the vehicle minus the length of all vehicles ahead and gaps between the vehicles ahead.

- *The signal (sub)group does not have a queue.* If sg does not have a queue we schedule *arrival_at_queue* events for all road users assigned to sg. The time of the arrival is calculated as the distance which has to be travelled by the road user divided by the maximum speed. The distance which has to be travelled by the road user is the position of the vehicle minus the length of all vehicles ahead and gaps between the vehicles ahead.

After importing the road users, we initialise the module. For all signal (sub)groups the traffic light is set to the initial colour, i.e. the colour at the start time of the simulation, and all traffic light events are scheduled.

After scheduling the *register_results* events and *stop_simulation* event the main simulation loop of the program is started.

3.2.2 Main simulation

In this subsection we describe the main simulation loop. The simulation loop starts by removing the first event in time, e, from the FES. The (current) time of the simulation, t, is updated to the time of event e. The type of event e determines which of the following will take place

- *arrival_at_queue:* The road user, ru, arrives at the queue at signal (sub)group sg.

  If there are no vehicles in the queue at sg and the traffic light is green or amber, the road user can directly drive through. The road user, ru, will leave the system, i.e. pass the stopline, and its departure time, t, is registered.

  If the traffic light is red and there are no vehicles in the queue, arriving road user ru is the first vehicle in the queue. Thus we update the position in meters from the stop line of ru to zero and the position in number of vehicles in line of ru to one. Road user ru gets added to the queue at signal (sub)group sg and a stop is added to the number of stops of road user ru.

  If there is a queue, the road user gets placed in the queue behind the last vehicle, thus we do the following:

  – Update the position of road user ru in meters to the position of the back of the queue plus the variable *gap*, which denotes the distance in meter between two road users waiting in the queue.
  – Get the road user, *last_ru*, which is last in the queue at signal (sub)group sg.
  – Update the number in line of road user ru to the number in line of road user *last_ru* plus one.
  – Add the road user ru to the queue.
  – Add a stop to the number of stops of road user ru.

- *departure_from_queue:* Road user ru attempts to leave the queue at signal (sub)group sg.

  If the traffic light is green or amber, then the road user succeeds and passes the stop line. The road user is removed from the queue, leaves the system and their departure time, t, is registered. Then we check if there are still other road users waiting in the queue. If this is the case we get the first road user in the queue, *next_ru*, and schedule a *departure_from_queue* event for road user *next_ru*. This event will take place at time t plus the service time of road user *next_ru*. The service times are determined in Section 3.3. Else, if the queue is empty, we reset the position of the back and front of the queue in meters from the stop line to zero.

  If the traffic light is red, then road user ru may not leave the system, as the traffic light will have turned red before all road users in the queue could leave the system. We initialise the variable *position* to zero and do the following for i from 1 to the number of road users in the queue at sg:

  - Get i'th road user in the queue: ru.
  - Update the position of road user ru in meters to the variable *position*.
  - Update the position of road user ru in number of vehicles in line to i plus one.
  - Add a stop to the number of stops of road user ru.
  - Update the variable *position* by adding the length of (the vehicle of) road user ru plus the variable *gap* to it.

- *trafficlight_to_red:* Change the traffic light at signal (sub)group sg to red. Reset the position of the front and back of the queue to, respectively, zero and the length of the queue (in meters). The length of the queue is defined as the length of all vehicles in the queue plus the distance between these vehicles.

- *trafficlight_to_amber:* Change the traffic light at signal (sub)group sg to amber.

- *trafficlight_to_green:* Change the traffic light at signal (sub)group sg to green. If there are road users waiting in the queue in front of the traffic light, then we get the first road user in the queue, *next_ru*, and schedule a *departure_from_queue* event for road user *next_ru*. This event will take place at time t plus the service time of road user *next_ru*.

- *register_results:* For every road user we calculate its delay. The delay and the number of stops of the road user is then written into an output file. For a detailed explanation on how this delay is calculated we refer to Section 3.2.3.

After executing event e the next event is removed from the FES and executed. The simulation ends when the simulation time exceeds the maximum simulation time. The structure of the simulation loop can be seen in Algorithm 1.

The output of the simulation is a file containing the delay and number of stops of every vehicle sorted by time stamp (of the *register_result* event) and signal group.

3.2.3 Calculating delay

During the *register_results* event the delay per road user is calculated and written into an output file together with the number of stops per road user. In this section we explain how this delay is calculated.

We define the delay as the difference between the total time the road user has spent in the system minus the minimum time the road user is required to spend in the system in the most optimal situation. The most optimal situation denotes a situation in which there are no other vehicles present, the road user moves at maximum speed and the traffic light is green. The total time the road user has spent in the system is referred to as the sojourn time. We refer to the minimum time the road user is required to spend in the system in the most optimal situation as the minimum travel time. Thus the delay is defined as the

Algorithm 1 Main simulation

```
Create new FES
Initialize traffic light colours and add traffic light events to FES
Import all road users
Initialise time t
Schedule register_results events
Schedule stop_simulation event (at maxTime)
while t < maxTime do
    Get the next event e from FES
    Update the time t
    if type of e = arrival_at_queue then
        Get signal group sg of event e
        Get road user ru of event e
        if the traffic light is green or amber and the queue is empty at sg then
            Register departure time t of road user ru
        else if the traffic light is red and the queue is empty at sg then
            Update position of road user ru to 0
            Update the number in line of ru to 1
            Add road user ru to queue at sg
            Add a stop to road user ru
        else
            Update the position of road user ru (to position of the
                back of the queue plus the variable gap)
            Get last road user in the queue last_ru
            Update number in line of ru to number in line of last_ru +1
            Add road user ru to the queue
            Add a stop to road user ru
        end if
    end if
    if type of e = departure_from_queue then
        Get signal group sg of event e
        if the traffic light is green or amber at sg then
            Get road user ru of event e
            Remove road user ru from the queue at sg
            Register departure time t of road user ru
            if there are still road users in the queue at sg then
                Get next road user in the queue next_ru
                Schedule departure_from_queue for next_ru at t + service time
                    of next_ru
            else
                Reset position of the front and the back of the queue to 0
            end if
        else
            for all road users in the queue at sg do
                Update position and number in line
                Add extra stop
            end for
        end if
    end if
```

```
    if type of e = trafficlight_to_red then
        Get signal group sg of event e
        Change traffic light to red for signal group sg
        Reset the position of the front of the queue to 0 and the position
              of the back of the queue to the current length of the queue (in
              meters)
    end if
    if type of e = trafficlight_to_amber then
        Get signal group sg of event e
        Change traffic light to amber for signal group sg
    end if
    if type of e = trafficlight_to_green then
        Get signal group sg of event e
        Change traffic light to green for signal group sg
        if there are road users waiting in the queue at sg then
            Get the first road user in the queue next_ru
            Schedule a departure_from_queue event for road user next_ru
                  at time t+ service time of road user next_ru
        end if
    end if
    if type of e = register_results then
        for every signal group do
            Get the signal group sg
            for every road user associated with signal group sg do
                Get the road user ru
                if road user ru has left the system then
                    Calculate the delay of road user ru using the departure time
                else
                    if road user ru is in the queue then
                        Calculate the delay of road user ru using current time t
                              and position of road user ru
                    else
                        Calculate the delay by separating the delay into two
                             sections from the arrival time to the start time and
                             from the start time to the current time t and
                             estimating the distance road user ru has travelled
                    end if
                end if
                Store delay and number of stops of ru in output file
            end for
        end for
    end if
end while
return results
```

sojourn time minus the minimum travel time. As the delay is always greater than or equal to zero we take the maximum of 0 and the calculated delay. We consider three different situations:

- *The road user has left the system.* When a road user leaves the system, their departure time is registered. The sojourn time of this road user is thus their departure time minus the arrival time. The minimum travel time of the road user is calculated by dividing the length of the lane, which is the distance the road user has travelled, by the maximum speed allowed on the lane.

- *The road user is in the queue.* If the road user is still in a queue, the sojourn time is defined as the current simulation time minus the arrival time. The minimum travel time is calculated by first determining the distance the road user has travelled up until then, which is the length of the lane minus the position of the road user, and then dividing this distance by the maximum speed allowed on the lane. The position of the road user is the last known position in meters from the stop line, thus, in case the road user has travelled since the last time the position was updated, the delay of the road user may be overestimated.

- *The road user is in the system, but not in the queue.* If the road user is not in the queue we separate the calculation of the delay into two parts. The first part is the delay from the arrival time to the start time of the simulation. The second part is the delay from the start time of the simulation to the current time of the simulation. Adding the two parts of the delay then gives the total delay for this road user. Taking the maximum over 0 and the calculated delay happens only after the two parts of the delay are summed.

  The first part of the delay we can calculate exactly. The sojourn time is defined as the start time minus the arrival time and the minimum travel time is defined as the distance travelled by the road user divided by the maximum speed allowed on the lane. The distance travelled by the road user is known exactly and is the length of the lane minus the position of the road user.

  For the second part of the delay we define the sojourn time as the current simulation time minus the start time. The distance travelled by the road user is then estimated as the sojourn time times the speed of the road user at the start of the simulation. The minimum travel time is then again the distance travelled by the road user divided by the maximum speed allowed on the lane.

In our simulation, when calculating the delay, we replace all instances of the maximum speed allowed on the lane by the desired speed of the road user. The desired speed is an attribute given to the road user in Vissim. Vissim uses this attribute rather than the maximum speed to calculate the delay. As we wish to compare the delay calculated by our program with the delay calculated in Vissim, the manner in which the delay is calculated has to be the exact same.

3.3 Parameters

In this subsection we describe the model parameters. For each of the parameters we then determine its value. In the model described above we have the following parameters which need to be determined:

- *Service time:* The position of the vehicles in the queue is not static. When the vehicle in front of the queue starts their departure, vehicles further back may start accelerating and thus change position. We therefore define the service time of a vehicle as the required travel time from becoming the first in the queue to passing the stop line. As vehicles at the back of the queue will have had a longer time to accelerate, the service time is dependent on the original number in line of the vehicle. A vehicle approaching a green light without the presence of a queue will have a service time of zero.

- *The number in line:* As the service time is dependent on the original position of the vehicle in the queue, in terms of the number of vehicles in the queue, we require an additional variable indicating what number in line the vehicle is. During the simulation we are able to keep track of the number in line by simply assigning either a one, if the arriving vehicle is first in the queue, or the number in line of the vehicle ahead plus one, to the vehicle arriving in the queue. When the traffic light is red we reset the number in line for all vehicles still in the queue. During initialisation we do not have access to prior information, thus we need to estimate the original position for all vehicles in the queue.

- *The speed boundaries:* During initialisation, for every road user, it has to be determined whether or not they are in the queue. We use speed boundaries for this, i.e. below a certain speed the road user is assumed to be in the queue. We differentiate two boundaries: the speed with which road user arrive at the queue and the speed with which road users depart from the queue. These boundaries may be the same value. The speed boundaries may be determined by simulation. We observe which boundaries give the best result, i.e. for which boundaries the delay calculated by our model is closest to the delay calculated by Vissim, and choose this boundary. As fitting the speed boundary to the simulation may lead to overfitting we consider the same speed boundaries as in Vissim.

- *Gap:* The variable *gap* is defined as the average distance between two vehicles waiting in the queue. When the road users are positioned in the queue, this variable is necessary to estimate the position of the vehicles in meters from the stop line.

As we do not have access to real life data we use the data from Vissim to determine the parameter values of our discrete event model. We create a basic model in Vissim with a single lane and traffic light. We let the vehicles form a queue in front of the traffic light before turning the traffic light to green and registering the relevant attributes.

	1	2	3	4	5	6	7	8
Car	1.106	2.719	2.106	1.841	1.836	1.726	1.711	1.675
HGV	1.164	3.845	2.840	2.556	2.529	2.377	2.354	2.286

Table 1: The mean service times for passenger cars and heavy goods vehicles based on their position in the queue.

To determine the service times of the road users we register the time at which they pass the stop line. For the first road user in the queue we register the time from the traffic light turning green to the passing of the stop line. For all other vehicles we register the time which passes between two vehicles passing the stop line. We consider two types of vehicles, passenger cars and HGV (heavy goods vehicles). As the service time of the road user is dependent on the type of vehicle, we calculate the service times twice, once using only passenger cars and once using only HGV. We do not consider a mixture of vehicles as the type of the vehicle and its number in line are the main factors determining its service time. Furthermore, if we were to consider (all) vehicles ahead of a road user this would be too computationally expensive, as the number of possible combinations of their types would be too large. In Table 1 we see the mean service times after running the basic simulation in Vissim 100 times for every vehicle type. In Figure 2 and 3 the boxplots of the service times can be seen.

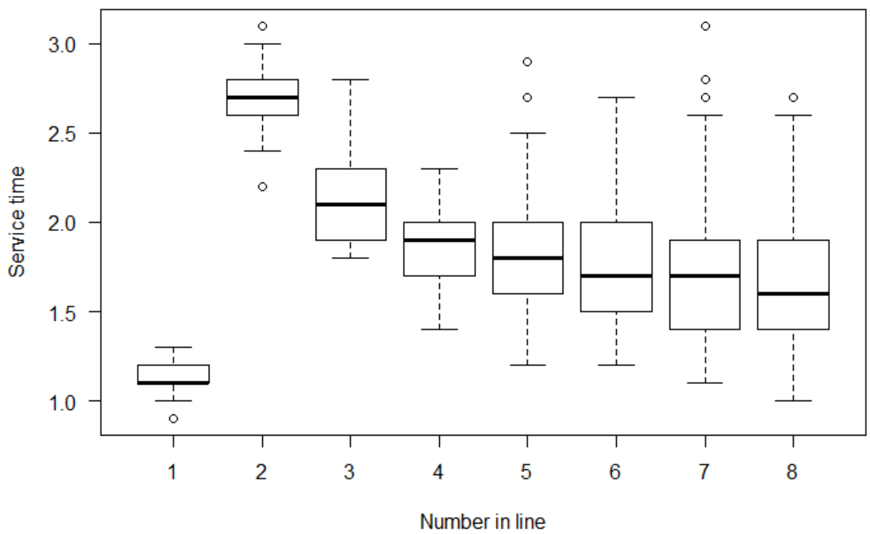


Figure 2: Boxplot of the service times for passenger cars after 100 runs of the basic Vissim model.

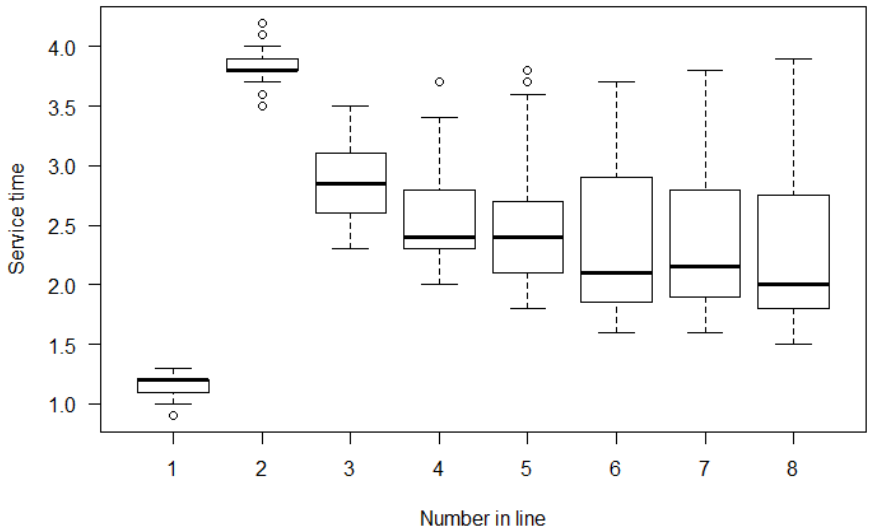


Figure 3: Boxplot of the service times for HGV after 100 runs of the basic Vissim model.

From the results of the service times we can conclude that the position of the road user in the queue, i.e. what number in line the road user is, is a relevant attribute in determining the service time. A snapshot created by *Monitoring* is used during initialisation. At the moment of the snapshot road users in front of the queue may have already passed the stop line, thus the original number in line of the road user has to be estimated. We estimate the original number in line based on the position and speed of the vehicle using a decision tree. We run the basic Vissim simulation 100 times with a mixture of passenger cars and HGV and partition the data into training and testing data. We train the decision tree in R using the speed and position of every vehicle in the queue at every simulation second, considering only training data acquired after the traffic light has turned green. Using the testing data we determine that the decision tree we obtain is 82.3% accurate, meaning that the prediction of the number in line of our decision tree is equal to the acual number in line in 82.3% of the cases. In Table 2 we see the prediction of the number in line of our decision tree compared to the actual number in line.

The third parameter, the speed boundaries, are required to determine whether a road user is in the queue at initialisation. We run the basic Vissim model 100 times with a mixture of passenger cars and HGV. The Vissim model has an attribute *inQueue*, which indicates if a vehicle is in the queue. For every road user, we consider the first instance that a road user is considered in the queue and the last instance that a road user is in the queue and examine the speed at these moments. The average speed at the moment a road user is first considered in the queue by Vissim is 4.71 km/h. The minimum speed is 2.42 km/h and the maximum speed is 5.00 km/h. From this we conclude that a vehicle arriving at

actual / pred.	1	2	3	4	5	6	7	8
1	855	64	1	1	0	0	0	0
2	80	1125	92	7	0	0	0	0
3	8	128	1340	141	20	0	1	0
4	4	6	109	1585	192	18	11	7
5	0	1	22	138	1876	189	50	18
6	0	1	6	22	230	1985	315	72
7	0	2	4	6	68	195	2241	454
8	0	0	0	4	16	48	239	2904

Table 2: Confusion matrix for the prediction of the number in line.

the queue is considered part of the queue when its speed is less than 5 km/h. Similarly, we determine when a vehicle is last considered to be in the queue and find an average of 9.51 km/h and a minimum of 8.91 km/h and a maximum of 10.00 km/h. Thus we conclude that a vehicle has left the queue when its speed is higher than 10.00 km/h.

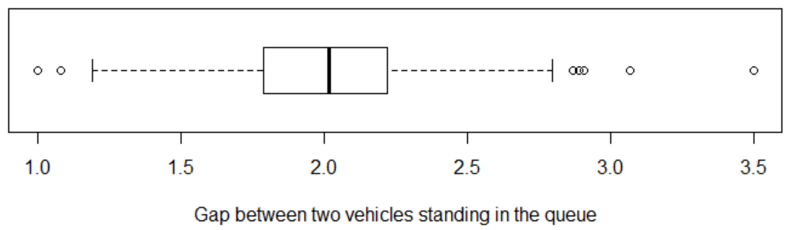


Figure 4: Boxplot of the variable *Gap* after running the basic Vissim model 100 times with a mixture of passenger cars and HGV.

Finally, to determine the parameter *gap*, we again run the basic Vissim simulation 100 times with a mixture of passenger cars and HGV. We consider only vehicles which are standing still and determine the gap between two consecutive vehicles. The resulting boxplot can be seen in Figure 4. Due to the stochastic nature of the Vissim model the gap between two vehicles varies from 1 meter to 3.5 meters. In our model we will implement the average value of the gap, which is 2.010249 meter.

4 Results discrete event simulation

In this section we discuss the results of the discrete event simulation. We will focus on the difference in the calculated delay of our model and the delay in the Vissim model, which functions as a representation of reality. We refer to Appendix A.1.3 for an example output of our model for a single run.

We test our model on a basic junction given to us by Sweco. In Figures 5 and 6 we can see the junction and its signal groups. The junction has four arrival places. Two arrival places at approximately 100 meters and two arrival place at approximately 50 meters. From every arrival place a road user may be assigned to one of three signal groups. As we wish to accurately compare our calculated delay with the delay from Vissim, we assume that the signal group to which a road user get assigned is known. Thus every road user will get assigned to the same signal group in our discrete event simulation as in the Vissim simulation.

We consider a low traffic scenario and a high traffic scenario. In the low traffic scenario road users arrive at every arrival place with a rate of 300 vehicles per hour. Thus the total arrival rate at the junction is 1200 vehicles per hour. In the high traffic scenario the vehicles arrival at a rate of 800 vehicles per hour, thus the total arrival rate at the junction is 3200 vehicles per hour. For both scenarios we consider three signal timing plans: one with a cycle time of 40 seconds, one with a cycle time of 60 seconds and one with a cycle time of 80 seconds. The signal timing plan for a cycle of 60 seconds can be see in Table 3. The signal timing plan for a cycle of 40 or 80 seconds respectively decreases or increases the amount of green time for every signal group by 5 seconds.

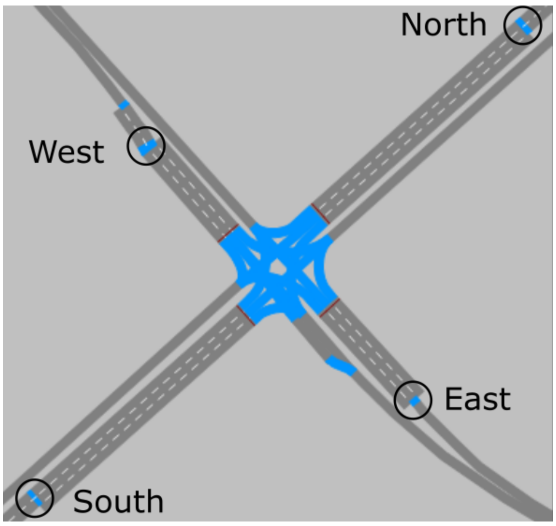


Figure 5: Screenshot of the Vissim model showing the structure of the junction and its arrival places.

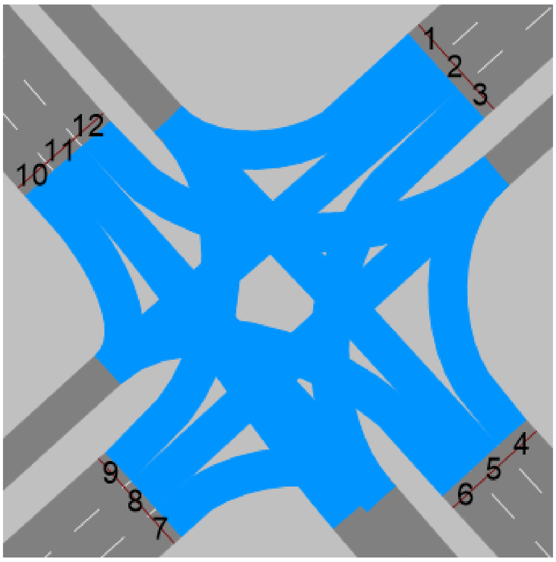
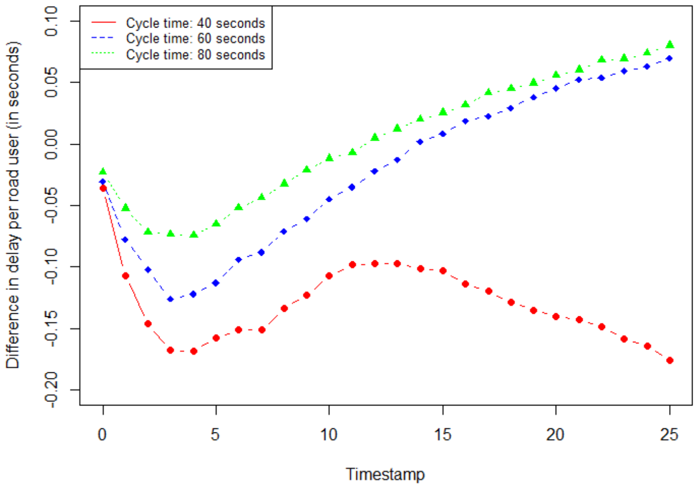


Figure 6: Screenshot of the Vissim model showing a close up of the junction used to test our model, with its signal groups.

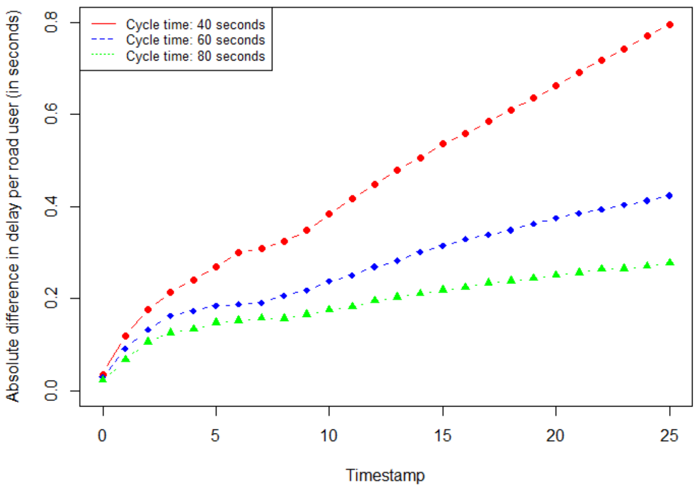
	Start times		
Signal Groups	Green	Amber	Red
1,2 and 3	0	12	15
4,5 and 6	15	27	30
7,8 and 9	30	42	45
10,11 and 12	45	57	60

Table 3: The signal timing plan for a cycle of 60 seconds.

We run the Vissim simulation 100 times for each traffic scenario and cycle time combination. The Vissim simulation lasts 10 minutes and we randomly choose a simulation second from which we obtain the snapshot. We then run our discrete event simulation for this snapshot and obtain the delay for every road user in the snapshot. We compare this delay with the delay in Vissim. A negative difference signifies an underestimation of the actual delay and a positive difference signifies an overestimation of the actual delay. We sum all the differences and average this over the road users to obtain the average difference per road user. We are also interested in the absolute difference in delay, as a (large) negative difference and a (large) positive difference may cancel each other out, which gives the idea that our simulation is accurate, when it is not. The absolute difference is obtained by taking the absolute value of the difference. We sum all the absolute differences and average this over the road users to obtain the average absolute difference per road user.

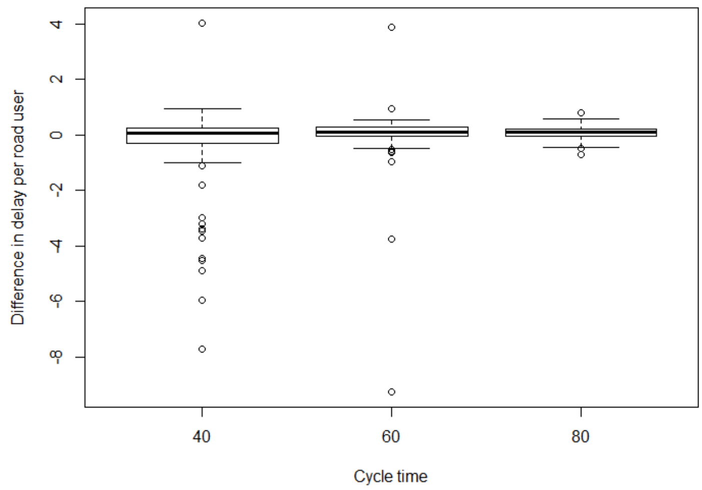


(a) Average difference in delay per road user.

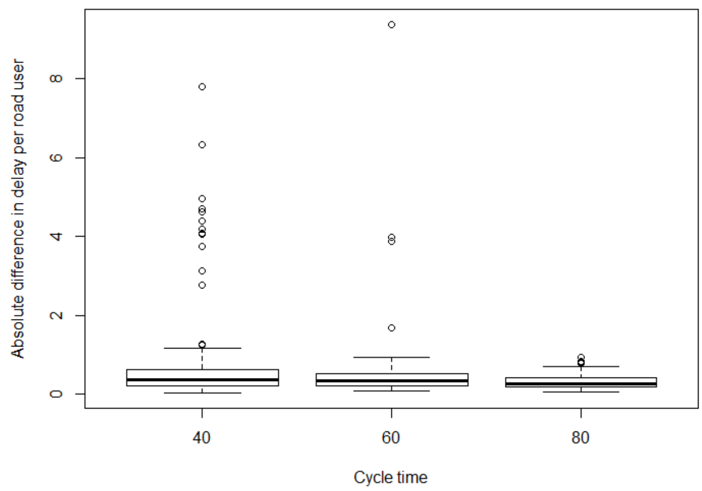


(b) Average absolute difference in delay per road user.

Figure 7: The average difference (7a) and average absolute difference (7b) per road user between the calculated delay of our discrete event model and the delay in Vissim for each timestamp over 100 snapshots with low traffic (300 veh/h per arrival place).



(a) Average difference in delay per road user.



(b) Average absolute difference in delay per road user.

Figure 8: Boxplots of the average difference (8a) and average absolute difference (8b) per road user between the calculated delay of our discrete event model and the delay in Vissim for 100 snapshots with low traffic (300 veh/h per arrival place) after all road users have left the system.

We first look at the results for the low traffic scenario, where road users arrive at every arrival place with a rate of 300 vehicles per hour. In Figure 7 we can see the average difference and average absolute difference per road user for every timestamp averaged over 100 snapshots. As can be seen in Subfigure 7a we underestimate the delay at the beginning of the simulation. After

4 seconds the difference in delay increases again indicating that we start to underestimate the delay less. For a cycle time of 60 and 80 seconds we start to overestimate the delay after approximately 12 seconds. For a cycle time of 40 seconds the difference decreases again after around 12 seconds signifying an increasing underestimation of the delay.

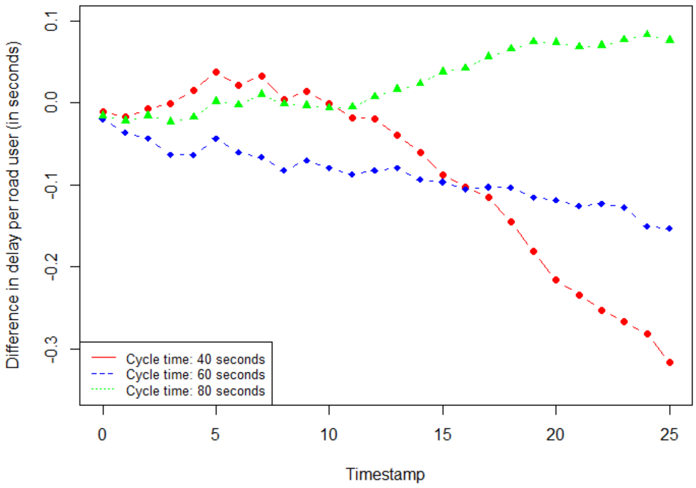
The delay in our model is calculated in three separate ways depending on the situation the road user is in. At the beginning of the simulation a significant part of the road users will be in the system, but not in the queue. As the maximum length of a lane is approximately 100 meters and the maximum speed on the lane is 50 km/h, a road user at the start of the lane will arrive at the queue or stop line in less than 8 seconds. The underestimation at the beginning may thus be caused by the delay calculation for road users not yet in the queue. The underestimation of the delay after 12 seconds, in case the cycle time is 40 seconds, may be due to road users passing the intersection in our model, while the road users in reality do not pass the junction. This happens due to the due to the variability of the service times in Vissim, which can be seen in Figures 2 and 3 in Section 3.3, and the possible faulty estimation of the number in line. We note that Figures 2 and 3 show that most boxplots skew to higher values, indicating that the service times may be much larger than the average. This will lead to road users passing the intersection in our model, due to underestimation of the service time, when in reality they do not pass the intersection.

From Subfigure 7b we conclude that our model is more accurate for higher cycle times. Big differences in delay between the Vissim model and our discrete event model happen when our model predicts a road user passing the intersection, when it should stop in front of a red light, or when our model predicts a road user waiting in front of a red light, when it should pass the intersection. As described before the this happens due to the due to the variability of the service times in Vissim and the possible faulty estimation of the number in line. When the cycle length is long, and thus the green time is long, the variability of the service times has less effect, as all road users will have the chance to pass the intersection. In Figure 8 the boxplots of the average difference and average absolute difference per road user after all road user have left the system is shown. From these boxplots we may conclude that the results show more variability when the cycle time is shorter.

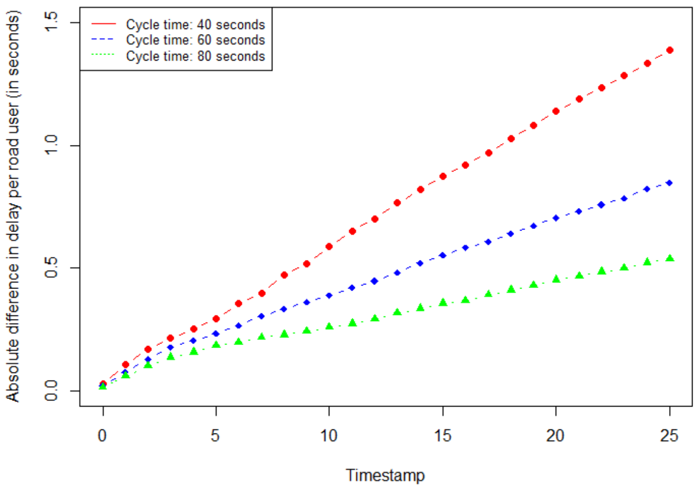
In Table 4 the average difference and average absolute difference per road user over 100 snapshots is shown after all road users have left the system. We note that the average difference is always less than 1 second per road user.

Cycle time	Average difference per road user	Average absolute difference per road user
40	-0.403	0.920
60	-0.0190	0.542
80	0.0795	0.329

Table 4: The average difference and average absolute difference in delay between our discrete event model and Vissim per road user over 100 snapshots with low traffic (300 veh/h per arrival place) after all road users have left the system.



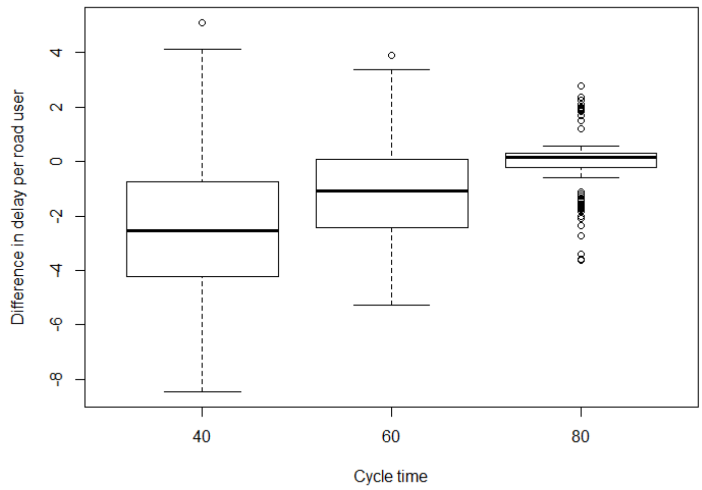
(a) Average difference in delay per road user.



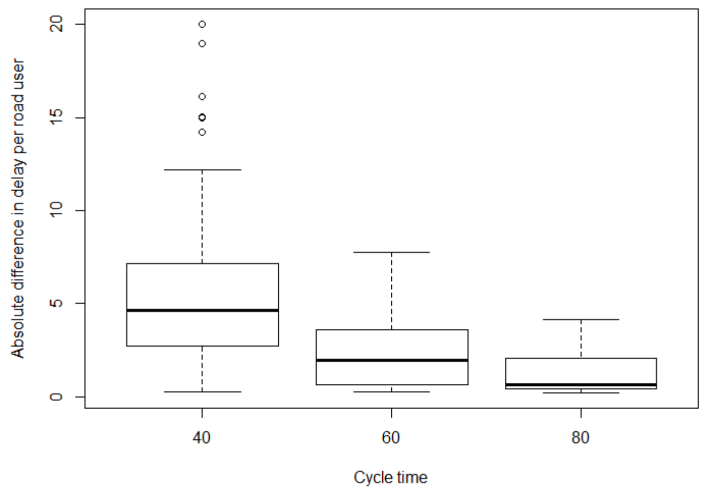
(b) Average absolute difference in delay per road user.

Figure 9: The average difference (9a) and average absolute difference (9b) per road user between the calculated delay of our discrete event model and the delay in Vissim for each timestamp over 100 snapshots with high traffic (800 veh/h per arrival place).

We now look at the high traffic scenario, in which road users arrive at every arrival place with a rate of 800 vehicles per hour, and compare the results with the low traffic scenario. In Figure 9a the average difference per road user for each timestamp is shown. We note that for a cycle time of 40 or 60 seconds we tend to underestimate the delay. For a cycle time of 80 seconds we overestimate the delay. In Figure 9b the average absolute difference per road user for each timestamp is shown and we see that our model, as in the low traffic scenario, is most accurate for a cycle time of 80 seconds.



(a) Average difference in delay per road user.



(b) Average absolute difference in delay per road user.

Figure 10: Boxplots of the average difference (10a) and average absolute difference (10b) per road user between the calculated delay of our discrete event model and the delay in Vissim for 100 snapshots with high traffic (800 veh/h per arrival place) after all road users have left the system.

In Figure 10 we see that the variability of the results for high traffic is, as was the case in low traffic, highest when the cycle time is smaller. As expected the variability is larger under high traffic than low traffic, as the high traffic scenario has more road users.

Cycle time	Average difference per road user	Average absolute difference per road user
40	-2.466	5.571
60	-1.134	2.483
80	-0.0402	1.220

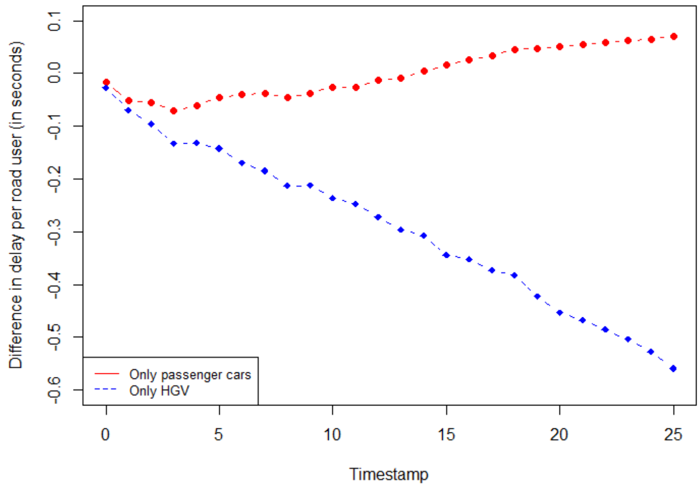
Table 5: The average difference and average absolute difference in delay between our discrete event model and Vissim per road user over 100 snapshots with high traffic (800 veh/h per arrival place) after all road users have left the system.

In Table 5 the average difference and average absolute difference per road user over 100 snapshots is shown after all road users have left the system. The average absolute difference per road user is much larger in the high traffic scenario than in the low traffic scenario. This is particularly visible, when the cycle time is 40 seconds. As described before this is caused by our model simulating vehicles passing the stop line, when in reality the road users either choose to not pass the stop line, when the traffic light is amber, or may not pass the stop line due to road users ahead driving slower than anticipated. As the high traffic scenario has more road users, the number of road users waiting behind other road users in line is higher than in the low traffic scenario. Hence, the effect of the variability of the service times is more pronounced.

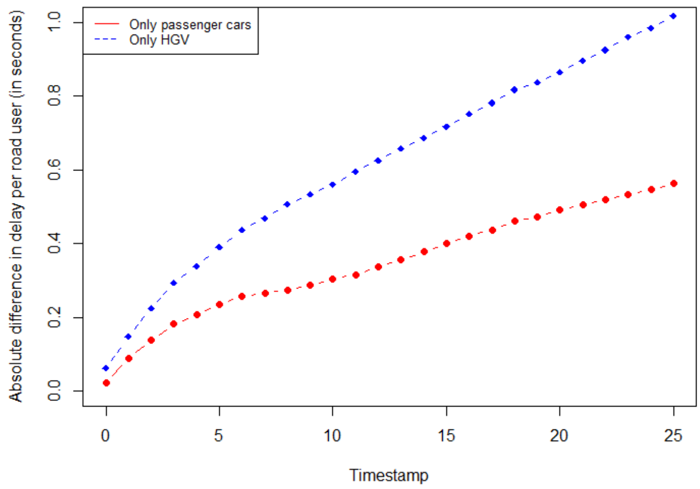
We note that the results in Table 4 and Table 5 do not have much relevance to Sweco, as the delay in *Smart Traffic* is calculated every second for 25 seconds after the start time. The delay after 25 seconds is not taken into consideration. Thus if we focus only on the relevant difference in delay for Sweco, we note that the maximum absolute difference in delay is less than 1.5 seconds per road user for both scenarios.

Another relevant performance measure for Sweco is the running time of the computation, as this is one of the key problems of SUMO. We observe that the average running time of our model for 100 runs is 1.416 seconds under high traffic and 0.875 seconds under low traffic.

In the previous examples we considered a mixture of passenger cars and HGV. We now briefly look at the results in case we only have passenger cars or only HGV. We run the simulation 100 times for only passenger cars and 100 times for only HGV. Each with a cycle time of 60 and an arrival rate of 500 vehicles per hour per arrival place. In Figure 11 the result of the simulations is shown. We note that the simulation is more accurate for only passenger cars. This may be due to the service times of the HGV showing more variance (see Figure 3). This makes it more likely for vehicles to pass the intersection in our model than in Vissim, causing the underestimation of the delay for HGV. We used a similar reasoning to conclude that our model is more accurate for higher cycle times. Another reason for the inaccuracy, in case we only have HGV, may be the calculation of the arrival at queue time. As HGV have a longer breaking distance, HGV will arrive at the queue later than passenger cars and have a larger delay than passenger cars over the same distance.



(a) Average difference in delay per road user.



(b) Average absolute difference in delay per road user.

Figure 11: The average difference (11a) and average absolute difference (11b) per road user between the calculated delay of our discrete event model and the delay in Vissim for each timestamp over 100 snapshots with 500 veh/h per arrival place and cycle time 60.

5 Polling model

In Section 2.1.4 we described the general working of *Smart Traffic*. One of the key challenges concerning *Smart Traffic* is the lack of information on the routing of vehicles. Currently, the routing of the vehicle is guessed based on a probability distribution. This means that an assumption on the routing of the vehicle is made. This assumption is extremely relevant for the performance of *Smart Traffic*, since it will calculate the delay of the vehicle based on the assumed direction of travel. Furthermore, when new information is provided by detection loops, the routing of the vehicle may be updated in accordance with its new position.

In this part of the thesis we will take a closer look at the effect this lack of routing information has on the performance of *Smart Traffic* by modelling a signalised intersection as a polling model. The vehicles or customers, as the actors in a queuing model are usually known, in this polling model may leave a queue to join another queue. This mimics the situation in *Smart Traffic*. When new information is provided on the direction of travel of a vehicle, the vehicle changes position and thus at which signal group or queue it is located.

5.1 Literature

A polling system is a queuing system consisting of a single server attending multiple queues. The server attends a single queue at a time serving the customers in the visited queue. After ending the visit to a queue the server progresses to the next queue. Between two consecutive queue visits the server may incur a switch-over time. A visual representation of a classical polling model can be seen in Figure 12.

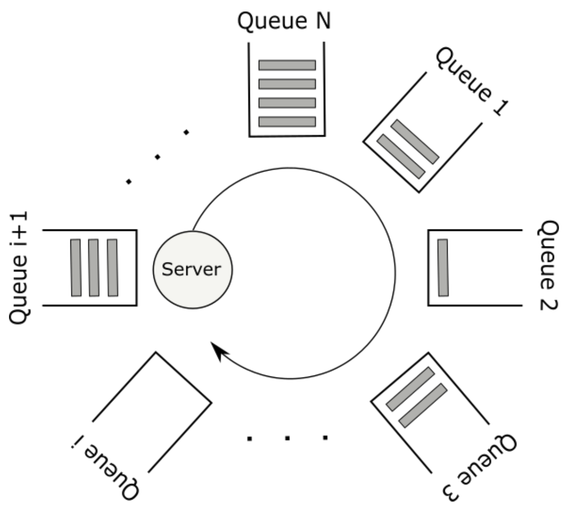


Figure 12: Classical polling model.

Polling models were first studied in 1957, where they were used to describe the problem of a single repairman servicing multiple machines in the cotton industry [17]. Now polling models find their application in many areas, from computer networks to production systems [18] [19]. We refer to [20] and [21] for more extensive overviews on the applicability of polling models.

In general polling models can be used to describe situations in which multiple customer types want to make use of the same common resource which is only available to one type of customer at a time. Traffic at a signalised intersection may thus also be described as a polling model. Vehicles from different directions want to make use of the same part of the road. The traffic light will function as the server, where giving green to a certain direction can be seen as giving service to this direction. Vehicles which arrive at a red traffic light will form a queue. When the traffic light turns green, the vehicles depart from the queue one-by-one. The time it takes for a vehicle to leave the queue may be considered the service time of the vehicle. The clearance time of the intersection or intergreen time can be considered the switch-over time.

Among others, polling models can be characterised by the following aspects:

- *Arrival process*: The arrival process describes the manner in which customers arrive at the queue. The usual assumption is that the interarrival times, i.e. the time between two arriving customers, are independent and identically distributed. A commonly chosen distribution is the exponential distribution, making the arrival process of customers a Poisson process. In literature various other arrival processes have been studied from Lévy processes to general renewal processes [22][23]. Furthermore, customers may arrive one-by-one or in batches [24] [25].

- *The behaviour of customers*: Customers waiting in line may grow impatient and leave the queue. In the literature on queuing systems this is referred to as impatience, abandonment or reneging. Polling models with reneging at polling instants, where customers may only abandon queues at the start of a visit or switch-over period, are studied in [26] and [27]. In the following we focus on a polling model in which customer may decide to change queues at polling instances.

- *Service process*: We usually assume that the service times are i.i.d. distributed and independent from the interarrival and switch-over times. The queuing discipline determines in what order the customers waiting in the queue will be served. Commonly used queuing disciplines are

  - First-Come-First-Serve (FCFS)
  - Random order
  - Last-Come-First-Served (LCFS)
  - Priorities (e.g. shortest processing time first)

  A classical polling model only consists of a single server, however there have been studies on polling models with multiple servers [28]. Vlasiou and Yechiali consider a polling model with an infinite amount of servers [29]. As with arrivals, customers may depart one-by-one or depart in batches [30].

- *The service discipline*: One of the most important properties of a polling model is the service discipline, which determines when a server will switch to serve another queue. The polling model may be analysed in an exact way if the service discipline satisfies the branching property. We describe the branching property in more detail in Section 5.2.1. Common service disciplines are

  - *exhaustive*: The server will leave the current queue if it is empty and it has served all customers.
  - *gated*: The server will only serve the customers which are present at the start of a visit. Customers arriving during the visit will have to wait until the next visit period to be served.
  - *globally gated*: Globally gated is a modification of gated service, introduced in [31]. Under globally gated service the server will only serve customers which are present at the start of a cycle.
  - *k-limited*: The server will leave the current queue if it is empty or if it has served a predefined number of k customers, whichever occurs first.

  Both exhaustive service and gated service satisfy the Branching property. Globally gated does not satisfy the Branching property, but does satisfy a weaker property, see [32], which makes the exact analysis of a polling model with globally gated service possible. In the following we will focus on exhaustive and gated service.

- *Switch-over process*: Between two consecutive queue visits a polling model may incur a switch-over time. A common assumption is that the switch-over times are independent from the current state of the system. In [33] the difference between a classical polling model with and without switch-over times is studied.

- *Server routing*: The server routing describes in which order the server visits the queues. Server routing may be static or dynamic. In a classical polling model we assume a fixed and cyclic visit order. Among others, alternative server routing mechanisms are periodic server routing, described in [34] and Markovian server routing, described in [35].

The focus of the polling model considered in this thesis is on the behaviour of customers. In the subsequent sections we consider a classical polling model with the additional assumption that customers may leave their queue to join another queue.

Before introducing our model in detail, we first describe two other polling models concerning the behaviour of customers, namely the polling model with smart customers as described in [36] and the polling model with reneging as discussed in [26]. The polling model with smart customers studies a polling system in which customers choose to join a queue based on the current position of the server. Then a polling model is obtained with a varying arrival rate in each queue based on the location of the server. A polling model with reneging describes a polling model where customers may choose to abandon the queue at polling instances. In this case, when focusing on the remaining customers only, again a polling system where the arrival rates for each queue depend on the location of the server is obtained. For both systems the cycle time, visit times waiting time distributions and queue length distributions are studied. The analysis of the polling model with smart customers and the polling model with reneging will serve as a template for our own analysis of the polling model with switching customers.

5.2 Analysis of polling model with switching customers

5.2.1 Model description and notation

We consider a polling model consisting of a single server serving N queues denoted by $Q_1, ..., Q_N$. We assume that the server visits the queues in cyclic order, thus after the server has finished serving Q_N it will start serving Q_1 again. The visit time of Q_i, i.e. the time between the visit beginning and visit ending at Q_i, is denoted by V_i. Customers of type i arrive to Q_i according to a Poisson process with parameter λ_i. The customers get served in first-come-first-served (FCFS) order and require a general service time B_i. The Laplace-Stieltjes transform (LST) of the service time is denoted by $\widetilde{B}_i$. After the server ends its visit to Q_i and moves to queue Q_{i+1} the server may incur a switch-over time S_i with LST $\widetilde{S}_i$. A cycle consists of all visit times and switch-over times $V_1, S_1, ..., V_N, S_N$. We assume that all switch-over times, interarrival times and service times are independent of each other.

An important property of a polling model is the service discipline. Every queue has a service discipline, which determines when the server switches position to serve another queue. The tractability of the analysis of a polling model greatly depends on the type of service discipline. Resing [37] shows that if a service discipline satisfies the following property the polling model can be analysed in a rather simple manner:

Property 1 (Branching Property) If the server arrives at Q_i to find k_i customers there, then during the course of the server's visit, each of these k_i customers will effectively be replaced in an i.i.d. manner by a random population having probability generating function (PGF) $h_i(z_1, ..., z_N)$, which can be any N-dimensional PGF.

If every queue in the polling models satisfies this branching property the polling model can be analysed as a Multitype Branching Process (MTBPs) with immigration [37]. Two common service disciplines which satisfy the branching property are gated and exhaustive service. In gated service the server will end its visit to the queue after serving all customers present at the start of the visit period. Customers arriving during the visit period will not be served and will need to wait for the next visit period. In exhaustive service the server will end its visit to the queue when the queue is empty. If the polling model does not satisfy the branching property, it may only be analysed in an analytical manner in a few exceptional cases, e.g. symmetric or two-queue models.

In the following we consider a polling system where customers may change type, i.e. every customer may leave the queue it is currently in to join another queue. We allow customers to only switch queues at the start of a visit or switch-over period, these periods are called polling instances. At the beginning of period $P \in \{V_1, S_1, ..., V_N, S_N\}$ a customer i may leave Q_i to join Q_j with probability p_{ij}. The probability that customer i stays in Q_i is given by $p_{ii} = 1 - \sum_{j=1, j\neq i}^{N} p_{ij}$. These probabilities are independent of the position of the server and the current joint queue length.

To analyse this model we use a technique previously used by Boon [26] and artificially split each period $P \in \{V_1, S_1, ..., V_N, S_N\}$ in a subperiod a and a subperiod b, thus visit time V_i is split into V_{ia} and V_{ib} and switch-over time S_i is split into S_{ia} and S_{ib}. During V_{ia}, right before service starts at Q_i, all customers may leave their queue to immediately join another queue. Thus a customer of type i may leave Q_i and join Q_j with probability p_{ij}, while simultaneously a type j customer at Q_j may join Q_i with probability p_{ji}. The customers changing queues happens instantly and does not require any time, therefore $\mathbb{E}[V_{ia}] = 0$. After the customers have changed queues the actual service will take place at Q_i during period V_{ib}. We thus note that $\mathbb{E}[V_{ib}] = \mathbb{E}[V_i]$. During S_{ia}, right before the beginning of S_i, all customers may again leave their queue to immediately join another queue. The actual switch-over time happens during S_{ib}, hence $\mathbb{E}[S_{ia}] = 0$ and $\mathbb{E}[S_{ib}] = \mathbb{E}[S_i]$. The splitting of the visit times and service times into subperiod is necessary, since the queue length differ between the beginning of subperiod a and the beginning of subperiod b.

5.2.2 Stability condition

The model described in the previous section can be viewed as a polling model with varying arrival rates in each queue, depending on the position of the server. This polling model was first discussed by Boxma et al. [36], who referred to this model as a polling model with smart customers. The necessary and sufficient stability condition described by Boxma et al. is that the Perron-Frobenius eigenvalue of the matrix $R - I_N$ should be less then 0, where R is an N by N matrix with elements $\rho_{ij} := \lambda_i^{(V_j)}\mathbb{E}[B_i]$, where $\lambda_i^{(V_j)}$ is the effective arrival rate at Q_i during V_j, and I_N is the N by N identity matrix.

In order to determine the stability of the polling system we thus have to calculate $\lambda_i^{(V_j)}$. We can define the effective arrival rate at follows

$$\lambda_i^{(P)} = \sum_{k=1}^{N} \lambda_k \cdot q_{Q_k^{(P)} \to Q_i}, \tag{1}$$

where $q_{Q_k^{(P)} \to Q_i}$ is the probability of a customer leaving the system at Q_i, when it arrived at Q_k during a period P, with $P \in \{V_{1b}, S_{1b}, ..., V_{Nb}, S_{Nb}\}$. As $\mathbb{E}[V_{ia}] = \mathbb{E}[S_{ia}] = 0$ there are no arrivals during subperiod a. Hence, we only look at the arrivals during subperiod b and $\lambda_i^{(V_j)} = \lambda_i^{(V_{jb})}$.

To calculate these probabilities we consider a Markov chain with states $Q_j^{(V_{ib})}$ and $Q_j^{(S_{ib})}$, with $i = 1, .., N$ and $j = 1, ..., N$. Let us for the moment only consider a polling system with exhaustive service. Customers which arrive at the queue which is currently being served will also be served. We thus can consider $Q_1^{(V_{1b})}, Q_2^{(V_{2b})}, ..., Q_N^{(V_{Nb})}$ as absorbing states of the Markov chain. In Figure 13 a schematic representation of this Markov chain is given for $N = 2$. Let $q_{Q_j^{(V_{ib})} \to Q_k}$ denote the probability that a customer arriving at Q_j during V_{ib} leaves the system at Q_k and let $q_{Q_j^{(S_{ib})} \to Q_k}$ denote the same for customers arriving during S_{ib}. Then we obtain the following system of equations

$$\begin{aligned}
q_{Q_j^{(V_{1b})}\to Q_k} &= \sum_{n=1}^{N} p_{jn} \cdot q_{Q_n^{(S_{1b})}\to Q_k}, \quad j=2,...,N,\\
q_{Q_j^{(S_{1b})}\to Q_k} &= \sum_{n=1}^{N} p_{jn} \cdot q_{Q_n^{(V_{2b})}\to Q_k}, \quad j=1,...,N,\\
q_{Q_j^{(V_{2b})}\to Q_k} &= \sum_{n=1}^{N} p_{jn} \cdot q_{Q_n^{(S_{2b})}\to Q_k}, \quad j=1,3,...,N,\\
q_{Q_j^{(S_{2b})}\to Q_k} &= \sum_{n=1}^{N} p_{jn} \cdot q_{Q_n^{(V_{3b})}\to Q_k}, \quad j=1,...,N,\\
&\vdots\\
q_{Q_j^{(V_{Nb})}\to Q_k} &= \sum_{n=1}^{N} p_{jn} \cdot q_{Q_n^{(S_{Nb})}\to Q_k}, \quad j=1,...,N-1,\\
q_{Q_j^{(S_{Nb})}\to Q_k} &= \sum_{n=1}^{N} p_{jn} \cdot q_{Q_n^{(V_{1b})}\to Q_k}, \quad j=1,...,N,
\end{aligned}$$

with for the absorbing states

$$q_{Q_j^{(V_{ib})}\to Q_k} = \begin{cases} 1 & \text{if } i=j=k,\\ 0 & \text{if } i=j\neq k. \end{cases}$$

Solving this system of equations then leads to the needed probabilities.

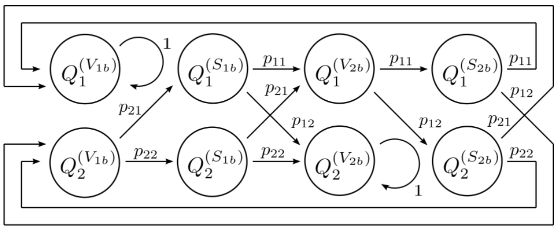


Figure 13: Schematic representation of the Markov chain for exhaustive service for $N=2$.

In case we have gated service the system of equations differs slightly, since customers which arrive at Q_i during the service of Q_i will not get served during this service. Thus $Q_1^{(V_{1b})}, Q_2^{(V_{2b})}, ..., Q_N^{(V_{Nb})}$ are no longer absorbing states. We introduce additional states $Q_1^{(V_{1b}^{'})}, Q_2^{(V_{2b}^{'})}, ..., Q_N^{(V_{Nb}^{'})}$ which will act as absorbing states for customers already in the system. A schematic representation of this Markov chain with $N=2$ is given in Figure 14. Let $q_{Q_j^{(P)}\to Q_k}$ again denote the probability that a customer arriving at Q_j during P leaves the system at Q_k, with $P \in \{V_{1b}, S_{1b}, ..., V_{Nb}, S_{Nb}\}$. Then we obtain the following set of equations

$$q_{Q_j^{(V_{ib})} \to Q_k} = \sum_{n=1}^{N} p_{jn} \cdot q_{Q_n^{(S_{ib})} \to Q_k},$$

$$q_{Q_j^{(S_{ib})} \to Q_k} = p_{j(i+1)} \cdot \mathbf{1}\{i+1=k\} + \sum_{n=1, n \neq (i+1)}^{N} p_{jn} \cdot q_{Q_n^{(V_{(i+1)b})} \to Q_k},$$

for $i = 1, ..., N$ and $j = 1, ..., N$, where we calculate modulo N, thus $N+1 = 1$, and with indicator function

$$\mathbf{1}\{i=k\} = \begin{cases} 1 & \text{if } i = k, \\ 0 & \text{if } i \neq k. \end{cases}$$

We can again solve this system of equations to obtain the relevant probabilities.

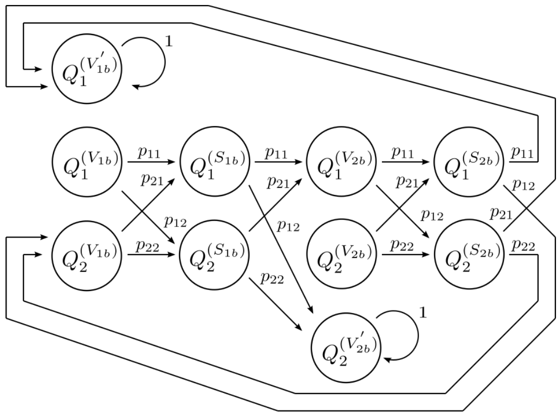


Figure 14: Schematic representation of the Markov chain for gated service for $N = 2$.

From these two examples we can easily see what the Markov chain would like for a polling system, which has a combination of exhaustive and gated service. Filling the relevant probabilities into Equation 1 gives the desired effective arrival rates required to determine the stability condition.

5.2.3 Joint queue length distribution at polling epochs

In this section we derive the joint queue length distribution at polling epochs, i.e. the epochs that the server starts or ends a visit to each queue, for every queue.

Let $\widetilde{LB}^{(P)}(\mathbf{z})$ denote the PGF of the joint queue length at the beginning of each subperiod P, where $P \in \{V_{1a}, V_{1b}, S_{1a}, S_{1b}, ..., V_{Na}, V_{Nb}, S_{Na}, S_{Nb}\}$ and $\mathbf{z}$ denotes the vector $(z_1, ..., z_N)$. We only consider polling systems which satisfy the Branching Property and can subsequently use the Buffer occupancy method to derive an expression for $\widetilde{LB}^{(V_{1a})}(\mathbf{z})$. The Buffer occupancy method relates the joint queue length PGF at the end of a visit to the joint queue length PGF at the beginning of a visit [38]. As we have split every period into subperiods a and b, we relate the PGFs of the joint queue length distributions at the beginning of various subperiods to each other.

We start by relating the PGF of the joint queue length at the beginning of V_{ia} to the PGF of the joint queue length at the beginning of V_{ib}. We have the following definition of the PGF of the joint queue length at the beginning of subperiod V_{ib}

$$\widetilde{LB}^{(V_{ib})}(\mathbf{z}) = \mathbb{E}[z_1^{LB_1^{(V_{ib})}} \cdot ... \cdot z_N^{LB_N^{(V_{ib})}}], \tag{2}$$

where $LB_j^{(V_{ib})}$ denotes the queue length distribution at Q_j during period V_{ib}. We note that the queue length distribution at Q_j during period V_{ib} is dependent on the queue length distribution during period V_{ia} and can be written as follows

$$\begin{aligned} LB_j^{(V_{ib})} &= X_1^{(1,j)} + ... + X_{LB_1^{(V_{ia})}}^{(1,j)} + X_1^{(2,j)} + ... + X_{LB_2^{(V_{ia})}}^{(2,j)} + ... + X_1^{(N,j)} + ... \\ &\quad + X_{LB_N^{(V_{ia})}}^{(N,j)} = \sum_{k=1}^{N} \sum_{l=1}^{LB_k^{(V_{ia})}} X_l^{(k,j)}, \end{aligned} \tag{3}$$

where $X_l^{(k,j)}$ is a Bernoulli random variable representing the l'th customer in Q_k. The l'th customer in Q_k leaves Q_k to join Q_j with probability p_{kj} and does not join Q_j with probability $1 - p_{kj}$. Similarly, the l'th customer in Q_j stays in Q_j with probability p_{jj} and leaves Q_j with probability $1 - p_{jj}$.

We now rewrite (2) using (3) and condition on the length of each queue at the beginning of V_{1a} to obtain

$$\begin{aligned} \mathbb{E}[z_1^{LB_1^{(V_{ib})}} \cdot ... \cdot z_N^{LB_N^{(V_{ib})}}] &= \mathbb{E}[z_1^{\sum_{k=1}^N \sum_{l=1}^{LB_k^{(V_{ia})}} X_l^{(k,1)}} \cdot ... \cdot z_N^{\sum_{k=1}^N \sum_{l=1}^{LB_k^{(V_{ia})}} X_l^{(k,N)}}] \\ &= \sum_{a_1,...,a_N=0}^{\infty} \mathbb{E}[z_1^{\sum_{k=1}^N \sum_{l=1}^{a_1} X_l^{(k,1)}} \cdot ... \cdot z_N^{\sum_{k=1}^N \sum_{l=1}^{a_N} X_l^{(k,N)}}] \\ &\quad \cdot \mathbb{P}(LB_1^{(V_{ia})} = a_1, ..., LB_N^{(V_{ia})} = a_N) \end{aligned} \tag{4}$$

We note that every customer in the queue is independent from other customers in the queue, thus $X_m^{(k,j)} \perp\!\!\!\perp X_n^{(k,j)}$ for all m and n and that queues are independent from each other, thus $X_l^{(y,j)} \perp\!\!\!\perp X_l^{(z,j)}$ for all y and z. There obviously is dependence between the Bernoulli random variables representing the same customer, i.e. $X_l^{(k,i)} \not\!\perp\!\!\!\perp X_l^{(k,j)}$. Using this information we can further rewrite (4) as follows

$$\begin{aligned} &\sum_{a_1,...,a_N=0}^{\infty} (\mathbb{E}[z_1^{X_1^{(1,1)}} \cdot ... \cdot z_N^{X_1^{(1,N)}}])^{a_1} \cdot ... \cdot (\mathbb{E}[z_1^{X_1^{(N,1)}} \cdot ... \cdot z_N^{X_1^{(N,N)}}])^{a_N} \\ &\cdot \mathbb{P}(LB_1^{(V_{ia})} = a_1, ..., LB_N^{(V_{ia})} = a_N) \\ &= \sum_{a_1,...,a_N=0}^{\infty} (\sum_{k=1}^{N} p_{1k} z_k)^{a_1} \cdot ... \cdot (\sum_{k=1}^{N} p_{Nk} z_k)^{a_N} \cdot \mathbb{P}(LB_1^{(V_{ia})} = a_1, ..., LB_N^{(V_{ia})} = a_N), \end{aligned} \tag{5}$$

where we used

$$\mathbb{E}[z_1^{X_1^{(j,1)}} \cdot ... \cdot z_N^{X_1^{(j,N)}}] = \sum_{x_1,...,x_N=0}^{\infty} \mathbb{P}(X_1^{(j,1)} = x_1, ..., X_1^{(j,N)} = x_N) \cdot z_1^{x_1} \cdot ... \cdot z_N^{x_N}$$
$$= p_{j1}z_1 + ... + p_{jN}z_N = \sum_{k=1}^{N} p_{jk}z_k.$$

Equation (5) can be recognised as the PGF of the joint queue length at the beginning of subperiod V_{ia}. Thus we obtain

$$\widetilde{LB}^{(V_{ib})}(\mathbf{z}) = \widetilde{LB}^{(V_{ia})}(\sum_{k=1}^{N} p_{1k}z_k, ..., \sum_{k=1}^{N} p_{Nk}z_k).$$

As the polling system satisfies the Branching Property we can use from its definition that each customer in Q_i at the start of visit period V_{ib} will effectively be replace in an i.i.d. manner by random population having PGF $h_i(z_1, ..., z_N)$. Hence, we find the following relation between the PGF of the joint queue length at the beginning of S_{ia} and the PGF of the joint queue length at the beginning of V_{ib}

$$\widetilde{LB}^{(S_{ia})}(\mathbf{z}) = \widetilde{LB}^{(V_{ib})}\big(z_1, ..., z_{i-1}, h_i(\mathbf{z}), z_{i+1}, .., z_N\big).$$

The PGF $h_i(\mathbf{z})$ depends on the type of service discipline. For gated service we have $h_i(\mathbf{z}) = \widetilde{B}_i\big(\sum_{j=1}^{N} \lambda_j(1 - z_j)\big)$ and for exhaustive service we have $h_i(\mathbf{z}) = \widetilde{BP}_i\big(\sum_{j \neq i} \lambda_j(1 - z_j)\big)$, where $\widetilde{BP}_i$ is the LST of the duration of a busy period at Q_i.

The relation between $\widetilde{LB}^{(S_{ib})}(\mathbf{z})$ and $\widetilde{LB}^{(S_{ia})}(\mathbf{z})$ is similar to the relation between $\widetilde{LB}^{(V_{ib})}(\mathbf{z})$ and $\widetilde{LB}^{(V_{ia})}(\mathbf{z})$, as, similar to V_{ia}, customers may switch queues during S_{ia}.

The queue length distribution at beginning $V_{(i+1)a}$ is equal to the queue length distribution at the beginning of S_{ib} plus all customers which arrived during the switch-over time S_i, thus we obtain the following relation

$$\widetilde{LB}^{(V_{(i+1)a})}(\mathbf{z}) = \widetilde{LB}^{(S_{ib})}(\mathbf{z})\widetilde{S}_i(\sum_{j=1}^{N} \lambda_j(1 - z_j)).$$

Relating the PGFs of the joint queue lengths distributions at the beginning of various subperiods to each other leads to the following set of equations, also referred to as the laws of motion, which can be solved recursively to obtain an expression for $\widetilde{LB}^{(V_{1a})}(\mathbf{z})$

$$
\begin{aligned}
\widetilde{LB}^{(V_{1b})}(\mathbf{z}) &= \widetilde{LB}^{(V_{1a})}(\sum_{k=1}^{N} p_{1k}z_k, ..., \sum_{k=1}^{N} p_{Nk}z_k),\\
\widetilde{LB}^{(S_{1a})}(\mathbf{z}) &= \widetilde{LB}^{(V_{1b})}(h_1(\mathbf{z}), z_2, .., z_N),\\
\widetilde{LB}^{(S_{1b})}(\mathbf{z}) &= \widetilde{LB}^{(S_{1a})}(\sum_{k=1}^{N} p_{1k}z_k, ..., \sum_{k=1}^{N} p_{Nk}z_k),\\
\widetilde{LB}^{(V_{2a})}(\mathbf{z}) &= \widetilde{LB}^{(S_{1b})}(\mathbf{z})\widetilde{S}_i(\sum_{j=1}^{N} \lambda_j(1-z_j)),\\
&\vdots\\
\widetilde{LB}^{(V_{Nb})}(\mathbf{z}) &= \widetilde{LB}^{(V_{Na})}(\sum_{k=1}^{N} p_{1k}z_k, ..., \sum_{k=1}^{N} p_{Nk}z_k),\\
\widetilde{LB}^{(S_{Na})}(\mathbf{z}) &= \widetilde{LB}^{(V_{Nb})}(z_1, ..., z_{N-1}, h_N(\mathbf{z})),\\
\widetilde{LB}^{(S_{Nb})}(\mathbf{z}) &= \widetilde{LB}^{(S_{Na})}(\sum_{k=1}^{N} p_{1k}z_k, ..., \sum_{k=1}^{N} p_{Nk}z_k),\\
\widetilde{LB}^{(V_{1a})}(\mathbf{z}) &= \widetilde{LB}^{(S_{Nb})}(\mathbf{z})\widetilde{S}_i(\sum_{j=1}^{N} \lambda_j(1-z_j)).
\end{aligned}
$$

5.2.4 Joint queue length distribution at arbitrary epochs

In the previous section we derived the joint queue length distribution at polling epochs. We can now use these results to derive an expression for the joint queue length distribution at arbitrary epochs.

We start from the following observation made by Eisenberg [39]. All service beginnings at Q_i coincide with a service completion at Q_i, except for the first service beginning which coincides with a visit beginning at Q_i. Similarly, all service completions at Q_i coincide with a service beginning at Q_i, except for the last service completion which coincides with a visit completion at Q_i. As shown by Boxma et al. [40], this observation can be used to derive the joint queue length distribution at an arbitrary epoch.

We first need to calculate the PGF of the joint queue length distribution at service beginnings at Q_i. Let $\widetilde{LS}_i(\mathbf{z})$ and $\widetilde{LD}_i(\mathbf{z})$, respectively, denote the PGF of the joint queue length distribution at service beginnings and service completions at Q_i. Then, using the observation made by Eisenberg [39] and the results of Boxma et al. [40], we obtain the following relation

$$
\frac{1}{\hat{\lambda}_i\mathbb{E}[C]}\widetilde{LB}^{(V_{ib})}(\mathbf{z}) + \widetilde{LD}_i(\mathbf{z}) = \widetilde{LS}_i(\mathbf{z}) + \frac{1}{\hat{\lambda}_i\mathbb{E}[C]}\widetilde{LB}^{(S_{ia})}(\mathbf{z}), \quad i = 1, ..., N, \quad (6)
$$

where $\hat{\lambda}_i$ is the effective arrival rate at Q_i and $\hat{\lambda}_i\mathbb{E}[C]$ is the mean number of customers served at Q_i per visit. It is easily seen that we have the following relation between $\widetilde{LS}_i(\mathbf{z})$ and $\widetilde{LD}_i(\mathbf{z})$

$$\widetilde{LD}_i(\mathbf{z}) = \widetilde{LS}_i(\mathbf{z})\widetilde{B}_i\big(\sum_{j=1}^{N}\lambda_j(1-z_j)\big)/z_i, \quad i=1,...,N. \tag{7}$$

Combining Equations (6) and (7) gives the following

$$\widetilde{LS}_i(\mathbf{z}) = \frac{1}{\hat{\lambda}_i\mathbb{E}[C]}\frac{z_i}{z_i - \widetilde{B}_i\Big(\sum_{j=1}^{N}\lambda_j(1-z_j)\Big)}(\widetilde{LB}^{(V_{ib})}(\mathbf{z}) - \widetilde{LB}^{(S_{ia})}(\mathbf{z})),$$

$$\widetilde{LD}_i(\mathbf{z}) = \frac{1}{\hat{\lambda}_i\mathbb{E}[C]}\frac{\widetilde{B}_i\Big(\sum_{j=1}^{N}\lambda_j(1-z_j)\Big)}{z_i - \widetilde{B}_i\Big(\sum_{j=1}^{N}\lambda_j(1-z_j)\Big)}(\widetilde{LB}^{(V_{ib})}(\mathbf{z}) - \widetilde{LB}^{(S_{ia})}(\mathbf{z})),$$

for $i = 1, ..., N$. We can now use these expressions to derive the joint queue length distribution at arbitrary epochs.

Let $\widetilde{L}(\mathbf{z})$ denote the PGF of the joint queue length distribution at an arbitrary epoch and $\widetilde{L}^{(P)}(\mathbf{z})$ with $P \in \{V_{1a}, V_{1b}, S_{1a}, S_{1b}, ..., V_{Na}, V_{Nb}, S_{Na}, S_{Nb}\}$ denote the PGF of the joint queue length distribution at an arbitrary epoch during P. Then, by the stochastic mean value theorem, we have

$$\widetilde{L}(\mathbf{z}) = \sum_{i=1}^{N}\left(\frac{\mathbb{E}[V_i]}{\mathbb{E}[C]}\widetilde{L}^{(V_{ib})}(\mathbf{z}) + \frac{\mathbb{E}[S_i]}{\mathbb{E}[C]}\widetilde{L}^{(S_{ib})}(\mathbf{z})\right), \tag{8}$$

where we use that $\mathbb{E}[V_{ia}] = \mathbb{E}[S_{ia}] = 0$, $\mathbb{E}[V_{ib}] = \mathbb{E}[V_i]$ and $\mathbb{E}[S_{ib}] = \mathbb{E}[S_i]$, for all $i = 1, ..., N$.

To derive an expression for $\widetilde{L}^{(V_{ib})}(\mathbf{z})$ we use an observation by Boxma et al. [40] stating that the PGF at an arbitrary epoch during period V_{ib}, in which customers get served, is equal to the PGF at an arbitrary epoch during a service time B_i. The number of customer at an arbitrary point of time during service time B_i is equal to the number of customers present at the beginning of service time B_i plus all customers arriving during the elapsed part of the service. Hence, we obtain the following

$$\widetilde{L}^{(V_{ib})}(\mathbf{z}) = \widetilde{LS}_i(\mathbf{z})\widetilde{B}_{i,past}\Big(\sum_{j=1}^{N}\lambda_j(1-z_j)\Big),$$

where $\widetilde{B}_{i,past}(\mathbf{w})$ is the LST of the elapsed part of service time B_i given by

$$\widetilde{B}_{i,past}(\mathbf{w}) = \frac{1-\widetilde{B}_i(\mathbf{w})}{\mathbf{w}\mathbb{E}[B_i]},$$

see [41]. Thus we find the following expression for $\widetilde{L}^{(V_{ib})}(\mathbf{z})$

$$\widetilde{L}^{(V_{ib})}(\mathbf{z}) = \widetilde{LS}_i(\mathbf{z})\frac{1-\widetilde{B}_i\Big(\sum_{j=1}^{N}\lambda_j(1-z_j)\Big)}{\sum_{j=1}^{N}\lambda_j(1-z_j)\mathbb{E}[B_i]}. \tag{9}$$

Similarly, we can define $\widetilde{S}_{i,past}(\mathbf{w})$ as the LST of the elapsed part of switch-over time S_i to obtain the following expression for $\widetilde{L}^{(S_{ib})}$

$$\widetilde{L}^{(S_{ib})}(\mathbf{z}) = \widetilde{LB}^{(S_{ib})}(\mathbf{z})\widetilde{S}_{i,past}\Big(\sum_{j=1}^{N}\lambda_j(1-z_j)\Big) = \frac{\widetilde{LB}^{(S_{ib})}(\mathbf{z}) - \widetilde{LB}^{(V_{(i+1)a})}(\mathbf{z})}{\sum_{j=1}^{N}\lambda_j(1-z_j)\mathbb{E}[S_i]}, \tag{10}$$

where we used $\widetilde{LB}^{(V_{(i+1)a})} = \widetilde{S}_i(\mathbf{z})\widetilde{LB}^{(S_{ib})}(\mathbf{z})$.

By substituting (9) and (10) into (8) we obtain the PGF of the joint queue length distribution at an arbitrary epoch,

$$\begin{aligned}\widetilde{L}(\mathbf{z}) &= \sum_{i=1}^{N}\Bigg(\frac{\mathbb{E}[V_i]}{\mathbb{E}[C]}\frac{1}{\hat{\lambda}_i\mathbb{E}[C]}\frac{z_i(\widetilde{LB}^{(V_{ib})}(\mathbf{z}) - \widetilde{LB}^{(S_{ia})}(\mathbf{z}))}{z_i - \widetilde{B}_i\Big(\sum_{j=1}^{N}\lambda_j(1-z_j)\Big)}\frac{1-\widetilde{B}_i\Big(\sum_{j=1}^{N}\lambda_j(1-z_j)\Big)}{\sum_{j=1}^{N}\lambda_j(1-z_j)\mathbb{E}[B_i]}\\ &+ \frac{\mathbb{E}[S_i]}{\mathbb{E}[C]}\frac{\widetilde{LB}^{(S_{ib})}(\mathbf{z}) - \widetilde{LB}^{(V_{(i+1)a})}(\mathbf{z})}{\sum_{j=1}^{N}\lambda_j(1-z_j)\mathbb{E}[S_i]}\Bigg)\\ &= \frac{1}{\mathbb{E}[C]}\sum_{i=1}^{N}\Bigg(\frac{z_i(\widetilde{LB}^{(V_{ib})}(\mathbf{z}) - \widetilde{LB}^{(S_{ia})}(\mathbf{z}))}{z_i - \widetilde{B}_i\Big(\sum_{j=1}^{N}\lambda_j(1-z_j)\Big)}\frac{1-\widetilde{B}_i\Big(\sum_{j=1}^{N}\lambda_j(1-z_j)\Big)}{\sum_{j=1}^{N}\lambda_j(1-z_j)}\\ &+ \frac{\widetilde{LB}^{(S_{ib})}(\mathbf{z}) - \widetilde{LB}^{(V_{(i+1)a})}(\mathbf{z})}{\sum_{j=1}^{N}\lambda_j(1-z_j)}\Bigg),\end{aligned}$$

where we used $\mathbb{E}[V_i] = \hat{\lambda}_i\mathbb{E}[C][B_i]$.

5.2.5 Mean cycle time and mean visit times

In the previous section we derive the joint queue length distributions at arbitrary epochs. The expression found contains the mean cycle time $\mathbb{E}[C]$. In this section we briefly describe how to obtain the mean cycle time using the LSTs of the distributions of the visit times V_i.

Let $\widetilde{V}_i(w)$ denote the LST of the visit time at Q_i. For any service discipline satisfying the branching property, as described in Section 5.2.1, $\widetilde{V}_i(w)$ can be expressed in terms of the joint queue length distribution at visit beginnings as follows

$$\widetilde{V}_i(w) = \widetilde{LB}^{(V_i)}(1,...,1,\widetilde{\theta}_i(w),1,...,1), i = 1,...,N, \tag{11}$$

where $\widetilde{\theta}_i(w)$ is the LST of the time the server spends at Q_i due to the presence of a single customer. For gated service $\widetilde{\theta}_i(w) = \widetilde{B}_i(w)$, and for exhaustive service $\widetilde{\theta}_i(w) = \widetilde{BP}_i(w)$, where $\widetilde{BP}_i$ is the LST of the duration of a busy period at Q_i.

By differentiating (11) we obtain the mean visit times $\mathbb{E}[V_i]$. A cycle consists of all visit times and switch-over times, $C = V_1 + S_1 + ... + V_N + S_N$. The mean cycle time can thus be obtained using

$$\mathbb{E}[C] = \sum_{i=1}^{N}(\mathbb{E}[V_i] + \mathbb{E}[S_i]).$$

6 Numerical examples polling model

In this section we consider two examples of a polling model with switching customers. In the first example we focus on a two-queue polling model and describe the effects of the parameters. In the second example we consider a two-queue polling model which mimics the situation in *Smart Traffic*.

6.1 Example 1: A two-queue polling model

We consider a two-queue polling model with switching customers as described in the previous section. At the beginning of a polling epoch a customer may stay in Q_1 or Q_2 with probability p_{11} or p_{22}, respectively. All customers which choose not to stay in the queue will join the other queue.

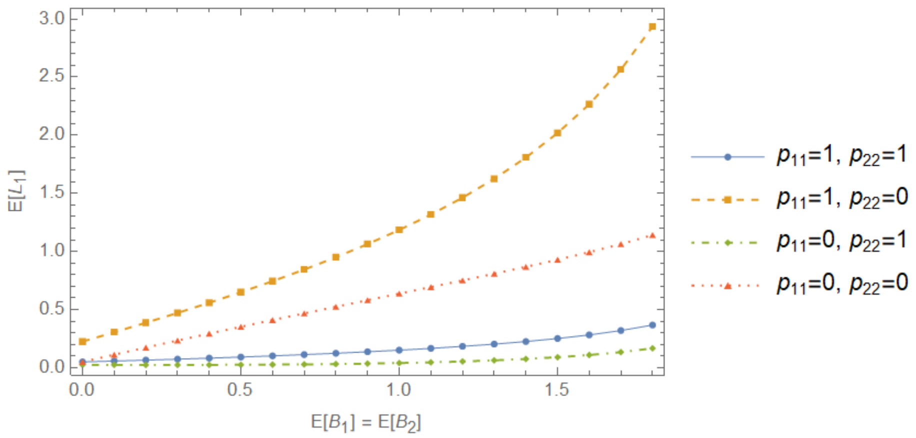


Figure 15: The expected queue length of Q_1 for $\mathbb{E}[S_i] = 1$ and varying values of $\mathbb{E}[B_i]$ under gated service.

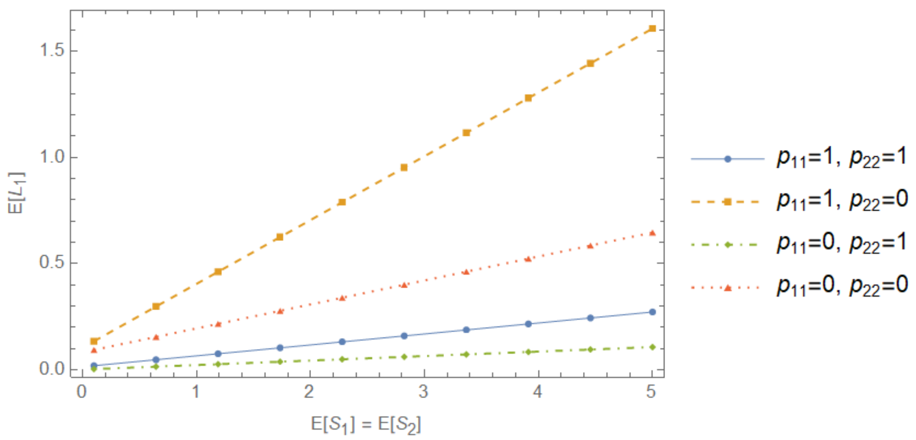
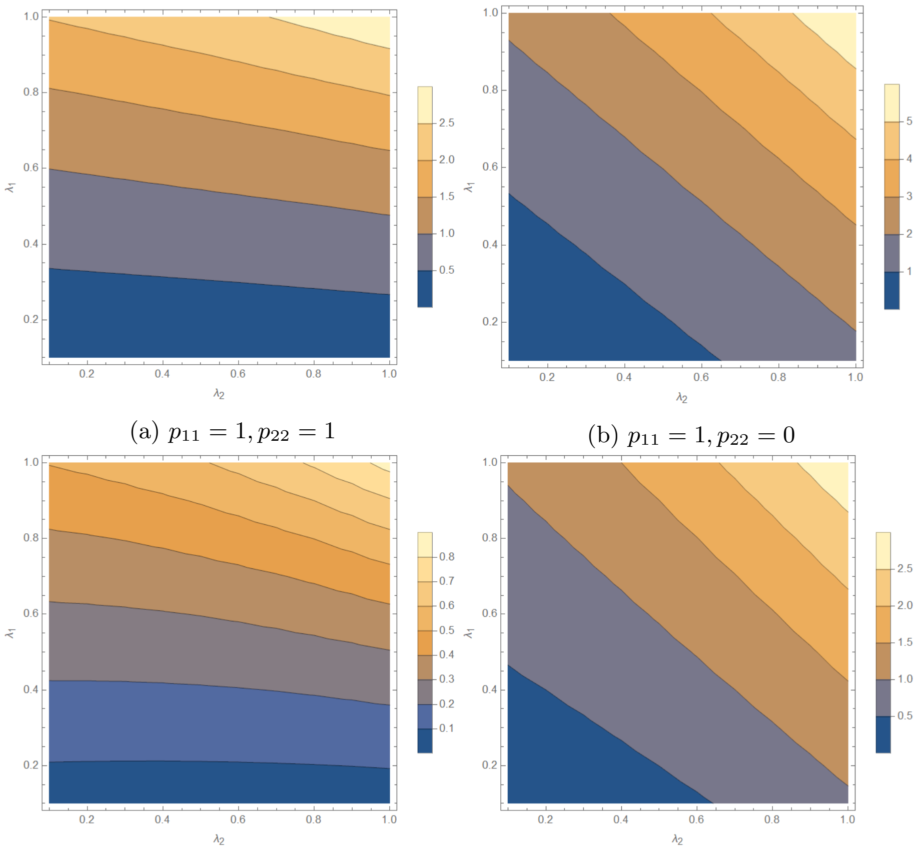


Figure 16: The expected queue length of Q_1 for $\mathbb{E}[B_i] = \frac{1}{4}$ and varying values of $\mathbb{E}[S_i]$ under gated service.

We assume both queues have the same deterministic service times and switch-over times, thus $\mathbb{E}[B_1] = \mathbb{E}[B_2]$ and $\mathbb{E}[S_1] = \mathbb{E}[S_2]$. The arrival processes are Poisson with parameter $\lambda_1 = \frac{1}{20}$ for Q_1 and $\lambda_2 = \frac{7}{20}$ for Q_2. We denote the mean queue length at arbitrary epochs by $\mathbb{E}[L_i]$, $i = 1, 2$.



(a) $p_{11} = 1, p_{22} = 1$

(b) $p_{11} = 1, p_{22} = 0$

(c) $p_{11} = 0, p_{22} = 1$

(d) $p_{11} = 0, p_{22} = 0$

Figure 17: The expected queue length of Q_1 under gated service for $\mathbb{E}[B_i] = \frac{1}{4}$ and $\mathbb{E}[S_i] = 1$.

We study the effect of the model parameters on the mean queue lengths and consider both exhaustive and gated service. In Figure 15 the effect of $\mathbb{E}[B_i]$ on $\mathbb{E}[L_1]$ is shown for gated service. The effect is as expected with $\mathbb{E}[L_1]$ increasing as $\mathbb{E}[B_i]$ increases. In Figure 16 the effect of $\mathbb{E}[S_i]$, again for $\mathbb{E}[L_1]$ under gated service, is shown. $\mathbb{E}[L_1]$ increases linearly as $\mathbb{E}[S_i]$ increases. For completeness, the effect of varying arrival rates λ_1 and λ_2 on $\mathbb{E}[L_1]$ under gated service is shown in Figure 17. When $p_{22} = 1$ all customers in Q_2 stay in Q_2, thus in this case the effect of λ_2 is minimal on the mean queue length of Q_1. We do note that $\mathbb{E}[L_1]$ is higher when λ_2 is higher. As λ_2 increases the time between a visit ending at Q_1 and the subsequent visit beginning at Q_1 increases, due to the increased $\mathbb{E}[V_2]$, leading to an increase in mean queue length at Q_1. When $p_{22} = 0$ all customers in Q_2 switch to Q_1 at polling epochs. The mean queue length of Q_1 is then greatly dependent on the arrival rate at Q_2. This is visible in Figures 17b and 17d. For $\mathbb{E}[L_1]$ under exhaustive service and for $\mathbb{E}[L_2]$ under both exhaustive and gated service, $\mathbb{E}[B_i]$, $\mathbb{E}[S_i]$ and λ_i, with $i = 1, 2$, exhibit similar behaviour.

After showing that the effect of $\mathbb{E}[B_i]$, $\mathbb{E}[S_i]$ and λ_i on $\mathbb{E}[L_i]$ are as expected, we focus on the more interesting parameters of the model p_{11} and p_{22}. We compare the following four systems, with deterministic service and switch-over times:

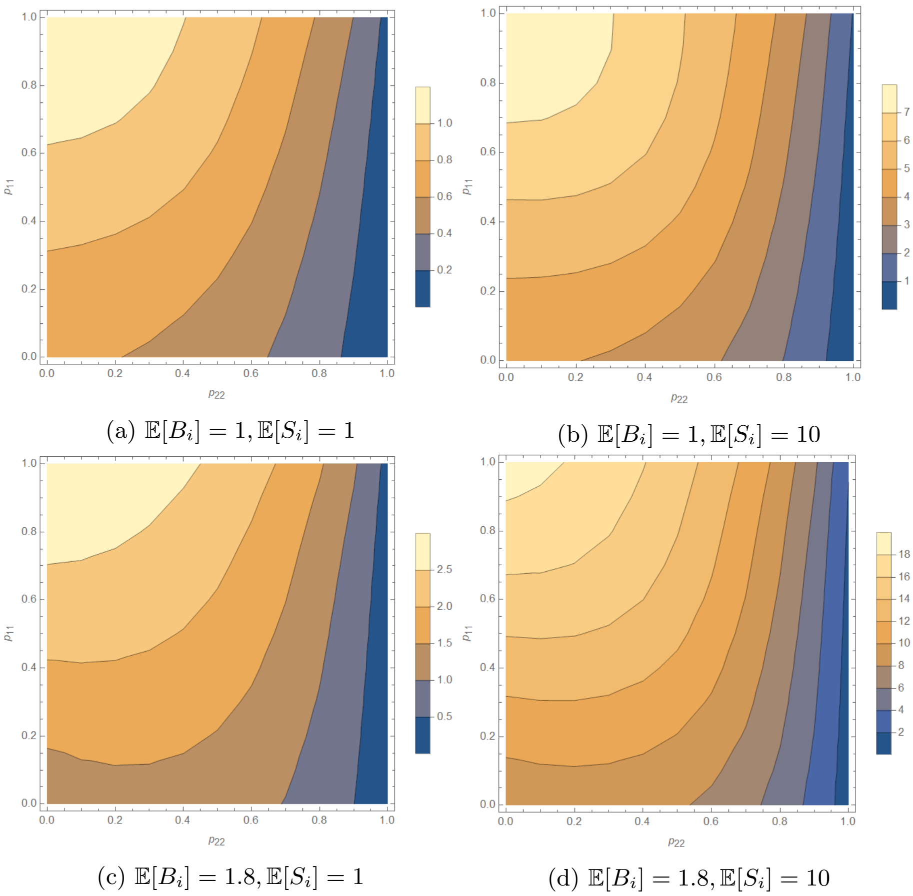
(a) $\mathbb{E}[B_i] = 1$ and $\mathbb{E}[S_i] = 1$,

(b) $\mathbb{E}[B_i] = 1$ and $\mathbb{E}[S_i] = 10$,

(c) $\mathbb{E}[B_i] = 1.8$ and $\mathbb{E}[S_i] = 1$,

(d) $\mathbb{E}[B_i] = 1.8$ and $\mathbb{E}[S_i] = 10$,

with $\lambda_1 = \frac{1}{20}$ and $\lambda_2 = \frac{7}{20}$. In Figures 18-20 we show how the mean queue length depends on p_{11} and p_{22} for the four systems. We omitted the results for $\mathbb{E}[L_1]$ under exhaustive service, as exhaustive service shows similar results to gated service for Q_1.



(a) $\mathbb{E}[B_i] = 1, \mathbb{E}[S_i] = 1$

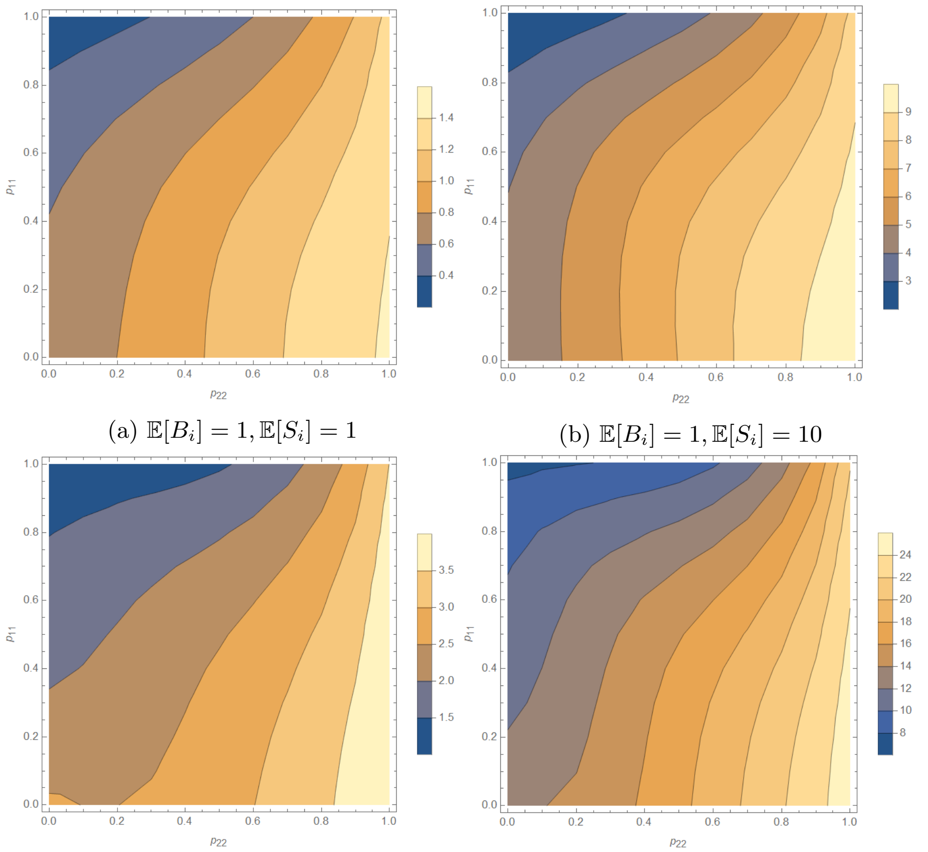
(b) $\mathbb{E}[B_i] = 1, \mathbb{E}[S_i] = 10$

(c) $\mathbb{E}[B_i] = 1.8, \mathbb{E}[S_i] = 1$

(d) $\mathbb{E}[B_i] = 1.8, \mathbb{E}[S_i] = 10$

Figure 18: $\mathbb{E}[L_1]$ under gated service.

In Figure 18 the mean queue length of Q_1 under gated service is shown. We note that the mean queue length is largest when all customers stay in Q_1 and all customers in Q_2 join Q_1. The second largest mean queue length among the four extremes ($p_11 = 0$ or 1 and $p_22 = 0$ or 1) is when all customers decide to change queues. Customer arrive at Q_2 at a much higher rate than at Q_1. Since the customers change queues at polling epochs, customers may arrive at Q_1 with rate λ_1, switch to Q_2 where they are joined by customers with arrival rate λ_2 and then all switch to Q_1 again. Thus the effective arrival rate at Q_1 in case $p_{11} = p_{22} = 0$ is closer to the average of both arrival rates and thus higher than the arrival rate in case $p_{11} = p_{22} = 1$. When all customers decide to leave Q_1 and no customers join from Q_2 the arrival rate is the lowest. Figure 19 shows the results for the mean queue length of Q_2 under gated service.

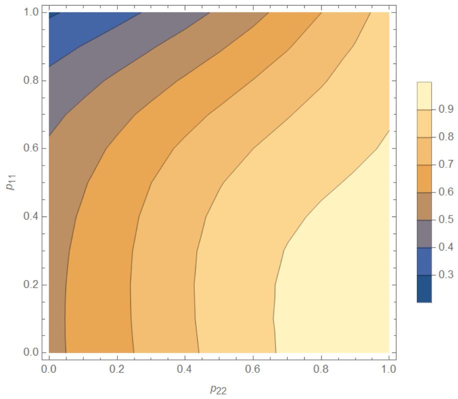


(a) $\mathbb{E}[B_i] = 1, \mathbb{E}[S_i] = 1$ (b) $\mathbb{E}[B_i] = 1, \mathbb{E}[S_i] = 10$

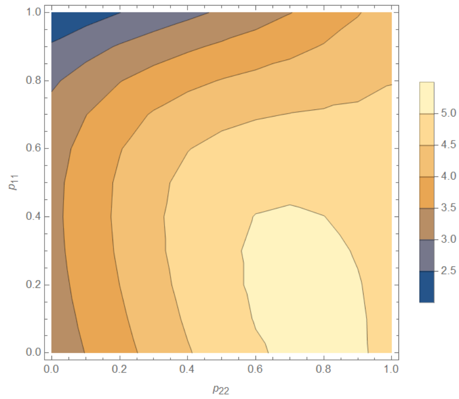
(c) $\mathbb{E}[B_i] = 1.8, \mathbb{E}[S_i] = 1$ (d) $\mathbb{E}[B_i] = 1.8, \mathbb{E}[S_i] = 10$

Figure 19: $\mathbb{E}[L_2]$ under gated service.

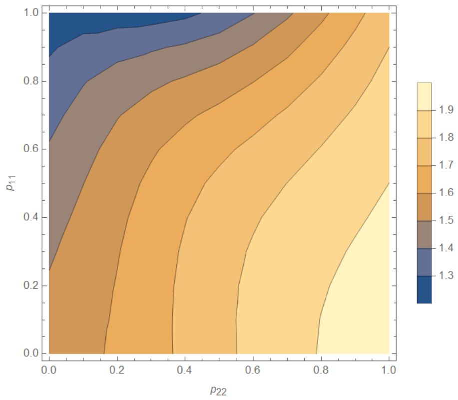
Intuitively, one would expect the mean queue length to decrease as the number of customers opting to leave the queue increases and the number of customers joining the queue decreases and similarly one would expect the mean queue length to increase as the number of customers opting to leave the queue decreases and the number of customers joining the queue increases. However, in system (c), depicted in Subfigure 18c and Subfigure 19c, we can note some non-monotonic behaviour, where for low values p_{11} the expected queue length of Q_1, $\mathbb{E}[L_1]$, may be lower for lower values of p_{22} and the expected queue length of Q_2, $\mathbb{E}[L_2]$ may be higher for lower values of p_{22}.



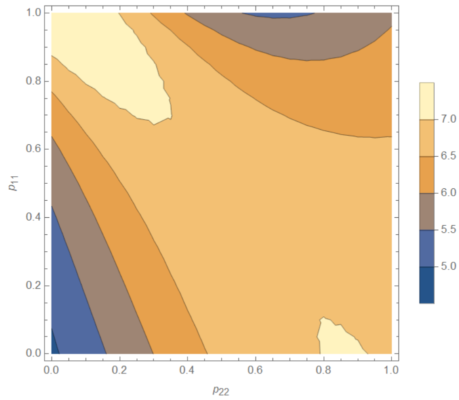
(a) $\mathbb{E}[B_i] = 1, \mathbb{E}[S_i] = 1$



(b) $\mathbb{E}[B_i] = 1, \mathbb{E}[S_i] = 10$



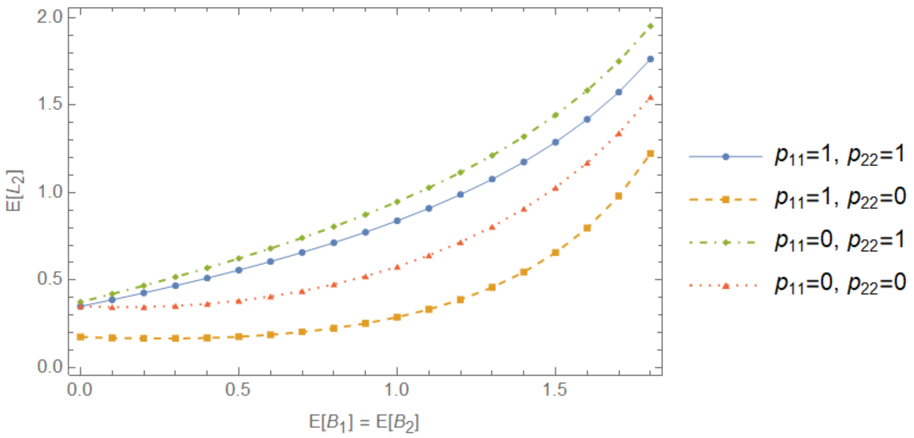
(c) $\mathbb{E}[B_i] = 1.8, \mathbb{E}[S_i] = 1$



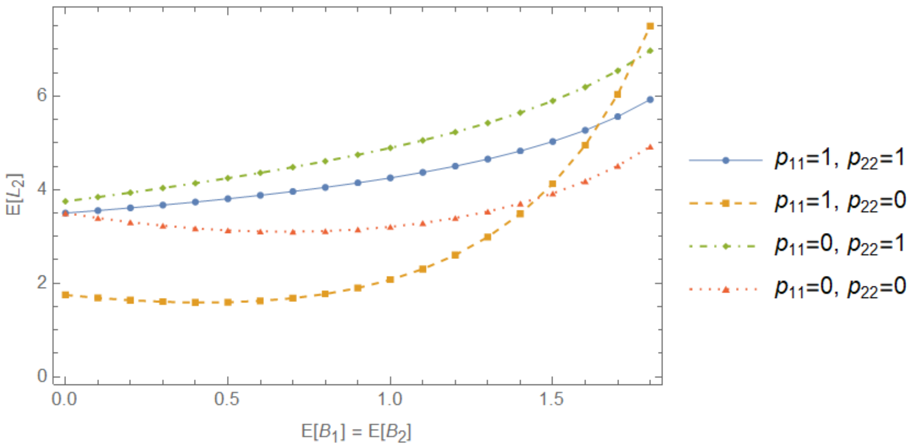
(d) $\mathbb{E}[B_i] = 1.8, \mathbb{E}[S_i] = 10$

Figure 20: $\mathbb{E}[L_2]$ under exhaustive service.

In Figure 20 the results for the mean queue length in Q_2 under the exhaustive discipline are shown. Subfigure 20b shows clear non-monotonic behaviour for lower values of p_{11}. In system (b), $\mathbb{E}[S_i]$ is much larger than $\mathbb{E}[B_i]$, thus the switch-over periods become dominant. The parameters p_{11} and p_{22} do not influence the arrivals during the switch-over periods, as the switching of customers only takes place at polling epochs. When p_{22} is high and p_{11} is low, a large number of customers will stay in Q_2 or join Q_2 from Q_1 at a polling epoch, this means that the visit period of Q_2 will be large. Simultaneously, due to the large switch-over periods and customers leaving Q_1, the amount of customers at the start of a visit period at Q_1 will be small. Consequently, Q_1 has a smaller visit period and thus the the visit period at Q_2 will start sooner. Hence, the mean queue length at Q_2 is smaller if Q_1 has a smaller visit period. As p_{11} increases and p_{22} decreases, the length of the visit period of Q_1 increases, however for high enough values for p_{11} or low enough values for p_{22} this is offset by the (low) amount of customers in Q_2.



(a) $\mathbb{E}[S_i] = 1$



(b) $\mathbb{E}[S_i] = 10$

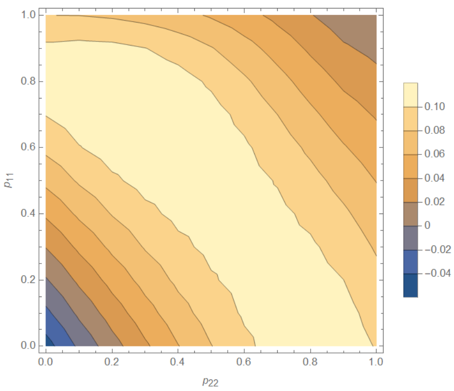
Figure 21: The expected queue length of Q_2 for varying values of $\mathbb{E}[B_i]$ under exhaustive service.

In system (d), where $\mathbb{E}[B_i] = 1.8$ and $\mathbb{E}[S_i] = 10$, seen in Subfigure 20d, we see that the expected queue length of Q_2 is highest for $p_{11} = 1$ and $p_{22} = 0$. Thus all customers leaving Q_2 to join Q_1 and no customers joining Q_1 from Q_2 leads to the highest expected queue length in Q_2. This seems counter-intuitive, as Q_2 is emptied at each polling epoch and in all other cases studied $p_{11} = 1$ and $p_{22} = 0$ gives the lowest expected queue length in Q_2, as we can see in Figure 19 and Subfigures 20a, 20b and 20c. To study this counterintuitive behaviour we plot $\mathbb{E}[L_2]$ for varying rates of $\mathbb{E}[B_i]$ in Figure 21 and note that when $\mathbb{E}[B_i]$ increases $\mathbb{E}[L_2]$ increases more steeply for $p_{11} = 1, p_{22} = 0$ than the other combinations plotted. As described previously, when $\mathbb{E}[S_i]$ is large, a large number of customers will accumulate in both Q_1 and Q_2 and the parameters p_{11} and p_{22} will not influence the arrivals during the switch-over periods. Due to the customers arriving during the long switch-over periods and all customers from Q_2 joining Q_1, the busy period at Q_1 will start with a large number of customers. Unlike system (b), system (d) additionally has large mean service times, combined with the exhaustive service discipline, this leads to an even longer busy period at Q_1. During the busy period of Q_1, customers will accumulate at Q_2. These customers will not be served during the busy period, since the server is at Q_1, and will not leave to join Q_1 until the next polling epoch. This explains the behaviour visible in Figure 21 and Subfigure 20d.

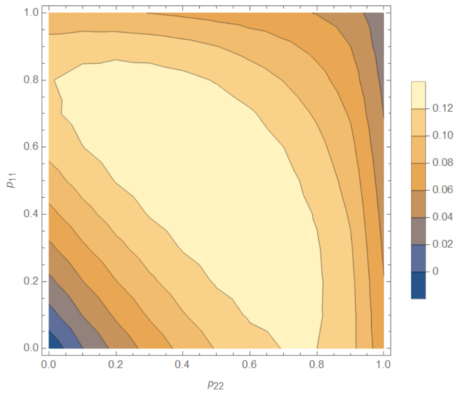
We also briefly take a look at the correlation between the queue length of Q_1 and the queue length of Q_2. The Pearson correlation coefficient is defined as

$$\rho_{L_1,L_2} = \frac{cov(L_1, L_2)}{\sigma_{L_1}\sigma_{L_2}},$$

where $cov(L_1, L_2)$ is the covariance, σ_{L_1} is the standard deviation of L_1 and σ_{L_2} is the standard deviation of L_2 [42]. In Figure 22 the correlation coefficient can be seen for system (a), which has $\mathbb{E}[B_i] = 1$ and $\mathbb{E}[S_i] = 1$, under both exhaustive and gated service. We note that the correlation coefficient is higher under gated service.



(a) Exhaustive service



(b) Gated service

Figure 22: Correlation coefficient between the queue length of Q_1 and the queue length of Q_2 for $\mathbb{E}[B_i] = 1$ and $\mathbb{E}[S_i] = 1$ under exhaustive and gated service.

6.2 Example 2: Smart Traffic

In the polling model with switching customers, as described in Section 5, customers may change queues at every polling instance. In *Smart Traffic* the vehicle or customers may change queues, when new information on their position is available. It is unlikely that new information will lead to a vehicle switching back to a position it previously was. Hence, it is unlikely that a customer changes queues twice or more often.

In this example we will focus on a two-queue polling system in which customers may only change queues once. To obtain a system which satisfies this condition we require that customers may not leave one of the queues, thus we assume $p_{22} = 1$. The arrival processes to both queues are Poisson with parameter $\frac{1}{6}$. We further assume both queues have the same service times and switch-over times, with $\mathbb{E}[B_1] = \mathbb{E}[B_2] = 2$ and $\mathbb{E}[S_1] = \mathbb{E}[S_2] = 3$.

In Table 6 and 7 we see the expected queue length at the beginning of period $P \in \{V_{1a}, V_{1b}, S_{1a}, S_{1b}, V_{2a}, V_{2b}, S_{2a}, S_{2b}\}$ under exhaustive service for Q_1

	V_{1a}	V_{1b}	S_{1a}	S_{1b}	V_{2a}	V_{2b}	S_{2a}	S_{2b}
$p_{11} = 1$	2	2	0	0	0.5	0.5	1.5	1.5
$p_{11} = 0.9$	1.93	1.73	0	0	0.5	0.45	1.58	1.43
$p_{11} = 0.8$	1.83	1.47	0	0	0.5	0.4	1.67	1.33
$p_{11} = 0.7$	1.72	1.21	0	0	0.5	0.35	1.75	1.22
$p_{11} = 0.6$	1.59	0.956	0	0	0.5	0.3	1.82	1.09
$p_{11} = 0.5$	1.44	0.722	0	0	0.5	0.25	1.89	0.944
$p_{11} = 0.4$	1.28	0.511	0	0	0.5	0.2	1.94	0.778
$p_{11} = 0.3$	1.1	0.329	0	0	0.5	0.15	1.99	0.596
$p_{11} = 0.2$	0.902	0.18	0	0	0.5	0.1	2.01	0.402
$p_{11} = 0.1$	0.701	0.0701	0	0	0.5	0.05	2.01	0.201
$p_{11} = 0$	0.5	0	0	0	0.5	0	2	0

Table 6: The expected queue length of Q_1 at the beginning of period $P \in \{V_{1a}, V_{1b}, S_{1a}, S_{1b}, V_{2a}, V_{2b}, S_{2a}, S_{2b}\}$ under exhaustive service.

	V_{1a}	V_{1b}	S_{1a}	S_{1b}	V_{2a}	V_{2b}	S_{2a}	S_{2b}
$p_{11} = 1$	0.5	0.5	1.5	1.5	2	2	0	0
$p_{11} = 0.9$	0.658	0.851	1.72	1.72	2.22	2.27	0	0.158
$p_{11} = 0.8$	0.833	1.2	2.93	1.93	2.43	2.53	0	0.333
$p_{11} = 0.7$	1.02	1.54	2.14	2.14	2.64	2.79	0	0.524
$p_{11} = 0.6$	1.23	1.87	2.34	2.34	2.84	3.04	0	0.729
$p_{11} = 0.5$	1.44	2.17	2.53	2.53	3.03	3.28	0	0.944
$p_{11} = 0.4$	1.67	2.43	2.69	2.69	3.19	3.49	0	1.17
$p_{11} = 0.3$	1.189	2.66	2.82	2.82	3.32	3.67	0	1.39
$p_{11} = 0.2$	2.11	2.83	2.92	2.92	3.42	3.82	0	1.61
$p_{11} = 0.1$	2.31	2.94	2.98	2.98	3.48	3.93	0	1.81
$p_{11} = 0$	2.5	3	3	3	3.5	4	0	2

Table 7: The expected queue length of Q_2 at the beginning of period $P \in \{V_{1a}, V_{1b}, S_{1a}, S_{1b}, V_{2a}, V_{2b}, S_{2a}, S_{2b}\}$ under exhaustive service.

and Q_2 respectively. As expected the expected queue length of Q_1 is higher (or equal) at the start of a subperiod a than the subsequent subperiod b. As, during subperiod a, the customers in Q_1 will leave the queue with probability $1-p_{11}$ to join Q_2. Similarly, for Q_2 the expected queue length is higher (or equal) at the start of a subperiod b than the previous subperiod a. We further note that the sum of the expected queue lengths over both queues at the beginning of period a is equal to the sum of the expected queue lengths over both queues at the beginning of period b, thus we conclude that the system behaves as expected.

We can compare these results with the expected queue length at the beginning of period $P \in \{V_{1a}, V_{1b}, S_{1a}, S_{1b}, V_{2a}, V_{2b}, S_{2a}, S_{2b}\}$ under gated service seen in Table 8 and 9. We note that when $p_{11} = 0$, the expected queue length at the beginning of the periods for Q_1 is the same under exhaustive service and gated service, as all customers leave Q_1 before service. We further note that the system also behaves as expected for gated service.

During simulation vehicles will be assigned to a signal group and corresponding queue. It might only become apparent that a vehicle is in another queue, when new information on the position of the vehicle becomes available through

	V_{1a}	V_{1b}	S_{1a}	S_{1b}	V_{2a}	V_{2b}	S_{2a}	S_{2b}
$p_{11} = 1$	3	3	1	1	1.5	1.5	2.5	2.5
$p_{11} = 0.9$	2.57	2.32	0.772	0.695	1.19	1.08	2.3	2.07
$p_{11} = 0.8$	2.25	1.8	0.599	0.479	0.979	0.784	2.18	1.75
$p_{11} = 0.7$	1.98	1.39	0.462	0.323	0.823	0.576	2.11	1.48
$p_{11} = 0.6$	1.75	1.05	0.349	0.21	0.71	0.426	2.08	1.25
$p_{11} = 0.5$	1.53	0.765	0.255	0.127	0.627	0.314	2.06	1.03
$p_{11} = 0.4$	1.32	0.528	0.176	0.0704	0.57	0.228	2.05	0.821
$p_{11} = 0.3$	1.11	0.334	0.111	0.0334	0.533	0.16	2.05	0.615
$p_{11} = 0.2$	0.908	0.182	0.0606	0.0121	0.512	0.102	2.04	0.408
$p_{11} = 0.1$	0.703	0.0703	0.0234	0.00234	0.502	0.0502	2.03	0.203
$p_{11} = 0$	0.5	0	0	0	0.5	0	2	0

Table 8: The expected queue length of Q_1 at the beginning of period $P \in \{V_{1a}, V_{1b}, S_{1a}, S_{1b}, V_{2a}, V_{2b}, S_{2a}, S_{2b}\}$ under gated service.

	V_{1a}	V_{1b}	S_{1a}	S_{1b}	V_{2a}	V_{2b}	S_{2a}	S_{2b}
$p_{11} = 1$	1.5	1.5	2.5	2.5	3	3	1	1
$p_{11} = 0.9$	1.96	2.22	2.99	3.06	3.56	3.68	1.23	1.46
$p_{11} = 0.8$	2.34	2.79	3.39	3.51	4.01	4.2	1.4	1.84
$p_{11} = 0.7$	2.67	3.27	3.73	3.87	4.37	4.61	1.54	2.17
$p_{11} = 0.6$	2.98	3.68	4.03	4.17	4.67	4.95	1.65	2.48
$p_{11} = 0.5$	3.27	4.04	4.29	4.42	4.92	5.24	1.75	2.77
$p_{11} = 0.4$	3.56	4.35	4.52	4.63	5.13	5.47	1.82	3.06
$p_{11} = 0.3$	3.82	4.6	4.71	4.79	5.29	5.67	1.89	3.32
$p_{11} = 0.2$	4.07	4.8	4.86	4.91	5.41	5.82	1.94	3.57
$p_{11} = 0.1$	4.3	4.93	4.96	4.98	5.48	5.93	1.98	3.8
$p_{11} = 0$	4.5	5	5	5	5.5	6	2	4

Table 9: The expected queue length of Q_2 at the beginning of period $P \in \{V_{1a}, V_{1b}, S_{1a}, S_{1b}, V_{2a}, V_{2b}, S_{2a}, S_{2b}\}$ under gated service.

vehicles passing the detector near the stop line. As the detector may register fewer or more vehicles passing. This information only becomes available during period V_{1b} or V_{2b}. *Smart Traffic* will predict the traffic image and calculate the schedule based on information available prior to V_{1b} or V_{2b}. In the most extreme case $p_{11} = 0$, all vehicles will leave Q_1, and thus signal group 1, to join Q_2, and thus signal group 2. In this case *Smart Traffic* will sometimes assume the presence of some vehicles at signal group 1, as the expected queue length of Q_1 is 0.5 at the start of a visit period. The schedule calculated will then assign a non-zero green time to signal group 1, whereas the optimal schedule would assign no green time to signal group 1. For situations in which p_{11} is higher than 0, but unequal to 1, this also occurs to a lesser extent and the signal may stay green longer than optimal.

In the situation posed we only consider a two-queue polling system with cyclic routing. In *Smart Traffic* the routing is not cyclic, but is determined by calculating the most optimal scheme. The lack of information or incorrect information may thus not only lead to a signal having the non-optimal green time, but also to the incorrect or non-optimal signal receiving green time.

7 Conclusion and discussion

In this thesis we presented a discrete event model to predict traffic at a signalised intersection, as a replacement of microsimulation model SUMO used in the *Forecasting* module of *Smart Traffic*. The SUMO model currently used by Sweco is not suitable for the objective of Sweco to predict traffic over a longer period of time for a network of junctions due to its slow computation speed and complexity.

The performance of the model was measured by comparing it to the microsimulation traffic flow model Vissim. We found a maximum average absolute difference in delay between our discrete event model and the Vissim model per road user of 1.5 seconds.

The accuracy of the model is determined by two main components: the arrival at queue time and the chosen parameter values. The arrival at queue time is the predicted time at which a road user arrives at the queue. It is calculated during initialisation, by dividing the maximum distance a road user is required to travel and dividing this by the maximum speed. The maximum distance a road user is required to travel is assumed to be the position of the road user minus the total length (and gaps) of the road users ahead. We considered several other options for calculating the arrival at queue time, e.g. dividing by the speed of the road user at the beginning of the simulation rather than the maximum speed and using only the position of the road user. The chosen calculation of the arrival at queue time gave the best results during simulation. However, other more accurate calculations may be available. Due to time constraints and the possibility of overfitting, as we only consider one type of junction, we did not study this further.

We use data from Vissim to determine the parameters of the model. If we were to implement our discrete event model at a real junction, these parameters will have to be determined using real-life data from the junction. Traffic may behave very differently in different countries or cities and under different circumstances, such as weather conditions. If the model is not sufficiently calibrated to the real-life situation, the difference between the calculated the delay and the real-life delay may be much larger than the average 1.5 seconds found. The current parameter values are static, however it may be beneficial to investigate methods to adapt the parameter values based on real-time feedback.

In Section 4 we discussed the results of our discrete event simulation. One of the assumptions we made was that the signal group to which a road user gets assigned is known. However, one of the challenges of *Smart Traffic* is precisely the lack of knowledge on the vehicle routing. We described and studied the effect of this lack of knowledge in Sections 5 and 6, by modelling a signalised intersection as a polling model with switching customers, and concluded that it may lead to a non-optimal schedule. Currently, the simulation in SUMO chooses the vehicle routing based on the link a road user is located at, with every outgoing link having a predetermined chance of being reached. In our discrete event model we make use of an array of probabilities. In this way the vehicle routing is an attribute of the road user, rather than an attribute of the road. This makes it possible to include additional (historic) information to determine the vehicle routing, such as previous locations of the road user and public transport timetables. Furthermore, the discrete event model presented is fast, which makes it possible to perform multiple runs and average over the results.

The discrete event model presented in this thesis is easily extendable to include multiple junctions. Furthermore, it is less computationally expensive, more scalable and faster than SUMO, with an average runtime of maximal 1.416 seconds, and thus suitable to the objective of Sweco.

References

[1] Sweco, "Sweco annual report 2020," www.swecogroup.com/wp-content/uploads/sites/2/2021/04/sweco-annual-report-2020.pdf, 2021.

[2] Sweco.nl, "Grontmij heeft een nieuwe naam sweco," https://www.sweco.nl/actueel/nieuws/grontmij-heeft-een-nieuwe-naam-sweco/, accessed: June 30, 2021.

[3] S. Pandian, S. Gokhale, and A. K. Ghoshal, "Evaluating effects of traffic and vehicle characteristics on vehicular emissions near traffic intersections," *Transportation Research Part D: Transport and Environment*, vol. 14, no. 3, pp. 180–196, 2009.

[4] C. Dobre, A. Szekeres, F. Pop, and V. Cristea, "Intelligent traffic lights to reduce vehicle emissions," *Proceedings - 26th European Conference on Modelling and Simulation, ECMS 2012*, 2012.

[5] Sweco.nl, "Smart mobility voor een leefbare stad en minder co2-uitstoot," https://www.sweco.nl/innovaties/Smart-Mobility-voor-een-leefbare-stad-en-minder-CO2-uitstoot/, accessed: June 30, 2021.

[6] A. Al-Mudhaffar, K.-L. Bång, and N. M. P. Rouphail, *Impacts of Traffic Signal Control Strategies.* KTH, 2006.

[7] M. Papageorgiou, C. Diakaki, V. Dinopoulou, A. Kotsialos, and Y. Wang, "Review of road traffic control strategies," *Proceedings of the IEEE*, vol. 91, no. 12, pp. 2043–2067, 2003.

[8] A. J. Miller, "Settings for fixed-cycle traffic signals," *OR*, vol. 14, no. 4, pp. 373–386, 1963.

[9] A. Hamilton, B. Waterson, T. Cherrett, A. Robinson, and I. Snell, "The evolution of urban traffic control: changing policy and technology," *Transportation Planning and Technology*, vol. 36, no. 1, pp. 24–43, 2013.

[10] M. C. Bell and R. Bretherton, "Ageing of fixed-time traffic signal plans," in *International conference on road traffic control*, 1986.

[11] M. Papageorgiou, M. Ben-Akiva, J. Bottom, P. Bovy, S. Hoogendoorn, N. Hounsell, A. Kotsialos, and M. McDonald, "Its and traffic management," in *Transportation.* Elsevier, 2007, vol. 14, pp. 715–774.

[12] M. Treiber and A. Kesting, *Trajectory and Floating-Car Data.* Springer Berlin Heidelberg, 2013, pp. 7–12.

[13] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems.* Springer US, 2008.

[14] S. H. Jacobson, S. N. Hall, and J. R. Swisher, *Discrete-Event Simulation of Health Care Systems.* Springer US, 2006, pp. 211–252.

[15] S.-Y. D. Wu and R. A. Wysk, "An application of discrete-event simulation to on-line control and scheduling in flexible manufacturing," *International Journal of Production Research*, vol. 27, no. 9, pp. 1603–1623, 1989.

[16] M. Fellendorf and P. Vortisch, "Validation of the microscopic traffic flow model vissim in different real-world situations," in *Transportation Research Board (TRB)*, 2001.

[17] C. Mack, T. Murphy, and N. L. Webb, "The efficiency of n machines uni-directionally patrolled by one operative when walking time and repair times are constants," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 19, no. 1, pp. 166–172, 1957.

[18] H. Takagi, "Application of polling models to computer networks," *Computer Networks and ISDN Systems*, vol. 22, no. 3, pp. 193–211, 1991.

[19] A. Federgruen and Z. Katalan, "The stochastic economic lot scheduling problem: Cyclical base-stock policies with idle times," *Management Science*, vol. 42, no. 6, pp. 783–796, 1996.

[20] M. Boon, R. van der Mei, and E. Winands, "Applications of polling systems," *Surveys in Operations Research and Management Science*, vol. 16, no. 2, pp. 67–82, 2011.

[21] H. Takagi, "Analysis and application of polling models," *Lecture Notes in Computer Science*, vol. 1769, pp. 423–442, 2000.

[22] O. Boxma, J. Ivanovs, K. Kosinski, and M. Mandjes, "Lévy-driven polling systems and continuous-state branching processes," *Stochastic Systems*, vol. 1, no. 2, pp. 411–436, 2011.

[23] R. van der Mei and E. Winands, "Polling models with renewal arrivals: A new method to derive heavy-traffic asymptotics," *Performance Evaluation*, vol. 64, no. 9, pp. 1029–1040, 2007.

[24] R. van der Mei, "Polling systems with simultaneous batch arrivals," *Stochastic Models*, vol. 17, no. 3, pp. 271–292, 2001.

[25] H. Levy and M. Sidi, "Polling systems with simultaneous arrivals," *IEEE Transactions on Communications*, vol. 39, no. 6, pp. 823–827, 1991.

[26] M. Boon, "A polling model with reneging at polling instants," *Annals of Operations Research*, vol. 198, no. 1, pp. 5–23, 2012.

[27] I. Adan, A. Economou, and S. Kapodistria, "Synchronized reneging in queueing systems with vacations," *Queueing Systems: Theory and Applications*, vol. 62, no. 1, pp. 1–33, 2009.

[28] S. Borst, "Polling systems with multiple coupled servers," *Queueing Systems*, vol. 20, pp. 369–393, 1995.

[29] M. Vlasiou and U. Yechiali, "M/g/∞ polling systems with random visit times," *Probability in the Engineering and Informational Sciences*, vol. 22, no. 1, p. 81–106, 2008.

[30] O. Boxma, J. Wal, van der, and U. Yechiali, "Polling with batch service," *Stochastic Models*, vol. 24, no. 4, pp. 604–625, 2008.

[31] O. Boxma, H. Levy, and U. Yechiali, "Cyclic reservation schemes for efficient operation of multiple-queue single-server systems," *Annals of Operations Research*, vol. 35, pp. 187–208, 1992.

[32] S. Borst, *Polling systems.* Centrum voor Wiskunde en Informatica, 1996.

[33] S. Borst and O. Boxma, "Polling models with and without switchover times," *Operations Research*, vol. 45, no. 4, pp. 536–543, 1997.

[34] T. L. Olsen and R. van der Mei, "Polling systems with periodic server routeing in heavy traffic: distribution of the delay," *Journal of Applied Probability*, vol. 40, no. 2, p. 305–326, 2003.

[35] O. Boxma and J. Weststrate, "Waiting times in polling systems with markovian server routing," in *Messung, Modellierung und Bewertung von Rechensystemen und Netzen.* Springer Berlin Heidelberg, 1989, pp. 89–104.

[36] M. Boon, A. Wijk, van, I. Adan, and O. Boxma, "A polling model with smart customers," *Queueing Systems: Theory and Applications*, vol. 66, no. 3, pp. 239–274, 2010.

[37] J. Resing, "Polling systems and multitype branching processes," *Queueing Systems: Theory and Applications*, vol. 13, no. 4, pp. 409–426, 1993.

[38] R. B. Cooper and G. Murray, "Queues served in cyclic order," *The Bell System Technical Journal*, vol. 48, no. 3, pp. 675–689, 1969.

[39] M. Eisenberg, "Queues with periodic service and changeover time," *Operations Research*, vol. 20, no. 2, pp. 440–451, 1972.

[40] O. Boxma, O. Kella, and K. Kosinski, "Queue lengths and workloads in polling systems," *Operations Research Letters*, vol. 39, no. 6, pp. 401 – 405, 2011.

[41] I. Adan and M. Haviv, "Conditional ages and residual service times in the m/g/1 queue," *Stochastic Models*, vol. 25, pp. 110–128, 2009.

[42] J. Benesty, J. Chen, Y. Huang, and I. Cohen, "Pearson correlation coefficient," in *Noise reduction in speech processing.* Springer, 2009, pp. 1–4.

A Appendix: Code

A.1 Discrete event simulation

A.1.1 Simulation files

The Road_User class.

```
package swecobasicmodelextended;

public class Road_User {
    /* Integer denoting the id number of the road_user */
    protected int id;
    /* Double denoting the arrival time of the road user */
    protected double arrivalTime;
    /* Double denoting the departure time of the road user */
    protected double departureTime;
    /* Double denoting the length of the road user */
    protected double length; //
    /* String denoting the type of the road user */
    protected String type;
    /* String denoting the arrival place */
    protected String arrivalPlace;
    /* Double denoting the postion of the road user from the
        stop line in meters */
    protected double position;
    /* Array of probabilities for determining the signal group
        */
    protected double[] probabilities;
    /* Double denoting the speed of the road user */
    protected double speed;
    /* Double denoting the service time of the first vehicle */
    protected double serviceTimeFirstVehicle;
    /* Double denoting the service time */
    protected double serviceTime;
    /* Boolean indicating whether the road user has left the
        system */
    protected boolean leftSystem;
    /* Boolean indication whether the road user is in the queue
         */
    protected boolean inQueue;
    /* Integer denoting the number of stops */
    protected int nrStops;
    /* Integer denoting the number in line */
    protected int nrInLine;
     /* Double denoting the desired speed of the road user (in
         Vissim) */
    protected double desSpeed;


    public Road_User(int id, double arrivalTime, double length,
         String type, String arrivalPlace, double position,
        double[] probabilities, double speed, double desSpeed,
        int nrInLine){
        this.id = id;
```

```
        this.arrivalTime = arrivalTime;
        this.departureTime = 0;
        this.length = length;
        this.type = type;
        this.leftSystem = false;
        this.inQueue = false;
        this.arrivalPlace = arrivalPlace;
        this.position = position;
        this.probabilities = probabilities;
        this.speed = speed;
        this.nrStops = 0;
        this.nrInLine = 0;
        this.serviceTime = 0;
        this.desSpeed = desSpeed;
        this.nrInLine = nrInLine;
    }
    public int getId(){ //Returns id number
        return id;
    }
    public double getArrivalTime() {//Returns arrival time
        return arrivalTime;
    }
    public double getDepartureTime(){ //Returns the departure
        time
        return departureTime;
    }
    public double getLength() {//Returns length
        return length;
    }
    public String getType(){//Returns type
        return type;
    }
    public String getArrivalPlace(){ // Returns arrivalplace
        return arrivalPlace;
    }
    public double getPosition(){ // Returns position
        return position;
    }
    public double[] getProb(){ // Returns probability array
        return probabilities;
    }
    public double getSpeed(){ // Returns speed
        return speed;
    }
    public double getServiceTime(){ // Returns the service time
        switch (nrInLine) {
            case 1:
                if (null != type)switch (type) { //Determine (
                    residual)serviceTime based on type and
                    nrInLine
                    case "PASSENGER":
                        this.serviceTime = 1.10566;
                        break;
                    case "TRUCK":
```

```
                this.serviceTime = 1.164;
                break;
            }   break;
        case 2:
            if (null != type)switch (type) { //Determine (
                residual)serviceTime based on type and
                nrInLine
            case "PASSENGER":
                this.serviceTime = 2.718868;
                break;
            case "TRUCK":
                this.serviceTime = 3.845;
                break;
            }   break;
        case 3:
            if (null != type)switch (type) { //Determine (
                residual)serviceTime based on type and
                nrInLine
            case "PASSENGER":
                this.serviceTime = 2.10566;
                break;
            case "TRUCK":
                this.serviceTime = 2.84;
                break;
            }   break;
        case 4:
            if (null != type)switch (type) { //Determine (
                residual)serviceTime based on type and
                nrInLine
            case "PASSENGER":
                this.serviceTime = 1.840566;
                break;
            case "TRUCK":
                this.serviceTime = 2.556;
                break;
            }   break;
        case 5:
            if (null != type)switch (type) { //Determine (
                residual)serviceTime based on type and
                nrInLine
            case "PASSENGER":
                this.serviceTime = 1.835849;
                break;
            case "TRUCK":
                this.serviceTime = 2.529;
                break;
            }   break;
         case 6:
            if (null != type)switch (type) { //Determine (
                residual)serviceTime based on type and
                nrInLine
            case "PASSENGER":
                this.serviceTime = 1.726415;
                break;
```

```
                    case "TRUCK":
                        this.serviceTime = 2.377;
                        break;
                }   break;
            case 7:
                if (null != type)switch (type) { //Determine (
                    residual)serviceTime based on type and
                    nrInLine
                    case "PASSENGER":
                        this.serviceTime = 1.711321;
                        break;
                    case "TRUCK":
                        this.serviceTime = 2.354;
                        break;
                }   break;
            default: //Vehicles after 7
                if (null != type)switch (type) { //Determine (
                    residual)serviceTime based on type and
                    nrInLine
                    case "PASSENGER":
                        this.serviceTime = 1.674528;
                        break;
                    case "TRUCK":
                        this.serviceTime = 2.286;
                        break;
                }   break;
        }
        return serviceTime;
    }

    public void updatePosition(double d){ // Updates position
        to input value
        position = d;
    }
    public void leaveSystem(double t){ //Sets departure time to
         input value and set boolean leftSystem to true
        departureTime = t;
        leftSystem = true;
    }
    public boolean hasLeftSystem(){ // Returns leftSystem
        return leftSystem;
    }
    public boolean inQueue(){ // Returns inQueue
        return inQueue;
    }
    public void addStop(){ // Increase nrStops by one
        nrStops += 1;
    }
    public int getNrStops(){ // Returns nrStops
        return nrStops;
    }
    public void nrInLine(int d){ // Sets the number in line
        nrInLine = d;
    }
```

```java
    public int getNrInLine(){ // Returns the number in line
        return nrInLine;
    }

    public double getDesSpeed(){ // Returns the desired speed
        return desSpeed;
    }
}
```

The Signal_Group class.

```java
package swecobasicmodelextended;

import java.util.ArrayList;

public class Signal_Group {
    /* Traffic light colours */
    public static final int RED = 1;
    public static final int AMBER = 2;
    public static final int GREEN = 3;
     /* Double denoting the distance in meter between two road
         users waiting in the queue */
    public static double gap = 2;
    /* Integer indentifing the signal group */
    protected int identifier;
    /* Double denoting the travel time from arrival in the
        system to arrival at the traffic light of the signal
        group */
    protected double length;
    /* Double denoting the maximum speed on the lane associated
         with the signal group */
    protected double maxSpeed;
    /* Integer denoting the colour of the traffic light */
    protected int trafficLightColour;
    /* List of road users associated with signal group */
    protected ArrayList<Road_User> road_users;
    /* List of road users in the queue */
    protected ArrayList<Road_User> road_users_in_queue;
    /* Doubles denoting the size of the queue in meter and the
        distance (in meter) from the trafficlight to the front
        and back of the queue */
    protected double queueLengthInMeter;
    protected double frontOfQueue;
    protected double backOfQueue;
    /* Boolean idicating whether the signal group has a queue
        at initialisation */
    protected boolean hasQueue;
    /* Boolean idicating whether a road user has been below the
         speed boundary (departure boundary) */
    protected boolean checked;

    public Signal_Group(int identifier, double length, double
        maxSpeed){
        this.identifier = identifier;
        this.length = length;
```

```java
        this.trafficLightColour = RED;
        this.road_users = new ArrayList<>();
        this.road_users_in_queue = new ArrayList<>();
        this.queueLengthInMeter = 0;
        this.frontOfQueue = 0;
        this.backOfQueue = 0;
        this.maxSpeed = maxSpeed;
        this.hasQueue = false;
        this.checked = false;
    }
    public int getIdentifier(){ // Returns the identifier
        return identifier;
    }
    public int trafficLightColour() { //Returns the current traffic light colour
        return trafficLightColour;
    }
    public void changeTrafficLightColour(int newTrafficLightColour){ //Changes the traffic light colour to the new traffic light colour
        trafficLightColour = newTrafficLightColour;
    }
    public double getLength(){ //Returns the travel time
        return length;
    }
    public double maxSpeed(){ //Returns the maximum speed on the lane
        return maxSpeed;
    }
    public void addRoadUser(Road_User ru){//Add road user to the signal groups
        road_users.add(ru);
    }
    public int nrRoadUsers(){ // Returns the number of road users associated with the signal group
        return road_users.size();
    }
     public Road_User getRoadUser(int i){ // Returns the i'th road user
        return road_users.get(i);
    }
    public void addRoadUserToQueue(Road_User ru) { //Add road user to queue
        road_users_in_queue.add(ru); // Add the road user to the queue
        ru.inQueue = true; // Set inQeueu of the road user to true
        backOfQueue += ru.getLength() + gap; //Update position of the back of queue
        queueLengthInMeter += ru.getLength() + gap; //Update length of the queue
    }
    public void removeRoadUserFromQueue(Road_User ru) { // Remove road user from queue
```

```
        road_users_in_queue.remove(ru); //Removes the road user from the queue
        ru.inQueue = false; // Set inQeueu of the road user to false
        frontOfQueue += ru.getLength() + gap; //Update position of the front of queue
        queueLengthInMeter -= ru.getLength() + gap;//Update length of the queue
    }
    public Road_User getNextRoadUserInQueue(){// Returns the road user in front of the queue
        return road_users_in_queue.get(0);
    }
    public Road_User getRoadUserInQueue(int i){// Returns the i'th road user in the queue
        return road_users_in_queue.get(i);
    }
    public Road_User getLastRoadUserInQueue(){// Returns the road user in front of the queue
        return road_users_in_queue.get(road_users_in_queue.size() - 1);
    }
    public int queueLength() { // Returns the queue length
        return road_users_in_queue.size();
    }
    public void reset(){ // Reset
        road_users.clear(); // Empty array with road users
        road_users_in_queue.clear(); // Empty array with road users in queue
        queueLengthInMeter = 0; // Set queuelength in meter to zero
        frontOfQueue = 0; // Set position of front of the queue to zero
        backOfQueue = 0; // Set position of back of the queue to zero
        hasQueue = false; // Set hasQueue to false
        checked = false; // Set checked to false
    }
    public void resetQueue(){ // Sets the position of the back of the queue to the queuelength in meter and the position of the front of the queue to zero
        backOfQueue = queueLengthInMeter;
        frontOfQueue = 0;
    }
    public double getBackOfQueue(){ //Returns the position of the back of the queue
        return backOfQueue;
    }

    public void updateQueue(double q, double l){ // Update position of the front and back of the queue
        frontOfQueue = q;
        backOfQueue = q + l;
    }
```

```java
    void hasQueue() { // Set hasQueue to true
        hasQueue = true;
    }
    void checked(){ // Set checked to true
        checked = true;
    }
    public boolean getHasQueue() { // Return hasQueue
        return hasQueue;
    }
    public boolean getChecked(){ // Return checked
        return checked;
    }

}
```

The Event class.

```java
package swecobasicmodelextended;

public class Event {
    /* Events */
    /* Arrival and depature events */
    public static final int ARRIVAL_AT_QUEUE = 1;
    public static final int DEPARTURE_FROM_QUEUE = 2;
    /* Trafficlight events */
    public static final int TRAFFICLIGHT_TO_RED = 3;
    public static final int TRAFFICLIGHT_TO_AMBER = 4;
    public static final int TRAFFICLIGHT_TO_GREEN = 5;
    /* Stop simulation event */
    public static final int STOP_SIMULATION = 6;
    /* Register result event */
    public static final int REGISTER_RESULTS = 7;

    /* Event type */
    protected int type;
    /* Event time */
    protected double time;
    /* Signal group associated with event*/
    protected Signal_Group signal_group;
    /* Road user associated with event*/
    protected Road_User road_user;


    public Event(int type, double time, Signal_Group
        signal_group, Road_User road_user) {
        this.type = type;
        this.time = time;
        this.signal_group = signal_group;
        this.road_user = road_user;
    }
    public int getType() {//Returns type
        return type;
    }
    public double getTime() {//Returns time
        return time;
```

```java
    }
    public Signal_Group getSignalGroup(){//Returns signal group
        return signal_group;
    }
    public Road_User getRoadUser() {//Returns road user
        return road_user;
    }

}
```

The FES class.

```java
package swecobasicmodelextended;

import java.util.ArrayList;

// Future event set
public class FES {
    /* ArrayList of events */
    protected ArrayList<Event> events;

    public FES() {
        events = new ArrayList<Event>();
    }

    public void addEvent(Event newEvent) { //Add event to
        arrayList of events
    int insertIndex = 0;
    while (insertIndex < events.size()) {
      Event e = events.get(insertIndex);
      if (e.getTime() > newEvent.getTime()) break;
      insertIndex++;
    }
    events.add(insertIndex, newEvent);
  }
  public Event nextEvent() { //Returns first event (in time)
    return events.remove(0);
  }

}
```

The SimResults class.

```java
package swecobasicmodelextended;

public class SimResults {
    /* Performance measures per signal group */
    protected double[] sumS; // sum of the sojourn times
    protected double[] totalDelay; // sum of total delay
    protected double[] totalSquaredDelay; // sum of total
        squared delay
    protected int[] nrTotalStops; // sum of the total number of
         stops

```

```
    protected double totalCompare; // Total comparison of delay (between our model and vissim model)
    protected double totalPosCompare; // Total comparison of absolute delay (between our model and vissim model)

    protected double[] totDelayPTS; // Total comparison of delay (between our model and vissim model) per timestamp
    protected double[] totPosDelayPTS; // Total comparison of absolute delay (between our model and vissim model) per timestamp

    protected double totalComparePerRU; // Total comparison of delay (between our model and vissim model) per road user
    protected double totalAbsoluteComparePerRU; // Total comparison of absolute delay (between our model and vissim model) per road user

    protected double averageDepTime; // Average departure time
    protected double nrRu; // Number of road users

    public SimResults(int nrSignalGroups) {
        this.sumS = new double[nrSignalGroups+1] ;
        this.totalDelay = new double[nrSignalGroups+1];
        this.totalSquaredDelay = new double[nrSignalGroups+1];
        this.nrTotalStops = new int[nrSignalGroups+1];
        this.totalCompare = 0;
        this.totalPosCompare = 0;
        this.totDelayPTS = new double[26];
        this.totPosDelayPTS = new double[26];
        this.totalComparePerRU = 0;
        this.totalAbsoluteComparePerRU = 0;
        this.averageDepTime = 0;
        this.nrRu = 0;
    }
    void registerSojournTime(double sojournTime, Signal_Group sg) { //Registers the sojourn time
        sumS[sg.getIdentifier()] += sojournTime;
    }
    void registerDelay(double delay, Signal_Group sg){ // Registers the total delay and total squared delay
        totalDelay[sg.getIdentifier()] += delay;
        totalSquaredDelay[sg.getIdentifier()] += delay*delay;
    }
    void registerNrStops(int nrStops, Signal_Group sg) { // Registers the total number of stops
        nrTotalStops[sg.getIdentifier()] += nrStops;
    }
    public double[] getSojournTime(){//Returns sojourn time
        return sumS;
    }
    public double[] getTotalDelay(){//Returns total delay
        return totalDelay;
    }
    public double[] getTotalSquaredDelay(){//Returns total
```

```
        squared delay
        return totalSquaredDelay;
    }
    public int[] getTotalNrStops(){ //Returns the total number of stops
        return nrTotalStops;
    }
    void registerAbsoluteComparison(double tpc) {// Register total comparison of absolute delay (between our model and vissim model)
        totalPosCompare = tpc;
    }
    void registerComparison(double tc) {// Register total comparison of delay (between our model and vissim model)
        totalCompare = tc;
    }
    public double getTotalComparision(){ // Get total comparison of delay (between our model and vissim model)
        return totalCompare;
    }
    public double getTotalPosComparision(){ // Get total comparison of absolute delay (between our model and vissim model)
        return totalPosCompare;
    }
    void registerComparisonPerTimestamp(int f, double compareTotalDelay) { // Register total comparison of delay (between our model and vissim model) per timestamp
        totDelayPTS[f] = compareTotalDelay;
    }
    public double[] getTotDelayPTS(){ // Get total comparison of delay (between our model and vissim model) per timestamp
        return totDelayPTS;
    }
    void registerAbsoluteComparisonPerTimestamp(int f, double compareTotalPosDelay) { // Register total absolute comparison of delay (between our model and vissim model) per timestamp
        totPosDelayPTS[f] = compareTotalPosDelay;
    }
    public double[] getTotPosDelayPTS(){ // Get total absolute comparison of delay (between our model and vissim model) per timestamp
        return totPosDelayPTS;
    }

    void registerComparisonPerRU(double compareTotalFinalDelay, int nrRU) { // Register total comparison of delay (between our model and vissim model) per road user
        totalComparePerRU = compareTotalFinalDelay / nrRU;
    }
    public double getComparisonPerRU(){ // Get total comparison of delay (between our model and vissim model) per road
```

```
        user
        return totalComparePerRU;
    }
    void registerAbsoluteComparisonPerRU(double
        compareTotalFinalAbsoluteDelay, int nrRU) { // Register
        total comparison of absolute delay (between our model and
        vissim model) per road user
        totalAbsoluteComparePerRU =
            compareTotalFinalAbsoluteDelay / nrRU;
    }
    public double getAbsoluteComparisonPerRU(){ // Get total
        comparison of absolute delay (between our model and
        vissim model) per road user
        return totalAbsoluteComparePerRU;
    }

    void registerAverageDepTime(double depTime){ // Register
        average departure time
        averageDepTime = depTime;
    }
    public double getAverageDepTime (){ // Get average
        departure time
        return averageDepTime;
    }

    void registerNrRU(double nrRU){ // Register number of road
        users
        nrRu = nrRU;
    }
    public double getNrRu(){ // Get number of road users
        return nrRu;
    }

}
```

The Sim class.

```
package swecobasicmodelextended;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.util.*;
import java.util.Map.Entry;

/**
 *
 * @author Imke Vromans
 */
public class Sim {

    public SimResults simulate(double startTime, double maxTime
        , int nrTimestamps, Map<String, List<Integer>> junction,
         Map<Integer, List<Signal_Group>> sgMap, String module,
        String road_users, Map<Integer, Double> finalDelayMap,
```

```
        Map<Integer, Map<Double, Double>> delayMap) throws
        FileNotFoundException {
        PrintWriter pw = new PrintWriter(new File("Output.txt")); // Write output to file

        FES fes = new FES(); // Create new future event set
        int nrSignalGroups = sgMap.size(); // Total number of signal groups
        SimResults results = new SimResults(nrSignalGroups); // Create new simresults

        // Initialize traffic light colour and create trafficLight events
        importTrafficLightEvents(startTime, fes, sgMap, module);

        // Import all road users
        importRoadUsers(startTime, fes, junction, sgMap, road_users);

        double t = startTime; // Initialize time

        // The current output file from SUMO has for all signalgroups the vehicles and the delay and stops at timeStamps t=0 to t=25;
        // So to store the output in the same way we need to register the delay at every time stamp, for this we create an event REGISTER_RESULTS
        int f = 0;
        for (int i = 0; i < nrTimestamps; i++) {
            Event RegisterResults = new Event(Event.REGISTER_RESULTS, startTime + i, null, null);
            fes.addEvent(RegisterResults);
        }

        // Schedule the stop simulation event
        // (this event prevents the simulation from continuing beyond the desired time horizon and prevents the program from ending due to an empty FES)
        Event StopSimulation = new Event(Event.STOP_SIMULATION, maxTime, null, null);
        fes.addEvent(StopSimulation);

        // Start simulation loop
        while (t < maxTime) {
            Event e = fes.nextEvent();// Get the next event
            t = e.getTime();// Update the time

            switch (e.getType()) {
                case Event.ARRIVAL_AT_QUEUE: {
                    Signal_Group sg = e.getSignalGroup();// Get signal group associated with event
                    Road_User ru = e.getRoadUser(); // Get the road user
```

```
                if (((sg.trafficLightColour() == Signal_Group.GREEN || (sg.trafficLightColour() == Signal_Group.AMBER)) && sg.queueLength() == 0)) {// If the traffic light is green or amber and there are no road users in the queue: Directly drive through (do not add road user to queue)
                    ru.leaveSystem(t); //Registers departure time
                } else if (sg.trafficLightColour() == Signal_Group.RED && sg.queueLength() == 0) { //If the traffic light is red and there isnt a queue
                    ru.updatePosition(0); // Update current position of road user
                    ru.nrInLine(1); // Update the position of the road user in nr of vehicles in line
                    sg.addRoadUserToQueue(ru);// Add road user to queue
                    ru.addStop(); // Add stop
                } else { // If the traffic light is red and there is a queue
                    ru.updatePosition(sg.getBackOfQueue()); // Update current position of road user
                    Road_User priorRU = sg.getLastRoadUserInQueue(); // Get last road user in queue
                    ru.nrInLine(priorRU.nrInLine + 1); // Update the position of the road user in nr of vehicles in line
                    sg.addRoadUserToQueue(ru); // Add road user to queue
                    ru.addStop(); //Add stop
                }
                break;
            }

            case Event.DEPARTURE_FROM_QUEUE: {
                Signal_Group sg = e.getSignalGroup();   // Get signal group associated with event
                if ((sg.trafficLightColour() == Signal_Group.GREEN) || (sg.trafficLightColour() == Signal_Group.AMBER)) { //If the traffic light is green or amber:
                    Road_User ru = e.getRoadUser(); // Get the road user
                    sg.removeRoadUserFromQueue(ru);// Remove road user from the queue
                    ru.leaveSystem(t); //Registers departure time
```

```
                    if (sg.queueLength() > 0) {
                        //If there are still road users in
                            the queue:
                         Road_User nextRu = sg.
                             getNextRoadUserInQueue(); //
                             Get next road user in the queue
                        //Schedule departure for next road
                            user
                        Event newDepartureFromQueue = new
                            Event(Event.DEPARTURE_FROM_QUEUE
                            , t + nextRu.getServiceTime(),
                            sg, nextRu);
                        fes.addEvent(newDepartureFromQueue)
                            ;
                    } else {
                        sg.resetQueue(); //Else if
                            queueLength = 0, then the front
                            and back of the queue get set to
                             0 again
                    }
                } else { // If the traffic light is red
                    //Add extra stop to all road users in
                        queue, since the traffic light will
                        have turned red before they could
                        pass the intersection
                    int position = 0;
                    for (int i = 0; i < sg.queueLength(); i
                        ++) {
                        Road_User ru_i_q = sg.
                            getRoadUserInQueue(i); // Get i'
                            th road user in the queue
                        ru_i_q.updatePosition(position); //
                             Update position of road user
                        ru_i_q.nrInLine(i + 1); // Update
                            the position of the road user in
                             nr of vehicles in line
                        ru_i_q.addStop(); // Add stop
                        position += ru_i_q.getLength() +
                            Signal_Group.gap; // Update
                            value of position to be used for
                             road user in queue
                    }
                }
                break;
            }

            case Event.TRAFFICLIGHT_TO_RED: {
                Signal_Group sg = e.getSignalGroup(); //
                    Get signal group associated with event
                // Change traffic light to red for signal
                    group associated with event
                sg.changeTrafficLightColour(Signal_Group.
                    RED);
                sg.resetQueue(); // Reset the position of
```

```java
                        the front and back of queue
                    break;
                }

                case Event.TRAFFICLIGHT_TO_AMBER: {
                    Signal_Group sg = e.getSignalGroup(); // Get signal group associated with event
                    // Change traffic light to amber for signal group associated with event
                    sg.changeTrafficLightColour(Signal_Group.AMBER);
                    break;
                }

                case Event.TRAFFICLIGHT_TO_GREEN: {
                    Signal_Group sg = e.getSignalGroup(); // Get signal group associated with event
                    //Change traffic light to red for signal group associated with event
                    sg.changeTrafficLightColour(Signal_Group.GREEN);
                    if (sg.queueLength() > 0) { // If there are road users waiting in the queue in front of the trafficlight
                        Road_User nextRu = sg.getNextRoadUserInQueue(); // Get the first road user in the queue (the road user in front of the queue)
                        //Schedule a departure from queue event for this road user
                        Event newDepartureFromQueue = new Event(Event.DEPARTURE_FROM_QUEUE, t + nextRu.getServiceTime(), sg, nextRu);
                        fes.addEvent(newDepartureFromQueue);
                    }
                    break;
                }
                case Event.REGISTER_RESULTS:
                    double compareTotalDelay = 0;
                    double compareTotalAbsoluteDelay = 0;

                    pw.println("Timestamp " + (t-startTime));
                    for (Entry<Integer, List<Signal_Group>> entry : sgMap.entrySet()) { // For every signal group
                        List<Signal_Group> sgList = entry.getValue();
                        pw.println("Signal group " + entry.getKey());
                        for (Signal_Group sg : sgList) {
                            for (int k = 0; k < sg.nrRoadUsers(); k++) { // For every road user in the signal group
```

```
                    Road_User ru = sg.getRoadUser(k
                        ); //Get the road user
                    double delay;
                    double sojournTime;
                    double minTravelTime;
                    if (ru.hasLeftSystem()) { // If
                         the road user has left the
                        system
                        sojournTime = ru.
                            getDepartureTime() - ru.
                            getArrivalTime(); //
                            Calculate sojourn time
                        minTravelTime = sg.
                            getLength() / ru.
                            getDesSpeed(); //
                            Calculate minimum travel
                             time based on the total
                             length of the lane
                        delay = Math.max(0,
                            sojournTime -
                            minTravelTime); //
                            Calculate delay
                    } else { // If the road user is
                         still in the system
                        //Calculate delay using
                            time t
                        if (ru.inQueue()) { // If
                            the road user is in the
                            queue
                            sojournTime = t - ru.
                                getArrivalTime(); //
                                 Calculate sojourn
                                time
                            minTravelTime = (sg.
                                getLength() - ru.
                                getPosition()) / ru.
                                getDesSpeed();///
                                Calculate minimum
                                travel time
                            delay = Math.max(0,
                                sojournTime -
                                minTravelTime); //
                                Calculate delay
                        } else { // If the road
                            user is not in the queue
                            // Separate into two
                                sections, from the
                                arrival time to the
                                start time and from
                                the start time to
                                the current time
                            // The delay from the
                                arrival time to the
                                start time is known
```

```
exactly
double sojournTimeATS = startTime - ru.getArrivalTime(); // Calculate sojourn time from arrival to start
double distanceFromATS = sg.getLength() - ru.getPosition(); // Calculate distance from arrival to start
double minTravelTimeATS = distanceFromATS/ru.getDesSpeed(); // Calculate minimum travel time from arrival to start
double delayATS = sojournTimeATS - minTravelTimeATS; // Calculate delay from arrival to start
// Calculate delay from start time to current time
double sojournTimeSTC = t - startTime; // Calculate sojourn time start to t
double distanceFromSTC = sojournTimeSTC * ru.getSpeed(); // Estimate the distance the vehicle has travelled from start to t
double minTravelTimeSTC = distanceFromSTC / ru.getDesSpeed(); // Calculate minimum travel time from start to t
double delaySTC = sojournTimeSTC - minTravelTimeSTC; // Calculate delay from start to t
//Calculate total delay
delay = Math.max(0, delayATS + delaySTC);
```

```java
                                    }
                                }
                                double compareDelay = delay - delayMap.get(ru.getId()).get(t); // Compare calculated delay to vissim delay
                                double compareAbsoluteDelay = Math.abs(compareDelay); // Compare calculated delay to vissim delay in absolute difference
                                compareTotalDelay = compareTotalDelay + compareDelay; // Sum the difference between calculated delay and vissim delay
                                compareTotalAbsoluteDelay = compareTotalAbsoluteDelay + compareAbsoluteDelay; // Sum the absolute difference between calculated delay and vissim delay
                                //Write delay and nrStops to output file
                                pw.println("Road user " + ru.getId() + " Delay " + delay + " NrStops " + ru.getNrStops());
                            }
                        }

                    }
                    // Register the total differences between calculated delay and vissim delay
                    results.registerComparisonPerTimestamp(f, compareTotalDelay);
                    results.registerAbsoluteComparisonPerTimestamp(f, compareTotalAbsoluteDelay);
                    f++;
                    pw.println();
                    default:

                    break;
            }

        }
        pw.close();

        // After simulation calculate the final delay (after all vehicles have left the system)
        double compareTotalFinalDelay = 0; // Compare difference between calculated delay and vissim delay
```

```java
         after all vehicle have left the system
        double compareTotalFinalAbsoluteDelay = 0; // Compare absolute difference between calculated delay and vissim delay after all vehicle have left the system

        int nrRU = 0; // Number of road users
        double depTime = 0; // Departure time

        for (Entry<Integer, List<Signal_Group>> entry : sgMap.entrySet()) { // For every signal group
            List<Signal_Group> sgList = entry.getValue();
            for (Signal_Group sg : sgList) {
                int nrRUinSG = sg.nrRoadUsers();
                for (int k = 0; k < sg.nrRoadUsers(); k++) { // For every road user in the signal group
                    Road_User ru = sg.getRoadUser(k); // get the road user
                    double sojournTime = ru.getDepartureTime() - ru.getArrivalTime(); // Calculate sojourn time
                    double minTT = sg.getLength() / ru.getDesSpeed(); // Calculate the minimum travel time
                    double delay = Math.max(0, sojournTime - minTT); // Calculate the delay

                    // Register performance measures
                    //results.registerSojournTime(sojournTime, sg); // register sojourntime
                    //results.registerDelay(delay, sg); // register delay
                    //results.registerNrStops(ru.getNrStops(), sg); // register number of stops

                    // Register difference with Vissim
                    double compareDelay = delay - finalDelayMap.get(ru.getId()); // Compare difference between calculated delay and vissim delay
                    double compareAbsoluteDelay = Math.abs(delay - finalDelayMap.get(ru.getId())); // Compare absolute difference between calculated delay and vissim delay
                    compareTotalFinalDelay = compareTotalFinalDelay + compareDelay; // Sum the difference between calculated delay and vissim delay
                    compareTotalFinalAbsoluteDelay = compareTotalFinalAbsoluteDelay + compareAbsoluteDelay; // Sum the absolute difference between calculated delay and vissim delay

                    depTime += ru.getDepartureTime()-startTime;
```

```
                        // Calculate departure time

                }
                nrRU = nrRU + nrRUinSG; // Calculate total
                    number of road users
            }
        }
        // Register results
        results.registerAverageDepTime(depTime/nrRU);
        results.registerNrRU(nrRU);

        results.registerComparisonPerRU(compareTotalFinalDelay,
             nrRU);
        results.registerAbsoluteComparisonPerRU(
            compareTotalFinalAbsoluteDelay, nrRU);

        results.registerComparison(compareTotalFinalDelay);
        results.registerAbsoluteComparison(
            compareTotalFinalAbsoluteDelay);

        return results;
    }

    public void importRoadUsers(double startTime, FES fes, Map<
        String, List<Integer>> junction, Map<Integer, List<
        Signal_Group>> sgMap, String road_users) {
        // NOTE: the road users need to be in order from
            closest to trafficlight to furtherst from traffic
            light
        // (This is necessary since the queue length needs to
            be known before we can schedule arrival at queue
            events )
        int ru_id;
        //Read road users from file
        try (Scanner sc = new Scanner(new File(road_users))) {
            sc.useLocale(Locale.US);
            while (sc.hasNextLine()) {
                if (sc.hasNextInt()) {
                    ru_id = sc.nextInt();
                } else {
                    break;
                }
                double arrivalTime = sc.nextDouble();
                double length = sc.nextDouble();
                String type = sc.next();
                String arrivalPlace = sc.next();
                double position = sc.nextDouble();
                String a = sc.next();
                String[] ar = a.split(";");
                double[] prob = new double[ar.length];
                for (int i = 0; i < ar.length; i++) {
                    prob[i] = Double.parseDouble(ar[i]);
                }
                double speed = sc.nextDouble();
```

```java
                double desSpeed = sc.nextDouble();
                int nrInLine = sc.nextInt();

                // Create road user
                Road_User ru = new Road_User(ru_id, arrivalTime, length, type, arrivalPlace, position, prob, speed, desSpeed, nrInLine);

                //Determine signal group based on arrivalPlace and probabilities
                Signal_Group sg = determineSignalGroup(junction, sgMap, arrivalPlace, prob);
                // Add road user to the signal group it belongs to
                sg.addRoadUser(ru);

                if (speed < 2.777) { //If any of the vehicles have speed less than 10km/h, we assume there is a queue
                    sg.hasQueue();
                }
            }
            sc.close();
        } catch (FileNotFoundException e) {
            System.out.println("The road_users file " + road_users + " could not be found");
        }

        for (Entry<Integer, List<Signal_Group>> entry : sgMap.entrySet()) { //For every signal group
            List<Signal_Group> sgList = entry.getValue();
            for (Signal_Group sg : sgList) {
                if (sg.getHasQueue()) { // If the signal group has a queue
                    Road_User ru = sg.getRoadUser(0); // Get the first road user
                    sg.updateQueue(ru.getPosition(), ru.getLength());// Update position of the queue
                    sg.addRoadUserToQueue(ru); //Add user to the queue
                    ru.addStop(); //Add stop
                    if (ru.getSpeed() < 2.777){
                        sg.checked();
                    }
                    // If the signal group is green then a schedule departure from queue event for this road user
                    if (sg.trafficLightColour() == Signal_Group.GREEN) {
                        Event newDepartureFromQueue = new Event(Event.DEPARTURE_FROM_QUEUE, startTime, sg, ru);
                        fes.addEvent(newDepartureFromQueue);
```

```
            }
            // For all other road users
            double lengthOfRu = ru.getLength() +
                Signal_Group.gap;
            for (int i = 1; i < sg.nrRoadUsers(); i++)
                {
                Road_User nextRu = sg.getRoadUser(i);
                    // Get the road user
                lengthOfRu += nextRu.getLength() +
                    Signal_Group.gap;
                if (!sg.getChecked() || nextRu.getSpeed
                    () < 1.388) { //Put them in queue if
                     they are in the queue according to
                    the speedboundaries
                   if (nextRu.getSpeed() < 2.777){
                        sg.checked();
                    }
                    sg.addRoadUserToQueue(nextRu); //
                        Add road user to the queue
                    nextRu.nrInLine(ru.getNrInLine() +
                        i); // Update nrInLine
                    nextRu.addStop(); //Add stop
                } else { //Else schedule arrivals
                    double arrivalAtQueueTime = (nextRu
                        .getPosition() - lengthOfRu) /
                        sg.maxSpeed(); // Calculate
                        arrival at queue time
                    //Schedule arrival event for this
                        road user
                    Event newArrivalAtQueue = new Event
                        (Event.ARRIVAL_AT_QUEUE,
                        startTime + arrivalAtQueueTime,
                        sg, nextRu);
                    fes.addEvent(newArrivalAtQueue);
                }

            }
        } else { // If there isnt a queue schedule
           arrival events for all road users in the sg
            double lengthOfRu = 0;
            for (int i = 0; i < sg.nrRoadUsers(); i++)
                {
                Road_User ru = sg.getRoadUser(i); //
                    Get the road user
                lengthOfRu = ru.getLength() + sg.gap;
                double arrivalAtQueueTime = (ru.
                    getPosition() - lengthOfRu) / sg.
                    maxSpeed(); // Calculate arrival at
                    queue time
                //Schedule arrival event for this road
                    user
                Event newArrivalAtQueue = new Event(
                    Event.ARRIVAL_AT_QUEUE, startTime +
                    arrivalAtQueueTime, sg, ru);
```

```
                        fes.addEvent(newArrivalAtQueue);
                    }
                }
            }
        }
    }

    public Signal_Group determineSignalGroup(Map<String, List<Integer>> junction, Map<Integer, List<Signal_Group>> sgMap, String arrivalPlace, double[] prob) {
        Random rng = new Random(); //Random number generator
        Signal_Group sg = null; //Initialize sg with dummy

        double U = rng.nextDouble();
        for (int i = 0; i < prob.length; i++) {
            if (U < prob[i]) {  // If U is less than the probability then go to that signalgroup
                int id = junction.get(arrivalPlace).get(i); // Get the identity number of the signal group
                List<Signal_Group> sgList = sgMap.get(id); // Get all signal groups with this identity number
                sg = sgList.get(0); // Set sg to the first signal group with this identity number
                int ql = sg.nrRoadUsers(); // Get the queue length of this signal group sg
                for (int j = 1; j < sgList.size(); j++) { // For all signal groups with the same identity number
                    Signal_Group newSG = sgList.get(j); // Compare the queue length
                    int newQL = newSG.nrRoadUsers();
                    if (newQL < ql) { //If the queuelength is smaller then set sg to this signal group
                        sg = newSG;
                        ql = newQL;
                    }

                }
                break;
            }
        }
        return sg;
    }

    public void importTrafficLightEvents(double startTime, FES fes, Map<Integer, List<Signal_Group>> sgMap, String module) {
        double simSec;

        //Import traffic light events
        try (Scanner sc = new Scanner(new File(module))) {
            sc.useLocale(Locale.US);
            while (sc.hasNextLine()) {
```

```
                if (sc.hasNextDouble()) {
                    simSec = sc.nextDouble();
                } else {
                    break;
                }
                int sg_id = sc.nextInt();
                String colour = sc.next();
                if (simSec <= startTime) { //Initialize trafficLights
                    List<Signal_Group> sgList = sgMap.get(sg_id); // Get all signal groups with the same identity number
                    for (Signal_Group sg : sgList) { // Initialize traffic light colours for all signal groups
                        switch (colour) {
                            case "green":
                                sg.changeTrafficLightColour(Signal_Group.GREEN);
                                break;
                            case "amber":
                                sg.changeTrafficLightColour(Signal_Group.AMBER);
                                break;
                            case "red":
                                sg.changeTrafficLightColour(Signal_Group.RED);
                                break;
                        }
                    }
                } else { //Schedule trafficLight events
                    List<Signal_Group> sgList = sgMap.get(sg_id); // Get all signal groups with the same identity number
                    for (Signal_Group sg : sgList) { //Schedule traffic light events for all these signal groups
                        switch (colour) {
                            case "green":
                                Event newTrafficEventGreen = new Event(Event.TRAFFICLIGHT_TO_GREEN, simSec, sg, null);
                                fes.addEvent(newTrafficEventGreen);
                                break;
                            case "amber":
                                Event newTrafficEventAmber = new Event(Event.TRAFFICLIGHT_TO_AMBER, simSec, sg, null);
                                fes.addEvent(newTrafficEventAmber);
                                break;
```

```java
                            case "red":
                                Event newTrafficEventRed = new Event(Event.TRAFFICLIGHT_TO_RED, simSec, sg, null);
                                fes.addEvent(newTrafficEventRed);
                                break;
                        }
                    }
                }
            }
            sc.close();
        } catch (FileNotFoundException e) {
            System.out.println("The module file " + module + " could not be found");
        }
    }

    public static void main(String[] args) throws FileNotFoundException {
        //long start = System.nanoTime();

        double maxTime = 600; // Maximum time of simulation
        int nrTimestamps = 26; // Number of time steps
        int nrSnapshots = 100; // Number of snapshots

        //Read the junction structure from input file
        String junction_structure = "Junction_Structure.txt";
        Map<String, List<Integer>> junction = new HashMap<>(); //Maps the arrival place (string) to a list of the identifiers of possible signal group (which can be reached from this arrival place)
        Map<Integer, List<Signal_Group>> sgMap = new HashMap<>(); //Maps the identifier of the signal group to a list of signal (sub)groups

        // Create junction
        try (Scanner sc = new Scanner(new File(junction_structure))) {
            sc.useLocale(Locale.US);
            while (sc.hasNextLine()) {
                String arrivalPlace = sc.next();
                List<Integer> list = new ArrayList<>(); //List with identifiers available from arrivalPlace
                while (sc.hasNextInt()) {
                    List<Signal_Group> listSG = new ArrayList<>(); //List of signal (sub)groups
                    //Read from file: int identifier, double travelTime, double maxSpeed, int trafficLightColour and nrLanes
                    int id = sc.nextInt();
                    double length = sc.nextDouble();
                    double maxSpeed = sc.nextDouble();
```

```
                    int nrLanes = sc.nextInt();
                    // Create signal (sub)groups
                    for (int j = 0; j < nrLanes; j++) { // Create a signal (sub)group for each lane
                        Signal_Group sg = new Signal_Group(id, length, maxSpeed); //Create signal (sub)group
                        listSG.add(sg); //Add these to the list of signal (sub)groups
                    }
                    list.add(id); // Add id to the list of identifiers available from arrivalPlace
                    sgMap.put(id, listSG); //Map id to the signal (sub)groups
                }
                junction.put(arrivalPlace, list); //Map arrivalPlace to the list of identifiers available from arrivalPlace
            }
            sc.close();
        } catch (FileNotFoundException e) {
            System.out.println("The junction file " + junction_structure + " could not be found");
        }

        // Create arrays to store difference between calculated delay and Vissim delay
        double[] compareDelay = new double[nrSnapshots];
        double[] compareAbsoluteDelay = new double[nrSnapshots];
        double[][] compareDelayPTS = new double[nrSnapshots][nrTimestamps];
        double[] compareTotalDelayPTS = new double[nrTimestamps];
        double[][] compareAbsoluteDelayPTS = new double[nrSnapshots][nrTimestamps];
        double[] compareAbsoluteTotalDelayPTS = new double[nrTimestamps];

        double[] compareTotalDelayPTSperRu = new double[nrTimestamps];
        double[] compareAbsoluteTotalDelayPTSperRu = new double[nrTimestamps];

        double[] comparePerRU = new double[nrSnapshots];
        double[] compareAbsolutePerRU = new double[nrSnapshots];

        double[] averageDepTime = new double[nrSnapshots];
        double[] nrRu = new double[nrSnapshots];

        // Run for every snapshot
        for (int z = 1; z < (nrSnapshots + 1); z++) {
            String format = String.format("%03d", z);
```

```
        String simSec = "simSec" + format + ".txt";
        String module = "Module" + format + ".txt";
        String road_users = "Roadusers" + format + ".txt";

        String finalDelay = "FinalDelay" + format + ".txt";
        String delays = "Delay" + format + ".txt";

        // Read simSec file
        double startTime;
        try (Scanner sc = new Scanner(new File(simSec))) {
            sc.useLocale(Locale.US);
            startTime = sc.nextDouble();
        }

        // Read FinalDelay file
        // Create map to compare Vissim final delay to
            simulated final delay (after all vehicles have
            left the system)
        Map<Integer, Double> finalDelayMap = new HashMap
            <>(); // Maps the identifier of the road user to
             the delay
        try (Scanner sc = new Scanner(new File(finalDelay))
            ) {
            sc.useLocale(Locale.US);
            int ru_id;
            while (sc.hasNextLine()) {
                if (sc.hasNextInt()) {
                    ru_id = sc.nextInt();
                } else {
                    break;
                }
                double delayVis = sc.nextDouble();
                finalDelayMap.put(ru_id, delayVis);
            }
            sc.close();
        } catch (FileNotFoundException e) {
            System.out.println("The file with final delay "
                 + finalDelay + " could not be found");
        }

        // Read Delay file
        // Create map to compare Vissim delay to simulated
            delay at every timestamp
        Map<Integer, Map<Double, Double>> delaysMap = new
            HashMap<>(); //Maps the identifier of the road
            user to the delay
        try (Scanner sc = new Scanner(new File(delays))) {
            int ru_id;
            sc.useLocale(Locale.US);
            while (sc.hasNextLine()) {
                Map<Double, Double> secondMap = new HashMap
                    <>(); // Maps time to the vissim delay
                if (sc.hasNext()) {
```

```
                    sc.next();
                    ru_id = sc.nextInt();
                    while (sc.hasNextDouble()) {
                        double t = sc.nextDouble();
                        double delay = sc.nextDouble();
                        secondMap.put(t, delay);
                    }
                } else {
                    break;
                }
                delaysMap.put(ru_id, secondMap);
            }
            sc.close();

        } catch (FileNotFoundException e) {
            System.out.println("The file with delays " + delays + " could not be found");
        }

        //We can calculate the expected delay over multiple runs, if the signal groups are assigned on a stochastic basis
        //int nrRuns = 1; // Number of runs
        //int nrSignalGroups = sgMap.size(); //number of signal groups
        //double[] sumDelay = new double[nrSignalGroups]; // Sum of delays over all runs
        //double[] expDelay = new double[nrSignalGroups]; // Expected delay

        //Run simulation
        Sim sim = new Sim();
        //for (int j = 0; j < nrRuns; j++) {
            for (Entry<Integer, List<Signal_Group>> entry : sgMap.entrySet()) {
                for (Signal_Group sg : entry.getValue()) {
                    sg.reset(); // Reset all values before simulation
                }
            }
            SimResults results = sim.simulate(startTime, maxTime, nrTimestamps, junction, sgMap, module, road_users, finalDelayMap, delaysMap);// Simulation

            //Difference in results between java model and Vissim
            compareDelay[(z - 1)] = results.getTotalComparision(); // total difference in delay between java model and vissim (for every snapshot)
            compareAbsoluteDelay[(z - 1)] = results.getTotalPosComparision(); // total absolute difference in delay between java model and
```

```
                    vissim (for every snapshot)
                compareDelayPTS[(z - 1)] = results.getTotDelayPTS(); // total difference in delay per timestamp between java model and vissim (for every snapshot)
                compareAbsoluteDelayPTS[(z - 1)] = results.getTotPosDelayPTS(); // total absolute difference in delay per timestamp between java model and vissim (for every snapshot)

                comparePerRU[(z-1)] = results.getComparisonPerRU(); // total difference in delay between java model and vissim (for every snapshot) per road user
                compareAbsolutePerRU[(z-1)] = results.getAbsoluteComparisonPerRU(); // total absolute difference in delay between java model and vissim (for every snapshot) per road user

                averageDepTime[(z-1)] = results.getAverageDepTime(); // average departure time
                nrRu[(z-1)] = results.getNrRu(); // number of road users
                //double[] delay = results.getTotalDelay();
                //for (int k = 0; k < nrSignalGroups; k++) {
                //  sumDelay[k] = sumDelay[k] + delay[k]; // Add delay from run to total delay over all runs
                //}
            //}

            //for (int j = 0; j < nrSignalGroups; j++) { // Calculate the expected delay
            //  expDelay[j] = sumDelay[j] / nrRuns;
            //}
        }

        // Sum the (absolute) delay per timestamp over all snapshots
        for (int h = 0; h < nrTimestamps; h++) {
            for (int g = 0; g < nrSnapshots; g++) {
                compareTotalDelayPTS[h] = compareTotalDelayPTS[h] + compareDelayPTS[g][h];
                compareAbsoluteTotalDelayPTS[h] = compareAbsoluteTotalDelayPTS[h] + compareAbsoluteDelayPTS[g][h];
            }
            compareTotalDelayPTSperRu[h] = compareTotalDelayPTS[h]/Arrays.stream(nrRu).sum();
            compareAbsoluteTotalDelayPTSperRu[h] = compareAbsoluteTotalDelayPTS[h]/Arrays.stream(nrRu).sum();
```

```
        }
        //long endTime = System.nanoTime();

        //long duration = (endTime - start)/1000000;  //divide by 1000000 to get milliseconds.


        System.out.println("RESULTS");
        System.out.println();
        System.out.println("Average departure time");
        System.out.println(Arrays.toString(averageDepTime));
        System.out.println("Average departure time over all snapshot: " + Arrays.stream(averageDepTime).sum()/100);
        System.out.println();
        System.out.println("Total number of road users ");
        System.out.println(Arrays.stream(nrRu).sum());
        System.out.println();
        System.out.println("Difference in delay between model and Vissim per snapshot");
        System.out.println(Arrays.toString(compareDelay));
        System.out.println();
        System.out.println("Difference in absolute delay between model and Vissim per snapshot");
        System.out.println(Arrays.toString(compareAbsoluteDelay));
        System.out.println();
        System.out.println("Total difference in delay between model and Vissim (summed over all snapshots)");
        System.out.println(Arrays.stream(compareDelay).sum());
        System.out.println();
        System.out.println("Total absolute difference in delay between model and Vissim (summed over all snapshots)");
        System.out.println(Arrays.stream(compareAbsoluteDelay).sum());
        System.out.println();
        System.out.println("Total difference in delay between model and Vissim per timestamp (summed over all snapshots)");
        System.out.println(Arrays.toString(compareTotalDelayPTS));
        System.out.println();
        System.out.println("Total absolute difference in delay between model and Vissim per timestamp (summed over all snapshots)");
        System.out.println(Arrays.toString(compareAbsoluteTotalDelayPTS));
        System.out.println();
        System.out.println("Difference in delay between model and Vissim per snapshot per road user");
        System.out.println(Arrays.toString(comparePerRU));
        System.out.println("Total difference in delay between model and Vissim (summed over all snapshots) per
```

```
            road user");
        System.out.println(Arrays.stream(comparePerRU).sum()
            /100);
        System.out.println("Difference in absolute delay
            between model and Vissim per snapshot per road user
            ");
        System.out.println(Arrays.toString(compareAbsolutePerRU
            ));
        System.out.println("Total absolute difference in delay
            between model and Vissim (summed over all snapshots)
             per road user ");
        System.out.println(Arrays.stream(compareAbsolutePerRU).
            sum()/100);
        System.out.println("Total difference in delay between
            model and Vissim per timestamp (summed over all
            snapshots) average per road user");
        System.out.println(Arrays.toString(
            compareTotalDelayPTSperRu));
        System.out.println();
        System.out.println("Total absolute difference in delay
            between model and Vissim per timestamp (summed over
            all snapshots) average per road user");
        System.out.println(Arrays.toString(
            compareAbsoluteTotalDelayPTSperRu));
    }
}
```

A.1.2 Example input files

Junction_Structure: Txt file containing the structure of the junction. The txt file has, for every arrival place, the name of an arrival place and below it the id, the length of the lane, the maximum speed and the number of lanes.

```
Noord
1 107.84 13.88 1
2 107.84 13.88 1
3 107.84 13.88 1

Oost
4 48.29 13.88 1
5 48.29 13.88 1
6 48.29 13.88 1

Zuid
7 109.53 13.88 1
8 109.53 13.88 1
9 109.53 13.88 1

West
10 44.24 13.88 1
11 44.24 13.88 1
12 44.24 13.88 1
```

Roadusers: Txt file containing the snapshot. The txt file contains every road user in the snapshot, with for every road user its id number, arrival time, length, type, arrival place, position, probability array, speed and number in line.

```
78 263.92 4.76 PASSENGER Zuid 0.824 0.0;0.0;1.0 0 15.43333 1
80 272.03 4.76 PASSENGER Zuid 7.944 0.0;0.0;1.0 0.8722222 13.52778 2
82 264.75 4.61 PASSENGER Oost 0.895 0.0;1.0;1.0 0 15.43333 1
83 276.38 4.01 PASSENGER Noord 9.919 0.0;1.0;1.0 16.64444 20.53333 3
88 279.32 4.61 PASSENGER Oost 27.265 1.0;1.0;1.0 10.27778 13.80278 6
```

simSec: Txt file containing the starting time of the simulation.

```
282
```

Module: Txt file containing the scheme, with on every line: the simulation second, the signal group and traffic light colour.

```
280.0 1 green
280.0 2 green
280.0 3 green
280.0 10 red
280.0 11 red
280.0 12 red
270.0 7 red
270.0 8 red
270.0 9 red
260.0 4 red
260.0 5 red
260.0 6 red
287.0 1 amber
287.0 2 amber
287.0 3 amber
290.0 1 red
290.0 2 red
290.0 3 red
290.0 4 green
...
597.0 11 amber
597.0 12 amber
600.0 1 green
600.0 2 green
600.0 3 green
600.0 10 red
600.0 11 red
600.0 12 red
```

Delay: Txt file containing the Vissim delay for every road user in the snapshot per time stamp.

```
newVehicle
78
282 11.0379
```

```
283 12.0379
284 13.0379
285 14.0379
286 15.0379
287 16.0379
288 17.0379
289 18.0379
290 19.0379
291 20.0379
292 21.0379
293 22.0379
294 23.0379
295 24.0379
296 25.0379
297 26.0379
298 27.0379
299 28.0379
300 29.0379
301 29.97376
302 29.97376
303 29.97376
304 29.97376
305 29.97376
306 29.97376
307 29.97376
...
newVehicle
88
282 1.156828
283 1.422194
284 1.846223
285 2.491222
286 3.320966
287 4.266629
288 5.255762
289 6.255762
290 7.255762
291 8.185486
292 8.185486
293 8.185486
294 8.185486
295 8.185486
296 8.185486
297 8.185486
298 8.185486
299 8.185486
300 8.185486
301 8.185486
302 8.185486
303 8.185486
304 8.185486
305 8.185486
306 8.185486
307 8.185486
```

FinalDelay: Txt file containing the Vissim delay for every road user after they have left the system.

```
78 30.0
80 23.4
82 23.1
83 0.9
88 8.3
```

A.1.3 Example output file

Ouput: Txt file containing the results. For every timestamp the delay and number of stop per road user is given by signal group.

```
Timestamp 0.0
Signal group 1
Signal group 2
Road user 83 Delay 1.1998207110098607 NrStops 0
Signal group 3
Road user 75 Delay 21.885900553279004 NrStops 1
Signal group 4
Road user 88 Delay 1.2292556571937028 NrStops 0
Signal group 5
Signal group 6
Signal group 7
Signal group 8
Road user 71 Delay 39.34481556211208 NrStops 1
Signal group 9
Road user 78 Delay 11.063022225274803 NrStops 1
Road user 80 Delay 2.8096602990291046 NrStops 1
Signal group 10
Road user 81 Delay 23.604481795508626 NrStops 1
Road user 87 Delay 4.8452161690455675 NrStops 1
Signal group 11
Road user 86 Delay 5.989654575991809 NrStops 1
Signal group 12

Timestamp 1.0
Signal group 1
Signal group 2
Road user 83 Delay 1.3680510954628442 NrStops 1
Signal group 3
Road user 75 Delay 22.885900553279004 NrStops 1
Signal group 4
Road user 88 Delay 1.4810165343503336 NrStops 0
Signal group 5
Signal group 6
Signal group 7
Signal group 8
Road user 71 Delay 39.34481556211208 NrStops 1
Signal group 9
Road user 78 Delay 11.063022225274803 NrStops 1
Road user 80 Delay 3.8096602990291046 NrStops 1
Signal group 10
```

```
Road user 81 Delay 24.604481795508626 NrStops 1
Road user 87 Delay 5.8452161690455675 NrStops 1
Signal group 11
Road user 86 Delay 6.989654575991809 NrStops 1
Signal group 12
...
Timestamp 25.0
Signal group 1
Signal group 2
Road user 83 Delay 25.368051095462846 NrStops 1
Signal group 3
Road user 75 Delay 46.885900553279 NrStops 1
Signal group 4
Road user 88 Delay 24.43142942218887 NrStops 1
Signal group 5
Signal group 6
Signal group 7
Signal group 8
Road user 71 Delay 39.34481556211208 NrStops 1
Signal group 9
Road user 78 Delay 11.063022225274803 NrStops 1
Road user 80 Delay 3.8889878187551794 NrStops 1
Signal group 10
Road user 81 Delay 42.63274471873668 NrStops 1
Road user 87 Delay 26.2020291649789 NrStops 1
Signal group 11
Road user 86 Delay 25.03197623965171 NrStops 1
Signal group 12
```

A.2 Polling model with switching customers

A.2.1 Simulation file

```
(*Exact Analysis of Polling System with switching customers*)
(*LSTs for branching-type service disciplines*)
(*Author : Imke Vromans, June 2021 based on Marko Boon,
    November 2013*)
(*This Mathematica notebook computes performance measures for
    polling systems with switching customers at polling
    instances . It implements LSTs, which work for branching-
    type service disciplines. In particular, gated and
    exhaustive are implemented.*)
(*Numerical Input*)
(*Provide the moments of the input variables. *)
(*Service disciplines*)
(*Only exhaustive and gated are implemented. The number of
    queues is determined from the length of this list.*)
In[904]:= serviceDisciplines={exhaustive,exhaustive};
In[905]:= n=Length[serviceDisciplines];
(*Arrival intensities*)
(*Arrivals *)
In[906]:= lambdas={1/10,1/10};
Table[Subscript[\[Lambda], i]=lambdas[[i]],{i,1,n}];
```

```
(*Routing probabilities*)
In[908]:= p[1,1] = 10/10;
p[1,2]= 1- p[1,1];

p[2,2] = 3/10;
p[2,1] = 1-p[2,2];
(*Service-time distributions*)
(*In order to compute the actual queue-length probabilities, we
     need to provide the distributions of the service times and
    switch - over times. Otherwise, the moments would suffice.*)
In[912]:= T1=1;
S1=100/10;
(*This is a trick to create deterministic distributions:*)
In[914]:= Bdists=Table[TransformedDistribution[T1,u\[
    Distributed]NormalDistribution[\[Mu],\[Sigma]]],{n}];
Sdists=Table[TransformedDistribution[S1,u\[Distributed]
    NormalDistribution[\[Mu],\[Sigma]]],{n}];
(*Moments of the service time distributions*)
In[916]:= EBs=Table[Moment[Bdists[[i]],1],{i,1,n}];
EB2s=Table[Moment[Bdists[[i]],2],{i,1,n}];
EB3s=Table[Moment[Bdists[[i]],3],{i,1,n}];
EB4s=Table[Moment[Bdists[[i]],4],{i,1,n}];
In[920]:= Table[Subscript[EB, i]=EBs[[i]],{i,1,n}];
Table[Subscript[EB2, i]=EB2s[[i]],{i,1,n}];
Table[Subscript[EB3, i]=EB3s[[i]],{i,1,n}];
Table[Subscript[EB4, i]=EB4s[[i]],{i,1,n}];
(*Moments of the switch-over time distributions (note:
    Subscript[S, 1] is switch-over from Subscript[V, 1] to
    Subscript[V, 2], regardless of the service disciplines)*)
In[924]:= ESs=Table[Moment[Sdists[[i]],1],{i,1,n}];
ES2s=Table[Moment[Sdists[[i]],2],{i,1,n}];
ES3s=Table[Moment[Sdists[[i]],3],{i,1,n}];
ES4s=Table[Moment[Sdists[[i]],4],{i,1,n}];
In[928]:= Table[Subscript[ES, i]=ESs[[i]],{i,1,n}];
Table[Subscript[ES2, i]=ES2s[[i]],{i,1,n}];
Table[Subscript[ES3, i]=ES3s[[i]],{i,1,n}];
Table[Subscript[ES4, i]=ES4s[[i]],{i,1,n}];
(*Moments of the busy period *)
(*Busy period equation*)
In[932]:= bpeqn=BPLSTtmp[i][\[Omega]]==BLST[i][\[Omega]+
    Subscript[\[Lambda], i](1-BPLSTtmp[i][\[Omega]])];
(*First moment*)
In[933]:= D[bpeqn,{\[Omega],1}]/.\[Omega]->0;
In[934]:= sol  =Table[Solve[%,BPLSTtmp[i]'[0]]/.BPLSTtmp[i
    ][0]->1/.(BLST[i]^\[Prime])[0]->-Subscript[EB, i],{i,1,n}]//
    Flatten;
In[935]:= Table[Subscript[EBP, i] = -BPLSTtmp[i]'[0]/.sol[[i
    ]],{i,1,n}]
Out[935]= {10/9,10/9}
(*Second moment*)
In[936]:= D[bpeqn,{\[Omega],2}]/.\[Omega]->0;
In[937]:= sol  =Table[Solve[%,BPLSTtmp[i]''[0]]/.BPLSTtmp[i
    ][0]->1/.(BLST[i]^\[Prime])[0]->-Subscript[EB, i]/.BLST[i
    ]''[0]->Subscript[EB2, i]/.(BPLSTtmp[i]^\[Prime])[0]->-
```

```
    Subscript[EBP, i],{i,1,n}]//Flatten;
In[938]:= Table[Subscript[EBP2, i] = BPLSTtmp[i]’’[0]/.sol[[i
    ]],{i,1,n}]
Out[938]= {1000/729,1000/729}
(*Third moment*)
In[939]:= D[bpeqn,{\[Omega],3}]/.\[Omega]->0;
In[940]:= sol  =Table[Solve[%,BPLSTtmp[i]’’’[0]]/.BPLSTtmp[i
    ][0]->1/.(BLST[i]^\[Prime])[0]->-Subscript[EB, i]/.BLST[i
    ]’’[0]->Subscript[EB2, i]/.BLST[i]’’’[0]->-Subscript[EB3, i
    ]/.(BPLSTtmp[i]^\[Prime])[0]->-Subscript[EBP, i]/.(BPLSTtmp[
    i]^\[Prime]\[Prime])[0]->Subscript[EBP2, i],{i,1,n}]//
    Flatten;
In[941]:= Table[Subscript[EBP3, i] = -BPLSTtmp[i]’’’[0]/.sol[[i
    ]],{i,1,n}]
Out[941]= {40000/19683,40000/19683}
(*Fourth moment*)
In[942]:= D[bpeqn,{\[Omega],4}]/.\[Omega]->0;
In[943]:= sol =Table[Solve[%,BPLSTtmp[i]’’’’[0]]/.BPLSTtmp[i
    ][0]->1/.(BLST[i]^\[Prime])[0]->-Subscript[EB, i]/.BLST[i
    ]’’[0]->Subscript[EB2, i]/.BLST[i]’’’[0]->-Subscript[EB3, i
    ]/.BLST[i]’’’’[0]->Subscript[EB4, i]/.(BPLSTtmp[i]^\[Prime])
    [0]->-Subscript[EBP, i]/.(BPLSTtmp[i]^\[Prime]\[Prime])[0]->
    Subscript[EBP2, i]/.(BPLSTtmp[i]^(3))[0]->-Subscript[EBP3, i
    ],{i,1,n}]//Flatten;
In[944]:= Table[Subscript[EBP4, i] = BPLSTtmp[i]’’’’[0]/.sol[[i
    ]],{i,1,n}]
Out[944]= {6200000/1594323,6200000/1594323}
(*Exact Analysis*)
In[945]:= Table[Subscript[\[Sigma], i][0]=1,{i,1,n}];
Table[Subscript[\[Beta], i][0]=1,{i,1,n}];
Table[Subscript[\[Pi], i][0]=1,{i,1,n}];
Table[Subscript[\[Sigma], i]’[0]=-Subscript[ES, i],{i,1,n}];
Table[Subscript[\[Beta], i]’[0]=-Subscript[EB, i],{i,1,n}];
Table[Subscript[\[Pi], i]’[0]=-Subscript[EBP, i],{i,1,n}];
Table[Subscript[\[Sigma], i]’’[0]=Subscript[ES2, i],{i,1,n}];
Table[Subscript[\[Beta], i]’’[0]=Subscript[EB2, i],{i,1,n}];
Table[Subscript[\[Pi], i]’’[0]=Subscript[EBP2, i],{i,1,n}];
Table[Subscript[\[Sigma], i]’’’[0]=-Subscript[ES3, i],{i,1,n}];
Table[Subscript[\[Beta], i]’’’[0]=-Subscript[EB3, i],{i,1,n}];
Table[Subscript[\[Pi], i]’’’[0]=-Subscript[EBP3, i],{i,1,n}];
Table[Subscript[\[Sigma], i]’’’’[0]=Subscript[ES4, i],{i,1,n}];
Table[Subscript[\[Beta], i]’’’’[0]=Subscript[EB4, i],{i,1,n}];
Table[Subscript[\[Pi], i]’’’’[0]=Subscript[EBP4, i],{i,1,n}];
In[960]:= zi=Table[Subscript[z, i],{i,1,n}];
In[961]:= hi=Table[ If[serviceDisciplines[[i]]===gated,
    Subscript[\[Beta], i][\!\(
\*UnderoverscriptBox[\(\[Sum]\), \(j = 1\), \(n\)]\(
\*SubscriptBox[\(\[Lambda]\), \(j\)] \((1 -
\*SubscriptBox[\(z\), \(j\)])\)\)\)],If[serviceDisciplines[[i
    ]]===exhaustive,Subscript[\[Pi], i][\!\(
\*UnderoverscriptBox[\(\[Sum]\), \(j = 1\), \(n\)]\(If[j != i,
\*SubscriptBox[\(\[Lambda]\), \(j\)] \((1 -
\*SubscriptBox[\(z\), \(j\)])\), 0]\)\)]]],{i,1,n}];
gi=Table[Subscript[\[Sigma], i][\!\(
```

```
\*UnderoverscriptBox[\(\[Sum]\), \(j = 1\), \(n\)]\(
\*SubscriptBox[\(\[Lambda]\), \(j\)] \((1 -
\*SubscriptBox[\(z\), \(j\)])\)\)\)],{i,1,n}];
(*Joint Queue-length Distribution at polling epochs*)
(*The branching property is used here and an implicit equation
    for LB^(Subscript[V, 1a]) is formed*)
In[963]:= md[i_]:=Mod[i-1,n]+1

LVa[1][zi]=.;
joint=RotateRight@Table[
LVb[i][zi]=LVa[i][zi]/.Table[Subscript[z, k ]->  \!\(
\*UnderoverscriptBox[\(\[Sum]\), \(j = 1\), \(n\)]\(p[k, j]*\
\*SubscriptBox[\(z\), \(j\)]\)\),{k,1,n}];
LSa[i][zi]=LVb[i][zi]/.Subscript[z, i]->hi[[i]];
LSb[i][zi]=LSa[i][zi]/.Table[Subscript[z, k ]->  \!\(
\*UnderoverscriptBox[\(\[Sum]\), \(j = 1\), \(n\)]\(p[k, j]*\
\*SubscriptBox[\(z\), \(j\)]\)\),{k,1,n}];
LVa[md[i+1]][zi]=LSb[i][zi]gi[[i]],{i,1,n}];
joint//ColumnForm
LVa[1][zi]=.
During evaluation of In[963]:= Unset::norep: Assignment on LVa
    for LVa[1][{Subscript[z, 1],Subscript[z, 2]}] not found.
Out[966]= LVa[1][{Subscript[\[Pi], 1][1/10 (1-(7 Subscript[z,
    1])/10-3/10 ((7 Subscript[z, 1])/10+3/10 Subscript[\[Pi],
    2][1/10 (1-Subscript[z, 1])]))],7/10 Subscript[\[Pi],
    1][1/10 (1-(7 Subscript[z, 1])/10-3/10 ((7 Subscript[z, 1])
    /10+3/10 Subscript[\[Pi], 2][1/10 (1-Subscript[z, 1])]))
    ]+3/10 ((7 Subscript[z, 1])/10+3/10 ((7 Subscript[z, 1])
    /10+3/10 Subscript[\[Pi], 2][1/10 (1-Subscript[z, 1])]))}]
    Subscript[\[Sigma], 1][1/10 (1-Subscript[z, 1])+1/10 (1-(7
    Subscript[z, 1])/10-3/10 Subscript[\[Pi], 2][1/10 (1-
    Subscript[z, 1])])] Subscript[\[Sigma], 2][1/10 (1-Subscript
    [z, 1])+1/10 (1-Subscript[z, 2])]
LVa[1][{Subscript[\[Pi], 1][1/10 (1-(7 Subscript[z, 1])/10-(3
    Subscript[z, 2])/10)],3/10 ((7 Subscript[z, 1])/10+(3
    Subscript[z, 2])/10)+7/10 Subscript[\[Pi], 1][1/10 (1-(7
    Subscript[z, 1])/10-(3 Subscript[z, 2])/10)]}] Subscript[\[
    Sigma], 1][1/10 (1-Subscript[z, 1])+1/10 (1-Subscript[z, 2])
    ]


In[968]:= eqn=LVa[1][zi]==First[joint];
ones=Table[1,{n}];
LVa[_][ones]=1;

solution1stMoments=Flatten@Solve[D[eqn,{{zi}}]/.Table[Subscript
    [z, i]->1,{i,1,n}]];
In[972]:= solution2ndMoments=Flatten@Solve[D[D[eqn,{{zi}}],{{zi
    }}]/.Table[Subscript[z, i]->1,{i,1,n}]/.solution1stMoments];
solution3rdMoments=Flatten@Solve[D[D[D[eqn,{{zi}}],{{zi}}],{{zi
    }}]/.Table[Subscript[z, i]->1,{i,1,n}]/.solution1stMoments/.
    solution2ndMoments];
In[974]:= Table[LVa[i,j]=LVa[i][zi]/.Table[Subscript[z, k]->If[
    j==k,z,1],{k,1,n}],{i,1,n},{j,1,n}];
```

```
Table[LVb[i,j]=LVb[i][zi]/.Table[Subscript[z, k]->If[j==k,z,1],{k,1,n}],{i,1,n},{j,1,n}];
Table[LSa[i,j]=LSa[i][zi]/.Table[Subscript[z, k]->If[j==k,z,1],{k,1,n}],{i,1,n},{j,1,n}];
Table[LSb[i,j]=LSb[i][zi]/.Table[Subscript[z, k]->If[j==k,z,1],{k,1,n}],{i,1,n},{j,1,n}];
In[978]:= LVas=Flatten[Table[D[LVa[i,j],z]/.z->1,{i,1,n},{j,1,n}]]/.solution1stMoments]
LVbs=Flatten[Table[D[LVb[i,j],z]/.z->1,{i,1,n},{j,1,n}]/.solution1stMoments]
LSas=Flatten[Table[D[LSa[i,j],z]/.z->1,{i,1,n},{j,1,n}]/.solution1stMoments]
LSbs=Flatten[Table[D[LSb[i,j],z]/.z->1,{i,1,n},{j,1,n}]/.solution1stMoments]
Out[978]= {1733/505,1,464/303,124/101}
Out[979]= {4173/1010,3/10,3622/1515,186/505}
Out[980]= {0,230/303,1228/505,0}
Out[981]= {161/303,23/101,1228/505,0}
(*Joint Queue-length Distribution at arbitrary epochs *)
(*Calculate expected cycle length (by adding expected visit lengths and expected switch-over times)*)
In[982]:= \[Theta]i=Table[If[serviceDisciplines[[i]]===gated, Subscript[\[Beta], i][\[Omega]],If[serviceDisciplines[[i]]===exhaustive,Subscript[\[Pi], i][\[Omega]]]],{i,1,n}];
In[983]:= Vi = Table[LVb[i][zi]/.Table[If[i!=j,Subscript[z, j]-> 1,  Subscript[z, i ]-> \[Theta]i[[i]]]],{j,1,n}],{i,1,n}];
In[984]:= EVs = -D[Vi,\[Omega]]/.\[Omega] ->0/.solution1stMoments (* expected visit lengths *)
Out[984]= {1391/303,124/303}
In[985]:= ESs (* expected switch over times *)
Out[985]= {10,10}
In[986]:= EC = Total[ESs] + Total[EVs]  (* expected cycle length *)
Out[986]= 25
(*Joint queue-length distribution at arbitrary epochs L(z)*)
In[987]:= L[zi] =1/EC \!\(
\*UnderoverscriptBox[\(\[Sum]\), \(i = 1\), \(n\)]\((
\*FractionBox[\(
\*SubscriptBox[\(z\), \(i\)]*\((\(LVb[i]\)[zi] - \ \(LSa[i]\)[zi])\)*\((1 -
\(\*SubscriptBox[\(\[Beta]\), \(i\)]\)[
\*UnderoverscriptBox[\(\[Sum]\), \(j = 1\), \(n\)]
\*SubscriptBox[\(\[Lambda]\), \(j\)] \((1 -
\*SubscriptBox[\(z\), \(j\)])\)])\)\), \(\((
\*SubscriptBox[\(z\), \(i\)] -
\(\*SubscriptBox[\(\[Beta]\), \(i\)]\)[
\*UnderoverscriptBox[\(\[Sum]\), \(j = 1\), \(n\)]
\*SubscriptBox[\(\[Lambda]\), \(j\)] \((1 -
\*SubscriptBox[\(z\), \(j\)])\)])\)*\(
\*UnderoverscriptBox[\(\[Sum]\), \(j = 1\), \(n\)]
\*SubscriptBox[\(\[Lambda]\), \(j\)] \((1 -
\*SubscriptBox[\(z\), \(j\)])\)\)\)] +
\*FractionBox[\(\(LSb[i]\)[zi] - \(LVa[md[i + 1]]\)[zi]\), \(
```

```
\*UnderoverscriptBox[\(\[Sum]\), \(j = 1\), \(n\)]
\*SubscriptBox[\(\[Lambda]\), \(j\)] \((1 -
\*SubscriptBox[\(z\), \(j\)])\)\)])\)\);
(*Marginal queue lengths*)
In[988]:= Table[L[i]=L[zi]/.Table[Subscript[z, k]->If[i==k,z
    ,1],{k,1,n}],{i,1,n}];
In[989]:= EMLs=Table[D[Series[L[i],{z,1,3}],z]/.z->1/.
    solution1stMoments/.solution2ndMoments,{i,1,n}]
Out[989]=
    {5410514707061/2459012256000,1524833143339/2459012256000}
(*Correlation (2 queues) *)
Lz  = Series[L[zi],{Subscript[z, 1],1,3},{Subscript[z,
    2],1,3}];
In[991]:= EZ1 = SeriesCoefficient[Lz,{1,0}]/.solution1stMoments
    /.solution2ndMoments;
In[992]:= EZ2= SeriesCoefficient[Lz,{0,1}]/.solution1stMoments
    /.solution2ndMoments;
In[993]:= EZ1Z2 = SeriesCoefficient[Lz,{1,1}]/.
    solution1stMoments/.solution2ndMoments/.solution3rdMoments;
VarZ1 = (2*SeriesCoefficient[Lz,{2,0}]/.solution1stMoments/.
    solution2ndMoments/.solution3rdMoments )+ EZ1 - ((EZ1)^2);
VarZ2= (2*SeriesCoefficient[Lz,{0,2}]/.solution1stMoments/.
    solution2ndMoments/.solution3rdMoments )+ EZ2 - ((EZ2)^2);
In[996]:= Cor = (EZ1Z2-EZ1*EZ2)/(Sqrt[VarZ1]*Sqrt[VarZ2])
Out[996]= -(893001510383280110806412623/Sqrt
    [62079463186633570675272232767854224580878610836333019329])
In[1000]:= N[Cor]
Out[1000]= -0.113339
```